The COMPUTER JOURNAL

Programming - User Support
Applications

Issue Number 63 September/October 1993 US$4.00

SCSI EPROM Programmer part II

Z-System Corner

DR S-100

Real Computing

Support Groups

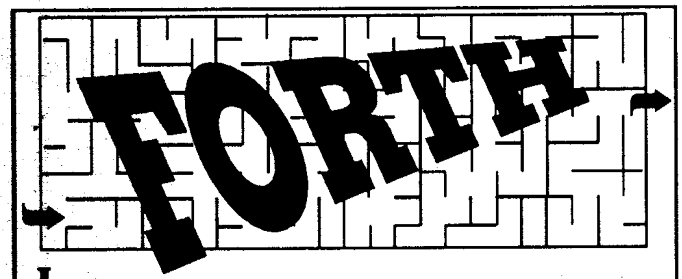
IDE Drives part II

6809 Operating Systems

Multiprocessing Part II

Mr. Kaypro

ISSN # 0748-9331


FORTH

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.

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

Subscription rates within the US: $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 63 September/October 1993

EDITOR'S COMMENTS

Welcome to issue #63. Chock full of good items for your reading pleasure. On time again as well! Our stable of regulars is back in force and we have more installments on our ever going projects list.

The Reader to Reader section is busting with items that are asking for help, and giving some as well. Robert Edgecombe gives some tips on MOVing CP/M. John Baker gives some insights to using CP/M on an Apple. ZX81 schematics are here by the handful, thanks to Paul (letter on page 7) and others. Next issues Reader to Reader will have letters pertaining to using Small-C.

JW Weaver provides some User Group names and addresses, as well as requesting your help in finding recent information on many items from his own collection of computers.

Jay finally fulfills his promise of explaining how he made a system perform non-stop testing even after the power goes off. Intended for PC based systems using 4DOS, Jay shows how he did it and give help in trying it on Z-Systems.

Herb Johnson starts explaining BIOS programming by covering how disk drives work and what the BIOS software must do to get data to or from the drive. This is a must read for first time hackers!

Rick Rodman comments on code quality and especially C compiler output. This is part of Rick's answer to TCJ choosing a standard language for our articles. Rick's and other Small C letters will be the focus of next times Reader to Reader column.

Chuck Stafford helps you keep those Kaypro's running by using clone power supplies. Sharpen up those hardware skills with this simple and easy to do project.

The long awaited second part of Tilmann Reh's IDE interface article is now here. In this part Tilmann explains the interface standard and starts talking about IDE registers. His next part will focus on the programming aspect, so pay attention and get these ''Basics'' under your belt.

Terry Hazen completes his SCSI EPROM programmer article, with samples of the software. You will see just how simple it can be done as you read this final (I hope not final from Terry) article.

Operating Systems part II is here with a review of 6809 operating systems. Flex, OS-9, and SK*DOS are reviewed as well as listing some vendors who still have 6809 products to sell. Since SK*DOS is the only actively supported operating system for 6809, I have added a listing of programs that can run on the 68K version. You will be surprised at just how many programs are available!

Brad Rodriguez provides part II of his multiprocessing article. He asks you to comment on your interest in buying bare boards for a ''Scroungemaster II'', all coverage starting on page 40.

Frank Sergeant comments and responds to your letters about his ''Reminiscing and Musings'' in issue #62. I think that Frank's comments on making boards, prompted the folks at Techniks to start advertising. Check-out their market place ad on the inside back cover for ''PCB's in Minutes.''

Pulling the last page as usual is my Computer Corner. A bit short due to so many good articles this time, but I thank those who sent me ZX81 schematics. You will find out I even got one in German. I prove that simple systems can be very popular and even fun to use...well at least fun to write about.

Please note that issues 20 to 25 are now available as Volume 3. There still remain some single issues, but their number is pretty low. Issue number 25 is available, but again in limited quantity. I have already made a Volume 4, Issues 26 to 31, but we have plenty of those issues still left.

Issue #63 is big and full of interesting reading. So if you don't find anything of interest, it only means that you haven't written to TCJ yourself. This magazine is based on your input and comments. So Send those letters and comments to:

The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535
or
B.Kibler@GEnis.com
B.Kibler on GEnie
Compuserve as 71563,2243

READER to READER

Letters to the Editor

All Readers

MINI Articles

Dear Bill,
I subscribe to Computer Journal
Can you help? This may sound stupid, and it is. There is NO Library over here, (Not even the British Library) who has more than the odd issue of:-

Microsystems
Interface Age
etc.

When reading through the back issues which you sent I come across references to articles in these two. I am stuck! I cannot read them.

I know copyright and all that, but if the publishers have ceased there is a real problem.

Is there any chance that you have, and can copy and send to me for my private study and research (I think that voids copyright problems):- (from TCJ No 49 p28). Microsystems Vol 4 No 9 Sept 1983 p 86 onward; 10 Oct, p 114; Vol 5, 1 Jan 1984, p 120. Three part article "Relocating assemblers and linkage editors". Micro/Systems Jl Vol 1 No 3 July/Aug 1985 p26. "Structured Programming with M80". If you cannot, could you pass request on to the author (I assume HE must have a copy!)

Similarly:-
INTERFACE AGE Vo12 Oct, Nov, Dec 1976, by R Edelson "Super Chip FD1771". Even, if that is no route, do you know any library who you could get copies from and then charge me (with postage and your costs!!).

I know it is a pain in the **, but I was barn in the sticks...and have to try to cope. On the longer view:- Can you insert a request for back issues of these magazines: Interface Age
Microsystems
Micro/systems Jl
with my address? I would purchase, and then donate to our national reference Library.

Thanks very much for the insert re ADM3A and 5 circuits. No replies at all. Might as well junk them, in the circumstances. I expect everyone thought "someone else will write, so I won't bother". People are humans, after all!

Yours truly
John Butler
16, Uphill Drive
London, NW9 0BU, United Kingdom

Gee John, England sounds worse off than I thought. Hopefully some of our other English readers will see this and help you out. If not let me know (a post card will do) and I will try and find those issues for you.

I must admit that yours is a typical problem. I have been after my writers to be more explicit about their references for people in your situation. You are right that copying magazines for your own use is OK, and If I happen to have the issues you need (which I doubt) I will gladly copy them for you. What you need however is someone or group in England to act as a clearing house of information. We have many old club/groups in the U.S. that have retained their old magazines for purposes like yours. Maybe there are some in your area and you just don't know it?

One of TCJ's problems has been the lack of good representation of European problems and solutions. I am trying to find a writer who wants to comment on what is happening over there. Might you be interested? Should you find a person while getting help on your back issues, that is fairly close to the going on's over there, please send them our way. Hope you get what you need, soon. Thanks again. Bill Kibler.

Dear Bill;
I ran a cursory count of the availability of earlier-released Intel-based computers offered for sale in the August Computer Shopper: I counted four ads for '286 machines and one for a second hand '88. Ads for '386s were also relatively few, compared to numbers of '486s offered.

Shouldn't you be considering adding the early '86 varieties to your stable of "Classic" one-lung Studebakers for which you publish tips on hacking buggy-whip holders? Or restrict yourself strictly and "officially" to 4 and 8 bit computers, of course dropping the Apple][-gs?

Over the last few years, as a PET and Sinclair graduate currently running a '386, I 've been amused by and enjoyed your fanzine, though I've never implemented any of the code or hardware hacks. However, I recently discovered the prozine "PC Techniques", and have found more usable code hacks in one issue than in years of your magazine. Granted, they still don't tell me about hardware hacks that require soldering my ALR '386 motherboard, but even paranoid Steve C. seems leary of that level of sophistication.

The real purpose of this note is to tell you that I'm going to let my subscription expire with issue #65, so don't waste your limited money and time importuning me for a renewal. So long, it's been amusing, but even us 70-year olds have to grow up sometime.

Sincerely yours,
Dave English

Well Dave, you don't happen to actually have a "one-lung Studebaker" do you? I know some auto collectors who would pay many times the cost of a new car for one! The more time that goes by the more it seems like the computer industry is just like the car industry.

I suppose one of the reasons they are getting more similar is the same marketing plan. That plan is based on obsoleting perfectly good items in favor of the latest and greatest. Along with that is convincing the buyer that you would be crazy to drive one of those! Well like any industry there are plenty of people who like to know what actually is happening under the "hood". TCJ has always written for those wanting to know what happens inside.

Since the modern PC has nothing inside that the average person could understand, TCJ has little reason to write about them. At some point in the future, I am sure we will occasionally give some space to articles about PC's. The fact is, I have an original IBM PC (256K) motherboard that would qualify for being a true collectible item (and is becoming more valuable daily).

I love your comment about Steve C. staying away from PC's, it proves my point, nobody with any real hardware knowledge wants anything to do with "that level of sophistication." Although I am sure many of our readers would liken PC's to Ramblers and Mercedes (the PC being like the Rambler!)

My hopes over the next few years is for TCJ to pickup about the same percentage of readers as there are people who collect old fashioned cars. Out of the millions of people who drive, I guess that one percent are interested in collectible cars. If one percent of computer users are interested in collecting older computers, then TCJ could have as many as 10 or 30 thousand readers in the future!

So Dave, sorry to see you go, but then you know where to find us when you get tired of those new fandangled PC's. Bill.

Dear Bill;

Please find enclosed my check for $57.34 to cover the cost of completing my TCJ library all the way back to issue #1.

My issue #62 is missing (1 llxl7 sheet) pages 7,8,45,& 46 . Please send the missing sheet.

In Reader to Reader #62. regarding James M. Harper and the Royal alpha Tronic. I think that I have some answers.

1) For DRI manuals see Elliam Associates advertisement on back cover. I have one of those. They are NEW and as complete as any that DRI printed. The HAYDEN book titled "CP/M REVEALED" by Jack D. Dennon, ISBN 0-8104-5204-9 is very good. It also includes the use of MOVCPM.

2) The command line parameter that follows MOVCPM must be a decimal whole number representing the top of memory in Kbytes. On one system that I have, the parameter must be an even number but on the XEROX it can be an odd number like 59. The first line is a BOOT screen print. The second line is the typed in MOVCPM command line. The 3rd, 4th, & 5th line are written to the screen by the running MOVCPM program.

```
Xerox 60k CP/M vers 2.2c #2-294 EC3001096
A)MOVCPM 59
Constructing Xerox 59k CP/M vers 2.2
Ready for "SYSGEN" or
"SAVE 34 CPM59.SYS"
A)
```

The XEROX does a ROM load into RAM memory F000h-FFFFh of a package called BOS "basic operating system". This BOS serves as a monitor with commands that let you read a particular disk sector, etc. The L command loads (boots) from any disk drive (default A) and the BOS stays put, then becomes a 4K BIG BIOS. Many systems use F800h-FFFFh for a 2K BIG BIOS. The BIOS from the system tracks is small, the standard is the last 7 sectors of track 1, plus 280h of RAM as a scratch pad. For some functions it simply is a jump vector to big bios. Even though this is a 64K system, for MOVCPM it is a 60K system. The top of memory is the first word address of BIG BIOS. This makes a very fast boot.

On the CCS (California Computer Systems) S-100 system, on coldstart, the CCP auto loads the BIG BIOS from a named RLOCBIOS.COM file on the boot disk. RLOCBIOS.COM is a combination of a RELOADER and a E00h (3.5K) BIG BIOS which loads at 100h like any X.COM file, then relocates the bios portion at F200-FFFFh overlaying the BOOT BIOS. Here top of memory (end of BDOS + 600h) is 62K (F200h + 600h = F800h). CCS modified MOVCPM so that the MOVCPM command line parameter is 64. The CCS manual says that all the system addresses are the same as if an unmodified MOVCPM were set to 62k. This MOVCPM carries a double copyright notice, CCS & DRI both. Here an odd number like 57 will give an error notice. By comparison, this is a slow boot.

A horrible MOVCPM error, undocumented by DRI, comes when you copy MOVCPM from the wrong distribution disk, a CCS disk on the XEROX, an example follows. MOVCPM is loaded and starts running, as expected, it writes line 2, & 3 to the screen then it hangs at the end of the line where I put the asterisk. I have 2 CCS distribution disks, with different serial numbers. This even happens on the CCS computer when I cross those 2 CCS disks.

```
A)G:MOVCPM 56
CONSTRUCTING 56k CP/M vers 2.2
SYNCHRONIZATION ERROR*
```

The software reset CTRL-ESC does not work, it takes the hardware reset button in back of the monitor to get back to running. The RUNNING OPERATING SYSTEM and MOVCPM must come from the same distribution disk. This MOVCPM & OPERATING SYSTEM cross serialization is documented in the book "INSIDE CP/M" by David E.

Cortesi ISBN 0-03-059558-4 page 268.
The MOVCPM COMMAND:
SERIAL NUMBER CHECKS ....

MOVCPM & SYSGEN are real MISNOMERS. SYSGEN does not generate anything, it simply reads the system tracks and writes to the sysgen area of memory and visa versa. Likewise MOVCPM generates a fresh copy of the system (Boot loader, CCP, BDOS, & BIOS) and writes it to the sysgen area of memory, with all addresses co-ordinated to the so called top of memory that you type in on the MOVCPM command line.

In the book ZCPR3 THE MANUAL by Richard Conn ISBN 0-918432-59-6 the sysgen area of memory is shown as starting at 1100h not the usual 900h, and he cautions BEWARE, all systems are not the same! If this book is not available, this same ZCPR3 installation information is shown on the public domain disks SIG/M 184-192.

The CCS manuals also describe non standard system disks, those where the system tracks hold big bios. This can be done on some single density formats and all double density formats. But it takes some some special work like writing the high byte rounded up of the bios last load address into 929h in the boot loader program. The CCS has 12 different formats, while the XEROX
has only 4. There is some more to it also, but on the XEROX there is no such information that I can find.

Do you have any information on where if anywhere that I can find XEROX 16/ 8 support.???

Sincerely yours,
Robert L. Edgecombe

Thanks for all the good information, Bob. I had forgotten to mention the serial number information. However I found the locations inside the programs and have just zeroed my serial numbers out. That way I can solve the "Synchronization" problem and can just forget about it for good.

If you want to find the serial number, just do some file compares between your two different SYSGEN and MOVCPM programs and they should pop out. If readers are really interested, I am sure I can find the approximate locations in DOS and MOVCPM. I remember it being near the end of the DOS program portion.

Using a RLOCBIOS is also a good way to overlay a changed BIOS to see if all your improvements are going to work. That way you know the system is up and running, and then get to test just that new part. Also beware that some systems, add or subtract from the value given to MOVCPM. I can't remember for sure which system, but one of mine, changes the 64K value to 62K (I believe the CCS when using RLOCBIOS does it) to make up for the extra large BIOS.

The variation comes again from what Bob indicated, with so many different possible disk formats, the vendor had plenty of freedom to implement how the system went together. You need to remember that many vendors had the complete source code to many of the utilities and even all of CP/M. That means they could make their system even more of a one of a kind monster.

Unfornately Bob, I have little information on the Xerox 8/16. My schematics contain little help fo. you. Possibly some readers will send us some good information, just like your help with MOVCPM. Thanks again! Bill Kibler.

Dear Mr. Kibler:
Where do I begin? I suppose that first, I should thank you, the past (and future!) editors and all the contributors to *TCJ* for making one of the few remaining computer-oriented magazines that's relevant to me. I read each issue voraciously from cover-to-cover within a few hours of receiving it and keep them handy for reference because it usually turns out that I'm involved in something that was discussed in a recent issue!

I certainly want that sort of thing to continue, so I've enclosed a check in the amount of a two-year subscription which I beleive your records should show will continue beginning with issue #64. I also wish to purchase a number of back issues which I have listed separately.

At this point, I'd like to tell you a little (a lot?) about myself as it relates to **our** journal.

I would guess that at 26 I'm likely to be one of your (if not *the*) youngest subscriber(s). I've been enthusiasically using Apple-hosted CP/M and Z-System computers ever since I encountered the PCPI Appli-Card in 1984 and then truly discovered it and CP/M in the summer of 1987. When I started back to school in the fall of 1988, I soon discovered that I wanted my trusty Franklin ACE 1000 and PCPI AppliCard with me. I wrote all my papers and reports with it and soon purchased a modem and started using the campus computers from my apartment. I was sold on these things!

I was concerned that I was depriving my younger brother of a computer at home by having it at school, so in the summer of 1989, I purchased an extended keyboard Apple //e, a pair of Apple UniDisk 3.5 disk drives, a monochrome composite monitor, and an Epson LQ510 printer. I moved the AppliCard over to my new machine and took off! I added a Sorrento Valley Associates ZVX4 8" disk controller in 1991 and quickly hacked cabling for attaching a pair of TEAC FD55-GFR high-density 5.25" disk drives to it. I replaced my Shugart SA801s with a pair of SA860s so that I could go for double-sided disks. (I gave the 801s to a fellow member of our remaining CP/M-Houston User's Group.)

All the while, I had been keeping tabs on (and drooling over) a new Z-System processor card for Apple][computers called the CardZ180. I entered correspondence with one of the people behind it and, after a two-year wait, finally received one. It took another 5 months before I got working system software (original disks were corrupt), but on 13 May 1992, my CardZ180 finally came on line and I have been extremely pleased with it. I purchased my first hard disk in August of 1992 and am quite satisfied with it.

Along the way, I acquired an Epson QX-10 with 10MB hard disk both of which died and then were resurrected with e-mail help of (contributors) Wayne Sung and Jay Sage, respectively. I picked up a couple of extra AppliCards, so the Franklin (which my parents still have) can continue as a Z-System machine. I also became enamored of the Amiga computers and recently purchased a used Amiga 500 and have proceeded to upgrade it with more RAM, the common hacks, hard disks, and System 2.1 software.

As for what I use my machines for, I use my CardZ180-equipped Apple //e for *everything* that I do. Right now that mostly consists of writing papers and reports for school and telecommunication (mostly internet news, mail, and other interactive activities). Right now the Amiga is, alas, mostly an entertainment machine, but all three of my computers are the object of my fascination and personal study. My plans for the Amiga involve using TeX and doing CAD work.

As a still fledgling hacker, I finally wrote my 'Opus 1' around July of 1992. It's an extension for the CardZ180 CBIOS--an alternate set of CONIN, CONOUT, and CONST routines that use the CardZ180 (HD64180) ASCI1 port for the console rather than the standard Apple console driver. The .COM file determines the type of system it's running in (full NZCOM, static ZCPR3 or plain ZCPR1 and saves the old CBIOS jump addresses and replaces them with the run-time addresses of the new routines which it places in a reserved area at the top of memory. This finally solved the problem of character loss in telecommunications that all Apple-hosted CP/M systems have, although the CardZ180 system's threshhold is around 4800bps rather than 1200bps for other systems (Appli-Card included). I packaged it up and uploaded it to the BBS that supports the CardZ180.

I was extremely pleased when my external terminal package caught the attention of the author of the CardZ180's system software. He very graciously offered me the source code for the whole shooting match and I quite naturally accepted. I recently acquired exactly the right combination of hardware to kluge together a hard-disk to send him so that he can send the huge volume of code (in 6502, 'c02, 'c802, 'c816, Z80, HD64180 assembly, PASCAL-MT+, and Turbo Modula-2 Z80) to me. I received it back in April 1993 and after examining the code a bit, implemented some enhanced functions in the standard Apple console driver which had been missing since the days of the PCPI Appli-Card!

Projects I've undertaken include a second degree. I completed a degree in Agricultural Engineering from Texas A & M University in December of 1992 and now have turned around and am studying Computer Engineering. I'm enjoying it immensely, but recently I taught myself more in two hours alone with my Z80 databook and my Apple //e Technical Reference Manual than I've learned in the first 8 weeks of my introductory digital design class! I suppose I should try to attach myself to the work of a professer where I can put my interests to use. I discovered that I'm probably searching for a mentor and the school's probably the best place to find one.

What I taught myself was all about DRAM and the Z80. At this point, I would like to ask you and all readers for a recommendation for sources of fast (100ns or less) DRAM that will operate with 128-cycle refresh for high-speed Z80 applications. My project is to accelerate my remaining PCPI AppliCards and I need the right DRAM for it to work. The Applicard circuit design is virtually bulletproof and has been run as fast as 12MHz. I plan to take my boards as high as I can get them to go! True, the systems will then begin to become I/O bound, but as long as there's no I/O to be done, they will fly.

I hope eventually to build a couple of the projects from the *Ciarcia's Circuit Cellar* books with the serial EPROM programmer (vol. 6) being among the first (if not THE first). I feel I have more to learn before I tackle one of those though.

Another thing I hope to do is get in touch with Hal Bower, et al. and work on implementing the Banked/Portable BIOS on my CardZ180. The current system already reserves an extra 64K bank of the 64180's 1MB address space for system expansions such as the B/P BIOS and I'd like to see if I can get it up on the CardZ180.

I would like to express my appreciation for David Goodenough's article on Z80 interrupts in issue #54. I soon adapted his interrupt-driven serial I/O and queue code to my copies of QTERM on my CardZ180. This finally allowed high-speed terminal operations with the CZ180's Apple console driver. I adapted just the queue portion for Appli-Card and QX-10 QTERM overlays and it operates quite nicely for regular polled I/O (it does help a bit). I had also recently studied queues in my data structures class so re-reading the article and implementing the code added greatly to my understanding.

I was also quite excited when I turned to the new Centerfold in issue #59. I saw things in the schematic that I had recently been tested over in my digital design class. I couldn't help but smile broadly and think, "I *know* what that is! I can even name the chip!" I looked at the parts list and confirmed that I was absolutely correct. I showed my copy of issue #59 to my professor and related what I experienced. He seemed quite pleased that I had made the class material "real" for myself, but offered no comment on *TCJ* itself. Perhaps I should dare to lend him one of mine to peruse.

Gosh, this letter has gotten long! Perhaps it might make a mini article! Seriously, though, I don't see much in the way of Apple][-hosted CP/M and Z-System machines represented here or in *The Z-Letter* and if I can make myself sit down and write things down, I'd be more than happy to supply an article dealing with CP/M and Z-System running on co-processor cards in Apple][machines. Thank you again for such a fantastic publication!

Yours truly,
John D. Baker
jdb8042@blkbox.com
jdbaker@taronga.com
jdb8042@tamuts.tamu.edu

Thanks for the 8 inch disk and your letter that was on it. Had no problem moving it off and accross several systems after I found the correct RS232 connector on the back of my S-100 system. Forgot had two dead sockets, but will leave the cable on the right one from now on, so transferring will be easier next time.

I have gotten several requests for apple //e articles, but as yet no offers from any writers. Your Epson QX-10 however is a real hot item. Seems everyone loves them, but can't find any support. I would love to print information about the QX-10 or Appli-Cards.

You are right in thinking that you have enough for a mini article. Your letter mentioned many items which our readers are interested in, but you didn't give enough details. I wonder if these card are still for sale, who has them, what comes with them, what restricitons/options apply? How did you install the Z-System, and what problems did you encounter and our writers helped you solve.

Quite often our writers help readers solve problems, but the problem and solutions never seem to make it to print. I have asked our writers to comment in their columns, and they have started doing that, but readers need to send me mini-articles as well. The reason being, our writers often do not know what the final outcome was and therefore are not sure the problem was solved. So it actually comes down to the reader sending me a letter/mini-article after the problem is solved as the only way we can get the information into print. You must remember, you are the only one who really has all the facts!

So that is it John, it is up to you to tell us what the problems were, and how our writers helped you find your way out of the darkness. I might also say, that if you drop me your professor's name, I'll send him/her a few copies for free. I still think TCJ is the best publication for college students and professors! Thanks again for the renewal and letter. Bill.

Dear Bill
I'm enclosing my check for another 12 issues of TCJ, having been a happy (if uncommunicative) subscriber since issue #1.

Does anyone out there remember (maybe is still using) the IMP-16 computer Hal Chamberlin designed back in the 70s, using the National Semiconductor chip set? I believe it was the first 16-bit computer for the amateur. I had a lot of fun wire wrapping his boards, and learned a lot (mostly now forgotten) on how the circuitry worked. Hal had a good way of explaining the details of such things in a lucid and entertaining fashion. Unfortunately, he went on to other things before completing his series of articles in "The Computer Amateur", later reprinted and continued in National's "Bit Bucket", and I never did get my IMP-16 up and running.

I had better luck with the Slicer computer, using the Intel 80186, which served me well (ah, the thrill of powering up your handiwork and seeing something appear on the screen!). Running DOS 2.1 with Heath terminal, about all I could run was MASM and Multiplan, though. Again, I wonder if there is still interest in this machine.

Norman Stanley.

Thanks for the renewal Norman, and the interesting comments on the IMP-16. I have heard of it, but never saw one. How about some more details and explanation of the "thrills" you got doing it? WordStar will also run off a terminal in DOS 2.1 if you use version 3.1 (did it myself for awhile!). Thanks again for being with us since #1! Bill.

Dear Mr. Kibler:
I just received my copy of issue 61 of The Computer Journal, and saw your response to Ken about the availability of schematics for the venerable Timex Sinclair ZX-81. I am going to go one step farther, and am sending you a copy of the schematic diagrams that I have for both the ZX-81 and its predecessor, the Sinclair ZX-80.

Most interestingly, last week I was logged into the White Sands Public Domain software pool, SIMTEL20, and in the messy-dos directory under emulators and I find the index has an emulator for these machines. Maybe someone will download this emulator and give us a report on it. I can't because it will not run under either of my CP/M systems.

I also have a full set of diagrams for the Intertec Compustar VPU-30 system that I would be willing to share with TCJ for others if there is interest. However, those schematics must be returned to me, for I am using them to support my two Cstar systems. The beauty of them was the use of a hard drive (boat anchor size) that would be the home of software for the remote systems. Each system (255 possible) could access the hard drive and read/write to the main disk partition (D:) while being assigned a small hard disk partition (C:) that was not accessible to any other system on the network. I have a 10 meg hard drive for these systems, and supposedly, Intertec came out with a 96 meg drive.

Keep up the good work with The Computer Journal.

Sincerely, Paul V. Pullen.

Thanks Paul for the schematics and do read my corner for how I put my foot in it.

I have the SIMTEL-20 CDROM and also found all the emulators, quite a collection. Guess I need some people to review them all! I also have an Intertec system, a QD, has two Z80's. Very fast and well designed. I have heard about their multi-user system and think they were ahead of their times. Oh well, lots of great systems were passed over for the rather poor PC hardware design.

Thanks again! Bill Kibler.

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

Regular Feature

Classic Support

Group Reviews

SUPPORT GROUPS FOR THE CLASSICS

By JW Weaver

Kind of fell behind, what with rebuilding my main transport, and problems with my BBS system. But, now I need to get with the task at hand, writing this column.

Connected with a BBS located in Salt Lake City, Utah. The BBS is associated with a user group supporting the Coleco ADAM. Name of the group is Adam-Link User's Group, with a newsletter ADAM Informant. Inquires should be posted on the BBS. SLC ADAM-Link BBS, (801) 484-5114, speeds 300/1200/2400 with 8N1 or 7E1.

A second user's group, is the San Diego OS-9 Users Group. Contact person is Warren Hrach, (619) 221-8246 - voice, associated BBS is Ocean Beach BBS, (619) 224-4878 8N1 9600.

The fellow who sent the info on the OS-9 group, Shaun C. Marolf, also runs a BBS, Eight Bit Heaven, dealing with the Classics. A friendly BBS, handling several 8 bit systems.

Possessing several older systems, each with a different approach to the design of hardware, I would be interested in hearing any history on these systems. So if you have any knowledge, personal involvement in the development of the organizations or systems, Please share with us.

Some of the systems I have are, (with what information I already have, such as last listed address or company name);

ADAM - Coleco Industries, Inc., Amsterdam, New York 12010

Altair 8800 - MITS Inc. 6328 Linn, N.E., POBox 8636, Albuquerque, N.M. 87108. 8080 processor with 4 kilobytes of memory, 8" floppy drive

Attache - Otrona Corp. 4755 Walnut, Boulder CO. 80301. Z80 with 2 half height 5 1/4 floppy drives, built in 5" crt

ECS 4500 - ECS Microsystems, Inc. Z80 with 2 full height 5 1/4 floppy drives, built in 9" crt

Focvs XVI - Fairchild Camera and Instrument Corporation. 16 bit descrete logic processor with programmable microcode

Hyperion - Dynalogic Info-Tech Corporation, Ottawa, Canada. 8088 with 2 half height 5 1/4 floppy drives, built in 5" crt

Micro Decision - Morrow Designs. Z80 with 2 two/thirds height floppy drives

Osborne I - Osborne Computer Corporation, Hayard, CA. Z80 with 2 half height drives, built in 4" crt

Super ELF - Quest Electronics, POBox 4430, Santa Clara, CA 95054

RCA 1802 processor with 256 bytes on static ram

I would also like to obtain any technical documents relating to these systems and especially any support groups or newsletters. Remember that some of these organizations are no longer in business, but often they sell the rights to other companies or service centers. IMSAI's were continued to be made and supported for several years after IMSAI went under by Fischer, Fischer and Company, (itself closed in 1982, I think).

I am planning to research as much of these systems and manufacturers as time will allow, and report all information, assuming that you the readers are interested. So let me know if this area would be of interest, and those of you that have knowledge, please send me what you have or know.

Mike Michaels of Canton, IL, supplied this list of firms supporting TRS80's (his letter was in #62). Anitek, PO Box 361136, Melbourne, FL 32936, (407)259-9397. They have memory expansions to 8 meg and speed up kits. Computer News 80, PO Box 680, Casper, WY 82602, (307)265-6483. A newsletter by Stan Slater and Ron Gatlin. Micro-Labs, 7309 Campbell Rd., Dallas, TX 75248 (214)702-8654, Hi-rez add on board by Ted Carter. Misosys, PO Box 239, Sterling, VA 20167 (703)450-4181. Hard drive and SCSI adaptors and software by Roy Soltoff. TRSTimes, 5721 Topanga Canyon Blvd. #4, Woodland Hills, CA 91367 (818)716-7154. A newsletter by Lance Wolstrup. (*This list has not been verified yet. BDK*)

On a personal note, My BBS contains a few of the Public Domain programs for the Kaypro systems. Open to the public, but, the first time you logon you will NOT have access to these files. Usually, by the following Saturday, your security will be upgraded for accessing all the Kaypro files.

Until next time, keep hackin'.
JW.

Write to:
TCJ Support Groups
Drawer 180
Volcano, CA 95689
BBS: (916) 427-9038
300/1200/2400 8N1

The Z-System Corner

By Jay Sage

Regular Feature

ZCPR Support

Failsafe Scripts

Techniques for Running Unattended Part 1: The Control Script

I have been promising for nearly a year to tell you about the techniques I developed before the summer of 1992 to allow my MS-DOS 486 computer to carry out simulations of electronic circuits completely unattended while I was away on travel for an entire month. In a series of columns starting with this one I will finally make good on that promise.

It is very important to my research projects that useful work be accomplished during times when I am away. This past summer, as I mentioned in my last column, I accomplished that aim by setting up an email account in Israel that permitted me to interact easily and frequently with my colleagues back at the laboratory. This year my research was in a circuit-testing phase, with a graduate student hard at work, so the email approach was quite suitable. Last year it was in an analytical phase that required a large number of time-consuming electronic circuit simulations to determine the operating margins for a new type of circuit I had invented.

There were two principal issues I had to address with the control programs I developed. First, in order to find the limiting values for a particular circuit parameter, such as the clock voltage, it was necessary not only to sweep the value for that parameter but also to determine automatically whether the circuit had operated successfully or had failed at the last value simulated. The parameter values could then be adjusted appropriately until the limiting values had been determined to the desired accuracy. A suite of PMATE editor macros handled this task. They will be the subject of a future column.

The second task was to make sure that a long sequence of computational tasks would be carried out even if there were power failures or other situations that required rebooting the computer. Certainly, I could not count on everything running perfectly smoothly for an entire month. The control programs that handled this problem will be the subject of this column.

One final introductory comment. What I am describing here operates on an MS-DOS computer using the 4DOS command processor replacement, which is very similar to the ZCPR3 command processor replacement under CP/M. Although I will not discuss in any detail the application of the same techniques under Z-System, I'm quite sure that they could be implemented if the need arose.

The AUTOEXEC.BAT File

When MS-DOS boots up, it runs a batch file with the name AUTOEXEC.BAT. This is where we have to start. Rather than add the special commands to AUTOEXEC.BAT itself, I simply added a line at the end of the standard AUTOEXEC.BAT file to invoke the power-failure-survival script, which I call FAILSAFE.BTM. Although a start-up command script is not a mandatory feature of Z-System, it is quite easy to make Z-System run a command or alias of the user's choice when the system boots.

Although we have just begun, it is already time for an aside. This column will necessarily serve in part as an introductory tutorial on 4DOS, because I make use of many of its special features. One of them is BTM files. These are the same as BAT files in terms of what they accomplish; the difference is that BTM (BaTch Memory) files run entirely from memory. When an MS-DOS BAT script is invoked, the first command is run. Then COMMAND.COM goes back to the file on disk and reads the second line, and then the third line, and so on. This procedure can be slow and can encounter problems if, for example, the BAT file is removed from the system, as it might be if it is run from a floppy.

With a BTM file the entire script is loaded into memory from the start. This allows it, with a slight penalty in memory usage, to run much faster. However, one does lose the ability, which is on rare occasions very useful, to modify the BAT file on the fly while the script is running. 4DOS supports both BAT and BTM files. In fact, 4DOS allows the mode, disk-based or memory-based, of either type of file to be changed on the fly by the LOADBTM command with the arguments ON and OFF. I almost always use BTM files because of the increased speed.

When I am not using the failsafe facility, which, of course, is almost all the time, I leave a very simple FAILSAFE.BTM file in the root directory. All it does is display a message reminding me of its name and presence and the need to modify it to perform any desired failsafe tasks. Listing 1 has an example. It also illustrates a very nice 4DOS command pair: TEXT and ENDTEXT. After a TEXT command, all lines in the script up to the ENDTEXT command are simply sent to the screen exactly as they appear in the file, including leading spaces. The same thing can be accomplished using ECHO commands, but that approach is clumsier and has problems with some characters (though 4DOS gets around that by allowing strings to be 'escaped' -- deprived of any special meaning -- by enclosing them with backward single-quote characters).

The Linking FAILSAFE.BTM Script

When I want the computer to perform some unattended tasks, I typically add some commands at the beginning of the dummy FAILSAFE.BTM command to change to the subdirectory where the work will be performed and then to invoke a full FAILSAFE.BTM script that resides there. Thus the FAILSAFE.BTM in the root directory serves as a link.

An example of the root FAILSAFE.BTM with these extra command lines is shown in Listing 2. Note the use of the 4DOS command CDD. This is like the DOS command CD except that it changes the drive as well as the subdirectory. A similar command, PUSHD, not only changes to the specified drive and directory but also pushes the name of the current directory onto a directory stack. The command POPD will take one back to the directory one was in before the last PUSHD was issued. The stack is fairly deep (255 characters), allowing one to backstep through several directory changes.

Note also the placement of an asterisk before the CDD command. 4DOS supports aliases just as Z-System does. One difference, however, is that 4DOS processes aliases before it processes real commands with the same name, and it is very common to define aliases to invoke built-in commands with certain options automatically included or with some other support commands. For example, I want to be able to enter "CD" and have CDD performed, so I have an alias named CD that invokes CDD. Sometimes, especially in scripts and even more especially in scripts that we distribute to others, we want to make sure that the real command runs, not an alias. Putting an asterisk in front of the command name does just that.

The Working FAILSAFE.BTM Script

Well, enough beating around the bush. Let's get down to the script that really does the work! Listing 3 shows a general-purpose FAILSAFE.BTM script that would be put in the subdirectory in which tasks are to be carried out.

Before we look at the command lines themselves, I want to say a few things about the general approach taken. The script is designed to read commands from a file called FAILSAFE.CMD. This makes it very easy to see what commands are going to be performed and to change those commands without modifying the FAILSAFE.BTM script. The environment variable FAILSAFE is used to store the number of the line in FAILSAFE.CMD to be processed next.

In order for the system to recover from a power failure or other event that requires rebooting, the entire environment (that is, the names and values of all environment variables) is saved in a file (FAILSAFE.ENV) that can be used to restore the environment after a reboot. The details of how this works will be covered shortly.

The Z-System, by the way, also supports the use of environment variables, though few people make use of this feature. Because the symbol names and values have to be stored in a disk file, rather than in memory as in DOS, access to them can be slow. Dreas Nielsen, a TCJ contributor in times past, was responsible for the most significant development of techniques and programs for using Z-System environment variables. The classic tools Rick Conn provided for working with them were SH, SHVAR, and SHDEFINE. Dreas contributed FOR, NEXT, PERFORM, and RESOLVE. The first three of those are generally distributed together in a package called FOR-NXT3.LBR, though there is a newer version of FOR distributed separately.

Now let's start working through the commands in the functional FAILSAFE.BTM. The first thing we do is make sure that there is a file FAILSAFE.CMD with commands to be performed. As we will see later, when all commands have been completed, this file is renamed to FAILCMDS.OLD to stop any further processing (otherwise, a reboot at this point would start the whole sequence over again).

Next we determine whether this is an initial invocation of the script or the resumption of operation after a reboot. We do this by looking for the file FAILSAFE.ENV. Any file of this name is deleted when all FAILSAFE operations have been completed, but the user should make sure that no such file is present, perhaps from a deliberately aborted run, when a new failsafe run is to be started.

This part of the script illustrates the powerful flow control processing that 4DOS has added to MS-DOS (of course, this is nothing new to Z-System people). The commands IFF, ELSE, ELSEIFF, and ENDIFF allow nested conditional processing of groups of commands. The indentation I use to make the script more readable has no effect on the actual processing.

In the present case, we test for the existence (or, more precisely, the absence) of FAILSAFE.ENV. If it does not exist, then this is the first time through the script; otherwise it is a reinvocation following a system reboot. For each cåse we perform a specific group of commands. Incidentally, I use the NOT option to reverse the existence test solely to suit my sense of logic. I want the commands for a first-time-though to come first in the script.

If this is the first time through the script, we initialize the environment variable FAILSAFE to the value 0 and write out the entire environment to the file FAILSAFE.ENV by redirecting the output of the SET command from the screen to the file. This is how we save the current state of the computation.

Note that we put an asterisk before the SET commands to make sure that the real 4DOS command runs and not an alias. I (and many other 4DOS users) have an alias called SET that paginates the display of the listing of environment variables by invoking the SET command with the /P option. We certainly don't want that to happen here!.

Besides those two essential actions, I like to keep a log of what has happened. When FAILSAFE first starts running, I write a line to a file called FAILSAFE.LOG that indicates that a new FAILSAFE run started at a particular time. This is done by redirecting the output of the ECHO command into a file in append mode using the ">>" redirection operator.

This command gives us a glimpse of a very powerful capability that 4DOS provides. There are system data variables, accessed like environment variables, that can tell almost anything about the computer system. Their names begin with the underscore character. In this command line we see %_DATE and %_TIME, which return the date and time stored in the system clock.

For those unfamiliar with MS-DOS, command-line tokens and variables are referenced using a leading '%'. Z-System uses '$' for most parameters, but shell variables, as in DOS, are referenced using '%'. There are dozens of other 4DOS system variables that allow one to find out such things as the kind of CPU chip and numerical coprocessor, the current drive and subdirectory, the position of the cursor on the screen and the color scheme there, whether or not Windows is running and which version, the number of the current DESQview window, and so on, and so on.

Now back to the script! If this is a resumption of operation, the environment variables saved in the file FAILSAFE.ENV are read back into the environment using the /R (read) option of the 4DOS SET command. Then a line is written to the log file indicating the time at which the script resumed operation after a power failure or other system reboot.

At this point we come to the main command processing loop. It functions as a WHILE loop by testing for the termination condition at the beginning. Specifically, we check to see if there are any more commands to perform in FAILSAFE.CMD. To do so we use a tremendously powerful feature of 4DOS. Besides the system variables we saw earlier, there is an even more powerful extension to the environment facility of DOS: system functions. These are like the system variables except that they take arguments.

System function names begin with the '@' character, and there are dozens of them. They test, parse, and otherwise manipulate strings and file names; compute with dates and times; determine how much disk space or memory of various types is available; reveal the sizes, time stamps, and attributes of files; read information off the screen; and so on and so on.

The first one we encounter here is @LINES. It takes a single argument, the name of a file, and returns the number of the last line in the file, where the first line has the number 0 (that's why we initialized the variable FAILSAFE to 0 rather than 1). The test "%failsafe GT %@lines[failsafe.cmd]" compares the next line number to be processed (%failsafe) to the number of the last line in the file FAILSAFE.CMD. As you can see, 4DOS can perform comparisons other than just equality and inequality, as in MS-DOS. The full set of relational operators is available (GT, GE, LT, LE, EQ, NE), for both string and numerical comparisons.

If we have run out of command lines, we delete the FAILSAFE.ENV file. We then want to rename FAILSAFE.CMD to FAILCMDS.OLD. Surprisingly to me, the 4DOS rename commands (REN or RENAME) do not have an option, as we have in Z-System, to delete any existing file with the new name. Consequently, we have to check explicitly for the existence of FAILCMDS.OLD and delete it, if necessary, before performing the rename operation.

The final step is to add an entry to the log file showing the time when the whole operation was completed. A blank line is also added by the "echo." command. With the job finished, we issue the 4DOS CANCEL command, which terminates all batch file processing, no matter how many levels deep we might be. (The QUIT command, which exits only from the current batch file, would probably have been adequate here, but CANCEL ensures that all batch processing will be terminated.) We then complete the IFF block with the ENDIFF command.

If the current command in the variable FAILSAFE does not exceed the number of lines in the FAILSAFE.CMD file, then the commands following the IFF block will be executed. We write a line to the log file showing the time at which the particular command line was started, and we then run the next command. How do we get the next command? We use the 4DOS system function @LINE (not to be confused with @LINES). It takes two arguments, the file name and the line number, and returns the specified line from the file.

After the command has been completed, we advance the command counter by using another 4DOS system function, @EVAL. This function takes one string argument, which is treated as an arithmetic expression, and returns the value of that expression. In this case the argument is "%failsafe+1". The term "%failsafe" gets replaced by the current value of the environment variable FAILSAFE. It is incremented by 1 by @EVAL, and the result is assigned by the SET command to the variable FAILSAFE, replacing the original value.

The modified environment is written out to the file FAILSAFE.ENV, replacing the older version, and the script then jumps back to the label LOOP. That's it.

At this point I would like to make one final observation about the precise way in which this script allows for recovery from a power failure or other mishap that requires rebooting. Processing does not begin at the exact point at which it was interrupted. That would be impossible to achieve purely in software. What happens is that processing starts over again at the beginning of the command line that was being executed when the problem occurred. It should be kept in mind, therefore, that this technique will not work if a partially completed command cannot simply be restarted from the beginning.

The commands I ran under FAILSAFE were all commands that could be rerun at any time, with any current output replacing any output from previous executions of the command. A situation to avoid is one in which required input files are modified or deleted. The commands should use input data that is left unchanged and write output data to new files, not to the input files.

Next time I will describe in more detail how I set up my command lists and how my commands were able to make intelligent decisions about what to do next.

Closing Announcements

I often start my columns with some announcements that are not related to the main topic. This time it seemed awkward to put them there, so I have stuck them on the end instead. There are two things I want to mention. First, in my column in issue 60 I referred to the newsletter that Frank Gaude at Echelon had distributed to his customers as "The Z Letter". Apparently no one, not even Bill Kibler, noticed the mistake except David McGlone. You see, David is the real publisher of "The Z Letter", which is still very much alive. Frank's newsletter was called "The Z-News". Second, I would like to provide some Z-Node update information. Lee Bradley, a stalwart of the Z community, has retired as sysop of Z-Node #12 in Hartford, Connecticut. I am happy to report, however, that Eric Palm has taken over its operation. The new phone number (still in the Hartford outdial of PC-Pursuit) is 203-826-5047. It is currently running on a Xerox 16/8 with a 10 meg hard disk, but Eric hopes to upgrade to a larger disk soon. *[Editor: Jay just notified me that Eric Palm returned Z-node 12 to Lee Bradley, so it is back at its old number.]*

I wish I could report the return to service of the Drexel Hill Z-Node in Philadelphia. Mike Finn is taking it over from Bob Dean, another stalwart of the Z community, but there continue to be relentless hardware problems. We all hope it really will be back up soon.

+-+

Listing 1. A dummy FAILSAFE.BTM file for use when the computer is operated normally. All it does is to display a message to the user.

```
@echo off
text
        This is a dummy FAILSAFE.BTM file in the root directory.
        To run commands in unattended mode, this file should be
        modified to invoke a real failsafe batch file.
endtext
```

+-+

Listing 2. A modified FAILSAFE.BTM file that would be placed in the root directory when unattended tasks are to be performed. The additional commands at the beginning change to the appropriate subdirectory, issue some initialization commands, and then invoke the main FAILSAFE.BTM script. In this example, we change to the subdirectory TESTCKT where my circuit will be simulated. We also add to the command search path the PSPICE directory where the circuit simulation program PSPICE is kept (ADDPATH is a utility I picked up somewhere, though a somewhat complex 4DOS command could perform the same function).

```
@echo off
*cdd c:\pspice\testckt
addpath c:\pspice
failsafe.btm
text
        This is a dummy FAILSAFE.BTM file in the root directory.
        To run commands in unattended mode, this file should be
        modified to invoke a real failsafe batch file.
endtext
```

+-+

Listing 3. The full FAILSAFE.BTM script that really does the work. See the text for a complete description of how it operates.

```
@echo off

REM     Quit if no commands to perform.
if not exist failsafe.cmd cancel

REM     Determine whether this is an initial start or a restart and perform the appropriate actions.
iff not exist failsafe.env then
        REM     If first time, initialize FAILSAFE
        *set failsafe=0
        *set >failsafe.env
        *echo %_date at %_time: Starting new FAILSAFE run.
                >>failsafe.log
else
        REM     On restart, restore the environment
        *set /r failsafe.env
        *echo %_date at %_time: restarting after failure
                >>failsafe.log
endiff
:loop

REM     See if commands remain to be performed. If not, clean up and quit.
iff %failsafe GT %@lines[failsafe.cmd] then
        *del failsafe.env
        if exist failcmds.old del /q failcmds.old
        *ren failsafe.cmd failcmds.old
        *echo %_date at %_time: FAILSAFE run completed.
                >>failsafe.log
        *echo. >>failsafe.log
        cancel
endiff

REM     Read the next command line from FAILSAFE.CMD and process it. Upon completion, advance
REM     the pointer in variable FAILSAFE.
*echo %_date at %_time: running line %failsafe >>failsafe.log
%@line[failsafe.cmd,%failsafe]
*set failsafe=%@eval[%failsafe+1]
*set >failsafe.env
*goto loop
```

Dr. S-100

By Herb R. Johnson

Regular Feature

Intermediate

Disk Drives and BIOS

Mail and Messages were all used up in my last column. Write more often!

Reader's comments in **TCJ's** letter column suggest that articles about computers assumc they already know how a computer works! Consequently, those articles contain words and concepts that are unfamiliar to a novice. More experienced computer users may have problems with "hardware" articles, and even programmers may not know the physics and mechanics of the devices they regularly program. Finally, some of us old-timers have forgotten some of this stuff.

This month, I give a tutorial on diskette drives that is good for anyone who owns a computer built after 1978, which should include most of the readership. I describe the physical nature of disks and disk drives, a discussion of the smarts in a diskette controller chip, and the operations of a disk drive at the BIOS (basic input/output system) software level. I'll leave advanced operations, such as disk directories and boot-up stuff, for another column or another columnist, depending on reader response to this column. But I will include some BIOS fragments in 8080 code that closely follows the methods in this article.

For my S-100 readers I have available a BIOS for SSSD 8" and a number of SD Systems diskette controllers (suitable for 8" and 5", single and double density). Contact me for details.

Tutorial Topic: What is a disk drive?

When I considered writing about diskette drive controllers, I decided to start at the basics. The following is a pretty complete description of the technology behind a disk drive. When you know the fundamentals, diskette drive operations make more sense. From this, BIOS (input/output) software will also make more sense. Before you continue to read this article, I recommend you locate a loose disk drive and a 5" or 8" floppy diskette and keep them on hand as you read.

History

The first S-100 systems had no disk drives, because floppy disks were a recent invention of IBM and were expensive. Gary Kildall had yet to develop his CP/M system and to start Digital Research to market it. Programs were stored in and read from **paper tape**, usually via Teletype ASR-33 printing terminals. Impatience and the inconvenience of a write-once (non-erasable) medium, and the availability of audio cassettes led to a variety of **tape cassette** formats supported by Tarbell, IMSAI, Processor Technology (Sol) and others. The microcomputer market was driven by surplus equipment; "hand-me-downs" from mainframes and minicomputers. Gary Kildall's CP/M was tested on a worn-out Shugart SA-800 8-inch drive, and for years the 8-inch diskette was the only cheap and fast mass media for personal computers. Today's 5.25 inch and 3.5 inch drives are little different from the original 8-inch drive, are very similar in general operation to hard disk drives, and similar in many principles to cartridge tape and even CD-ROM drives!

The Basics of Diskettes

A disk drive is a simple device, when you consider it as a collection of components each performing a function. It is designed to access locations on a diskette both by rotating the diskette and by moving an arm over the disk along one radius of the circular diskette. It must be in constant motion because it is the relative motion of the magnetized disk past a magnetic sensor on the arm that permits the head to "read" the magnetic information on the diskette. A moving magnetic field (like the magnetic fields on the diskette) creates an electric current in a fixed wire (like the magnetic head on the arm of the disk drive).

Grab a 5.25" diskette from your computer and examine it (figure 1). Diskettes are plastic disks with magnetic coatings, encased in an envelope with holes. The long hole is where the disk drive magnetic head moves around to access areas of the diskette surface. The small circular hole is where the index sensor optically looks for a smaller hole in the magnetic disk, which establishes a constant starting point for each rotation. The large circular hole in the center is where the diskette media "cookie" is grabbed through the jacket by the drive in order to rotate the media. (Cookie is the manufacturing term for the media, as it is punched out from a long magnetic tape in "cookie-cutter" fashion.)

Information on the diskette is stored as a series of records of fixed length, called **sectors** (figure 2). Sectors are lined up around one circumference (circle of constant radius) of the diskette on a **track**. These tracks are arranged around the center of the disk, like circular ripples in a pond, and are counted from track 0 from the track closest to the edge of the diskette. Sectors are counted from sector 1, the first following the index hole, and are **physically** numbered in sequence

(1,2,3,...the significance of this will be described later.)

The last consideration for diskette information storage is **density**. Data is stored as magnetic bits on a track using one of two methods. "Single density" stores a bit of data as a single pulse, which involves two changes or reversals of magnetic field. "Double density" stores more data by representing a data bit as only one change of magnetic field.

We can now intelligently describe the "classic" standard 8" format for tracks and sectors, derived from IBM's 3270 diskette 8" standard, was single-sided (i.e. one magnetic head on the drive), single density, 128 bytes per sector, 26 sectors per track, 77 tracks per disk. This is classically known as a "quarter-meg" or 256K byte format.

The disk drive

A disk drive (figure 3) has a motorized **spindle** to grab the diskette and to rotate it at constant (radial) speed. To access the various tracks, an **arm** moves across the diskette. The arm is driven by a "**stepper" motor**, which moves the head end of the arm to the same series of places along the radius of the diskette. These places, combined with the constant motion of the diskette, define the location of the tracks on the diskette. The **index sensor** on the drive identifies the beginning of the track, and timing establishes the general location of each sector and of the data in the sector.

The diskette's magnetic information is read by the **read/write head** at the end of the arm that makes contact with the diskette. Electronics will amplify signals read by the head and produce a stream of binary data from the drive; similar electronics take the binary data written to the drive and send them to the head, which creates magnetic fields that are recorded onto the diskette.

Additional electronics manage the motors and sensors, and communicate with the computer's disk controller as suggested by figure 4. The **stepper motor logic** is commanded to move the head by steps, forward or back. A **track 0 sensor** at the outer most head position signals the arrival of the head at track 0. Most drives hold the read/write head away from the diskette. A "head load" command operates a **head load relay** to drop the head down onto the magnetic surface. A **write protect sensor** reads a defined edge of the diskette jacket to see if the diskette has a "write protect" notch. (Curiously, on 8" disks the notch must be closed to write; on 5" disks the notch must be open to write!). **Drive select logic** examines the four drive select (address) signals from the computer, and compares them to the drive's **address jumpers**. If the comparison matches, and the diskette is inserted and up to speed, it enables the rest of the drive's logic including the **drive ready** signal. The read/write electronics are driven by the read and write signals from the controller chip, usually through some additional hardware, to adjust the timing of the data. The write gate signal from the controller switches the drives read/write electronics from read to write for the duration of the disk write.

The controller chip

The earliest diskette controllers were dedicated microprocessors with specialized programs, or a handful of digital logic chips. For most computer systems (except Macintosh!), the floppy controller primarily consists of a **floppy disk controller chip**. These chips provide a generic microprocessor interface that, to the microcomputer, make the floppy disk drive appear to be controlled by a number of registers (dedicated data locations) in address space. They also provide a control interface to the diskette drive which is consistent with its standard control signals; and read/write electronics that read or write data. Finally, the controller runs its own "program" to support the track and sector format of the data on the floppy disk, details that are normally invisible even to the programmer!

What does the programmer normally see? The disk controller registers, which are:
the Status (read) and Command (write) register;
the Track register;
the Sector register;
the Data register;
the Drive Select "register" (not a part of the controller chip).

The track and sector registers represent the value of the track and sector to be read from or written to. The command register receives the command from the microprocessor to read, write, seek a track, or to format a track. The status register tells the microprocessor the results of a command. The data register is where data contained in the sectors is read from or written to. The Drive Select register is usually implemented as additional electronics in the controller. This register controls the Drive Select lines and often controls other signals including drive type (8" or 5") and density (single or double).

Diskette operations, particularly reads and writes, are among the fastest I/O operations of your computer. Yet they are dependent on placing a particular portion of the diskette under the read/write head, an operation that may take a relatively long period of time to occur. So, the disk controller chip must have the ability to make the processor act when the diskette is ready. One way to force an action is by **interrupt.** The BIOS software performs operations to prepare for a disk read or write, executes the read or write command, and then the software enters and remains in a "do nothing" bit of code. When the disk controller chip sees that the diskette is ready, it sends a **hardware interrupt** signal which forces the processor to jump to an **interrupt handler** piece of code to perform the actual reading or writing of each byte. Interrupts end when the reading or writing is completed.

Another way to control the processor is by forcing it to **wait**. The BIOS software performs all its operations to prepare for a read or a write. Then the read or write command to the disk controller chip to send a **hardware wait signal** to the processor. The microprocessor then performs no operations at all until the disk is ready for a byte of data. Only then is the processor released from its **wait state** and it continues to execute the BIOS code to read or write the byte of data.

Wait states end when the reading or writing is completed.

In either case, the controller chip manages these waits or interrupts. If the disk is "never" ready, the controller chip is programmed to **time out** after a set period of time and to set error flags (bits) in the status register. The status register must be read at the end of the read or write routine to verify the successful completion of the read or write or to determine the type of error.

Diskette formats and the controller chip

What is going on when you **format** a diskette? More to the point, what information is written to the diskette at the track and sector level? The good news is that, for the most part, the floppy disk controller chip handles most of this; the "bad" news is that you have to know about this if you write a format program! If you are confused about something called "logical" sectors versus "physical" sectors, the following offers an explanation.

A diskette track contains more than a collection of sectors with (your) data. Before and after each sector is a **header** and a **trailer**, respectively. The header has a fixed format of several bytes, except for a byte that is the **physical sector number**. The trailer has a similar format, except for a **checksum byte**. A checksum is a consistent way of adding up the data in the sector to create a unique byte value: on read, this provides a confirmation to the controller chip of the data if it adds up to a value matching the checksum. On write, the chip generates a checksum after writing the sector data. The rest of the header and trailer provides time and diskette space for the controller chip to prepare or complete the read or write of the sector.

You'll remember that sectors are stored on the track one after another. You might presume that sectors can simply be read off, one after another as well. However, if you have a slow microprocessor, by the time the first sector is read **and processed**, the second sector may have gone by and the processor would have to wait until the next diskette revolution to read it. A more efficient solution is to "stagger" the sectors such that the next "logical" sector is actually some number of "physical" sectors away. For our standard 8" diskette, the distance is 6 sectors and so sector numbers are counted off by sixes (skipping 5) around the track. The first physical sector is also logical sector 1; the 6th physical sector is logical sector 2, the 12th is 3, and so on as illustrated. Look in the BIOS code for software details.

When a sector read or write is requested, it is the controller chip that reads these physical sector numbers, waiting for the "right" sector (by number) to appear before starting its data read or write. In a similar fashion, the track has a header and trailer, with the header containing a track number which is read and verified by the controller chip. (By the way, there is no "logical vs. physical" track complications on floppies; however there are on hard drives, allowing spare tracks to logically replace bad physical tracks!).

The drive in action

In the following action descriptions, I'll refer to the **disk drive,** the **(floppy diskette) controller chip,** the rest of the **controller**, and the **microprocessor** which is running the **BIOS** software. I'll also refer to the disk drive interface signals and registers by name. Use the previous illustrations to guide your reading, and try to follow along.

A drive is selected by address (and possibly by size and density), and appropriate data is sent to the Drive Select register. The drive address is sent to all drives, and the Drive Ready line of the addressed drive must become active if the drive is available. Before the first read from or write to a diskette, the drive should be "homed" so that the drive's track number and the track number register are identical. A home command or "seek to track 0" is sent to the diskette controller chip. The chip then sets the disk drive's Direction line to "back" and pulses its Step line (which moves the head one step at a time) until the Track 00 signal from the drive is active. If the Track 00 signal "never" appears, the controller chip eventually "times out" and sets error bits in the Status register indicating the Home or Seek operation failed.

To read a sector, the BIOS typically expects this order of operations:
Select drive;
Select track;
Select sector;
Read a sector.

"Select drive" may require a drive change. As the controller chip knows about only one drive at a time, information about the current drive must be stored, and information about the selected drive must be restored. This information certainly includes the track number; it may also include the type and density of the drive, and also may support one of a number of diskette formats.

"Select track" may require a step to a new track, if the current track is different. The controller chip is sent the new track number, and then a "seek track" command. The controller is then "busy", setting the Direction line and sending Step pulses until the track is reached or until a timeout occurs. The Status register is then read by the microprocessor to determine the result of the operation.

"Select sector" requires a sector value to be sent to the controller chip's Sector register. However, many BIOS's have a **sector translation** table. Why? Well, our earlier discussion suggested that logical sectors may be numbered differently from the physical sequence of sectors on the diskette, to allow time between sectors before the next "logical" numbered sector. While this can be done on the physical diskette, it can also be done by a conversion table in software. For a 26-sector diskette with a physical offset of 1 (i.e. all sectors in physical order) you can create a table to support a logical offset of, say, 6, as follows:
1,7,13,19,25,5,11,17,23,3,...
and you simply add the logical sector number to the table's address and read

the byte at that location to get the translated physical sector number.

Finally, after the correct disk drive is on the correct track (reading the track headers) and the sector number is loaded into the controller chip, a read or write command is sent to the controller chip. The chip verifies the track number, waits for the correct sector (in the sector headers) to come along, and then tells the rest of the controller (or microprocessor) to either receive (read) or provide (write) data until the sector is completed. The controller compares its accumulated checksum value with the read data's checksum in the trailer (or creates one and writes it out). The status register is set by the chip, and finally the microprocessor is told the operation is complete and the BIOS can read the status register to verify the operation.

If an error occurred, most BIOS's will retry: the read or write will be reissued a number of times. If a "wrong track" error occurred, the BIOS should try another "seek" operation; if that fails, a "home" operation is often tried as well. All error conditions are usually reported to the user if they persist; most errors are occasional and the user does not see them. Some utility programs can use these errors to "mark" a bad sector.

References

Any hardware manual on floppy disk controller chips would be very helpful. Any CP/M BIOS configuration book would be helpful, too. A hardware manual on disk drives would be useful. Check your local library, public or college; or ask a electronics tech or engineer that fools around with disk drives for a look through his or her library.

Bookstores are generally only good for **software** books these days. Find a large bookstore in your area and browse: my recommendation of a particular book won't help you if you can't find it. Look for "hardware secrets" or chapters (not sections) on "disk drives".

Local libraries are usually good for older books on computers, which tend to have more of these technical details than current books. Check the **electronics** section and the **computer** section: they may be separate. Also, look for 1970's issues of **Byte, Dr. Dobbs Journal,** or other computer or electronic magazines. If available, these are good sources.

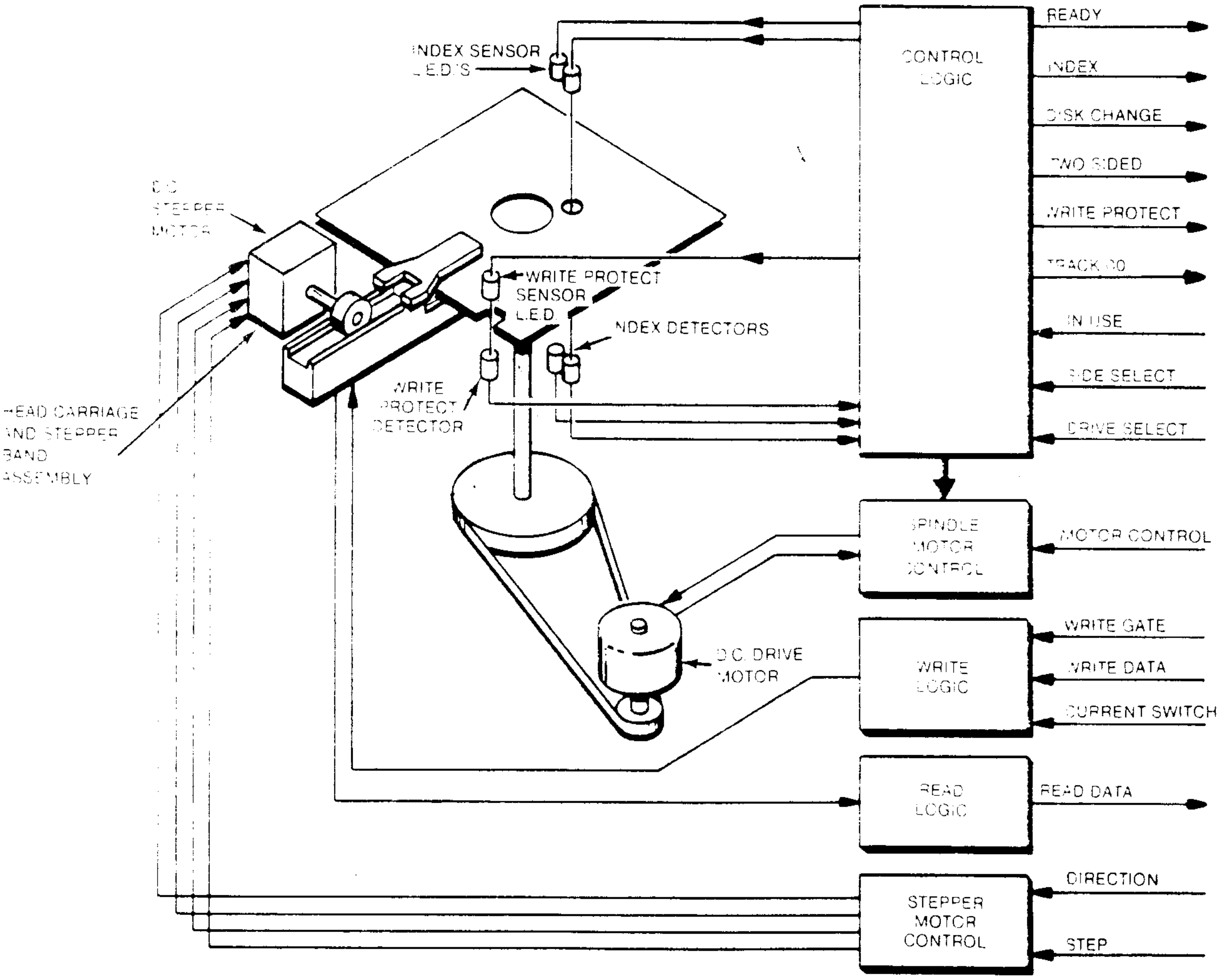
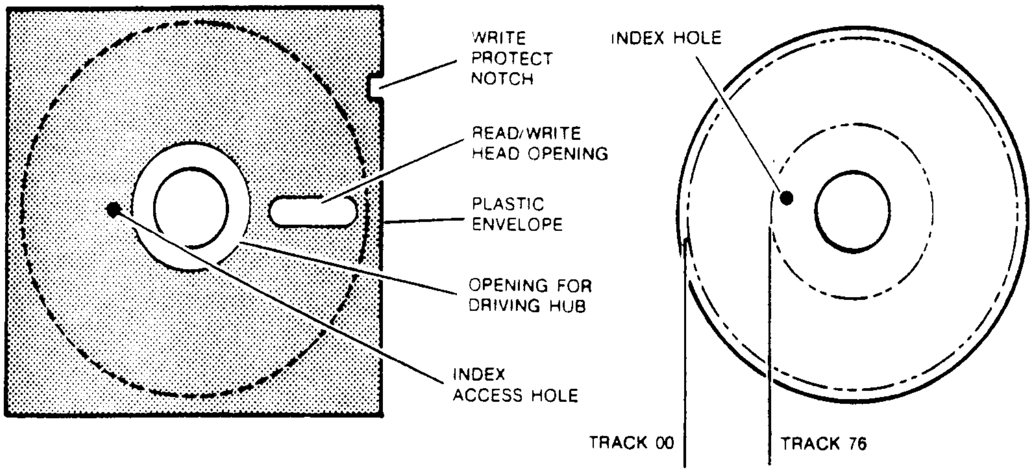
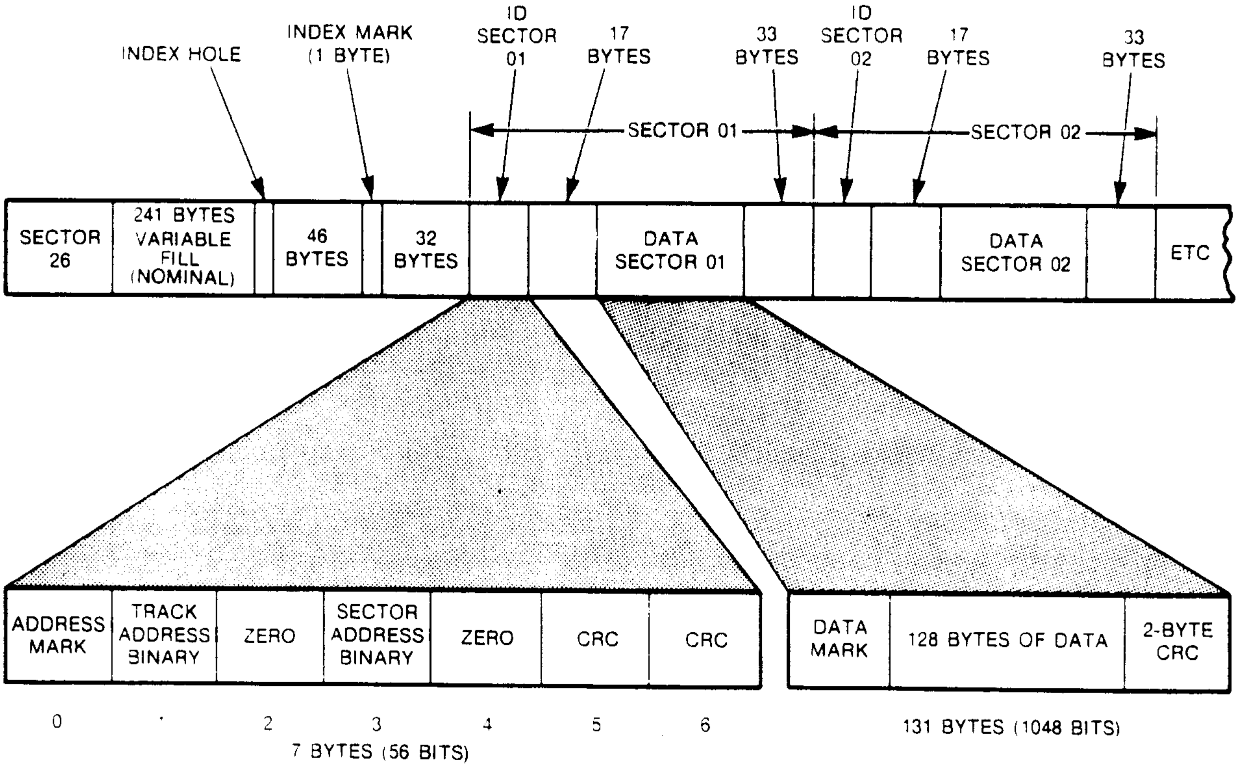


Figure 3 and 4



DISC CARTRIDGE AND DISC CONFIGURATION

Figure 1



IBM TRACK FORMAT (SEE IBM OEMI MANUAL GA21-9190-2)

Figure 2

CODE FRAGMENT ON NEXT PAGE.

Supporting code fragments

```
; code fragments as derived from a Tarbell SSSD BIOS and from SD SYSTEMS BIOS,
, to illustrate diskette controller operations. This code is close to working code, but may
; not be complete. The comments are functionally correct.
; For reference to people not familiar with 8080 processors:
;          8080 registers are: A, B&C, H&L, D&E. M is equivalent to (HL).
;          MOV A,B    moves a value from the B register into A, the accumulator.
;          STA HELLO           moves the A register into location HELLO
,          LXI H,0             moves 0000 (word) into the HL register pair
;          MVI H,0    moves 00 (byte) into the H register
;          LXI B,0    moves 0000 into the BC register pair
;          DAD D               adds the contents of DE to HL
; This controller chip uses WAIT STATES to delay the processor. A read from
; or a write to the DATA register produces waits states unless NOWAIT is previously
, called. YESWAIT enables wait states. The controller chips registers are:
TRK      EQU      ??        ;track
SECTOR   EQU      ??        ;sector
STATUS   EQU      ??        ;status
CMD      EQU      STATUS    ;command register, same address as STATUS
DATA     EQU      ??        ;data in/out register
NBYTES   EQU      128                ;one sector's worth
;        FIXED DATA TABLES FOR TWO DRIVE SYSTEM
DPBASE:  DW       TRANS,0000H        ,FOR DISK 0
         DW       0000,0000
         DW       DIRBF,DPBLK
         DW       CHK00,ALL00
         DW       TRANS,0000         ;FOR DISK 1
         DW       0000,0000
         DW       DIRBF,DPBLK
         DW       CHK01,ALL01
; the logical to physical translation table
TRANS:   DB 1,7,13,19,25,5,11,17     ;SECTOR TRANSLATE
         DB 23,3,9,15,21,2,8,14      ;sector offset of 6
         DB 20,26,6,12,18,24,4
         DB 10,16,22

;
; MOVE DISK TO TRACK ZERO.
HOME:    XRA  A               ;TRACK ZERO
         STA  TRK             ;UPDATE OLD WITH NEW.
         mvi  a,03H           ;home command
         out  cmd             ;send command to controller chip
         call busyx           ;wait for controller chip status not busy
         xra  a               ;and do a seek to zero to verify
         CALL SEEK
         RET                  ;RETURN.
; SELECT DISK DRIVE NUMBER ACCORDING TO REGISTER C.
; set up seek now, do physical select later at read or write time.
SELDSK:  LXI H,0              ;if error, return HL=0
         MOV A,C              ;GET NEW DISK NUMBER.
         CPI 2                ;is it > or = 2?
         RNC                  ;ONLY HAVE TWO DRIVES
         STA DISKNO           ,SAVE DISK NUMBER
         MOV L,A              ;DISK NUMBER in L, 0 in H
         DAD H                ;HL*2
         DAD H                ;*4
         DAD H                ;*8
         DAD H                ;*16 (16 is length of block)
         LXI D,DPBASE         ; get table address
         DAD D                ;add DE
         RET                  ;address of correct base table returned in HL

;
; SET TRACK NUMBER TO VALUE IN REGISTER C.
, ALSO PERFORM MOVE TO THE CORRECT TRACK (SEEK).
SETTRK:  MOV  A,C             ,GET NEW TRACK NUMBER.
         STA  TRK             ,UPDATE OLD WITH NEW.
         CALL SEEK            ;MOVE TO NEW TRACK.
         RET                  ;RETURN

;
; SET DISK SECTOR NUMBER, do sector locate at read/write time.
SETSEC:  MOV  A,C             ;GET SECTOR NUMBER.
         STA  SECT            ;PUT AT SECT # ADDRESS.
         RET                  ;RETURN FROM SETSEC.

;
;SECTOR TRANSLATE DE has the address of the translation table. (this is the first
; word of the DPBASE table, TRANS.) BC has the sector number to be translated
,  return HL with the new sector number.
SECTRAN: XCHG       ;swap HL and DE
         DAD B      ; HL=HL+BC, point to new sector number
         MOV L,M    ; read the new number from TRANS into L
         MVI H,0    , clear H
         RET

; MOVE THE HEAD TO THE TRACK IN REGISTER A.
; as a write to the data register is involved, wait on read/write must be disabled temporarily
SEEK:    PUSH B               ,save registers on stack
         PUSH PSW             ;processor status word
         PUSH H
         CALL SELDSK1         ,do physical drive select
         POP H                ;restore registers
         POP PSW
         MOV  B,A             ;SAVE DESTINATION TRACK.
         MVI  A,RTCNT         ;GET RETRY COUNT.
SRETRY:  STA  ERCNT           ;STORE IN ERROR COUNTER.
         IN   TRACK           ;READ PRESENT TRACK NO.
         CMP  B               ;SAME AS NEW TRACK NO.?
         JNZ  NOTHR           ;JUMP IF NOT THERE.
THERE:   POP  B               ;restore B&C.
         RET                  ;RETURN FROM SEEK.
NOTHR:   call nowait          ;disable wait on controller
         call busyx           ;wait for not busy on controller
         MOV  A,B             ;RESTORE track FROM B.
         OUT  DATA            ,TRACK TO DATA REGISTER.
         MVI  A,16H           ;SEEK command, SET FOR 10 MS STEP
;VERIFY ON LAST TRACK.
         OUT  CMD             ;ISSUE SEEK COMMAND.
         call busyx           ;test not busy on controller
         call yeswait         ; enable wait
         IN   STATUS          ;READ STATUS.
         ANI  91H             ;LOOK AT BITS.
         JZ   THERE           ;OK IF ZERO.
         LDA  ERCNT           ;GET ERROR COUNT.
         DCR  A               ;DECREMENT COUNT
         JNZ  SRETRY          ;RETRY SEEK.
bx2:     POP  B               ;RESTORE B&C.
         LXI  H,SKMSG         ;PRINT "SEEK ".
         IN   STATUS          ;READ DISK STATUS.
         ANI  91H             ;LOOK AT ERROR BITS.
         MOV  D,A             ;PUT IN REG D.
         JMP  ERMSG           ;DO COMMON ERR MESSAGES.
; READ THE SECTOR AT SECT, FROM THE PRESENT TRACK.
; READ into memory at ADDRESS DMAADD.
READ:    CALL SELDSK1         ;do physical drive select
read2:   MVI A,RTCNT          ;GET RETRY COUNT
RRETRY:  STA  ERCNT           ;STORE for countdown.
         LHLD DMAADD          ;GET STARTING ADR in HL.
         MVI  A,0D0H          ;send controller interrupt command
         OUT  CMD             ; to get data in status register
         XTHL                 ;SOME DELAY required by chip
         XTHL
         IN   STATUS          ;READ chip STATUS,
         ANI  20H             ;LOOK AT head load BIT.
         LDA  SECT            ;GET SECTOR NUMBER.
READ1:   OUT  SECTOR          ;write sector number to chip.
         MVI  A,RDHDL         ;READ command WITH HEAD LOAD
         JZ   READE           ;if zero, HEAD NOT LOADED.
readf:   MVI  A,RDCMD         ;else use command W/O HD LD.
READE:   OUT  CMD             ;SEND COMMAND TO controller chip.
         push b               ;save B
         MVI  B,0080H         ; counter to read 128 bytes (one sector)
         DI                   ;disable interrupts during read
RLOOP:   IN   DATA            ;READ A DATA BYTE FROM DISK, force wait.
;controller forces processor to wait until byte is available.
         MOV  M,A             ;PUT BYTE INTO MEMORY.
         INX  H               ;INCREMENT MEMORY POINTER.
         DCR  B               ; decrement counter
         JNZ  RLOOP           ;KEEP READING.
         pop  b               ;restore B
         EI                   ;reenable interrupts
RDDONE:  IN   STATUS          ;READ DISK STATUS.
         ANI  9CH             ;LOOK AT ERROR BITS except busy.
         RZ                   ;RETURN IF NO ERROR.
CHECK:   CALL ERCHK           ;CHECK FOR SEEK ERROR
         LDA  ERCNT           ;GET ERROR COUNT.
         DCR  A               ;DECREMENT COUNT.
         JNZ  RRETRY          ;TRY TO READ AGAIN
         LXI  H,RDMSG         ;PRINT "READ ".
         JMP  ERMSG
;PHYSICALLY SELECT DISK FOR READ,WRITE,HOME,OR SEEK
SELDSK1: LDA DISKNO           ;GET requested DISK number
         MOV C,A
         LXI H,DISKNOC        ;compare to DISK CURRENTLY SELECTED
         CMP M
         RZ                   ;if same, already there so return
         MOV  A,M             ;else GET OLD DISK NUMBER.
         PUSH D               ; save DE registers
         MOV  E,A             ;PUT OLD DISK NO. IN D&E.
         MVI  D,0
         LXI  H,TRTAB         ;GET ADDRESS OF TRACK TABLE.
         DAD  D               ;ADD DISK NO. TO ADDRESS to get old table.
         IN   TRACK           ;READ 1771 TRACK REGISTER.
         MOV  M,A             ;PUT result INTO old TABLE.
         MOV  A,C             ;GET NEW DISK NUMBER.
         MOV  E,A             ;PUT NEW DISK NO. IN D&E.
         LXI  H,TRTAB         ;to GET ADDRESS OF new TRACK TABLE.
         DAD  D               ;ADD DISK NO. TO ADDRESS.
         MOV  A,M             ;GET NEW TRACK NUMBER.
         OUT  TRACK           ;PUT INTO 1771 TRACK REG.
         MOV  A,C             ;UPDATE OLD DISK NUMBER.
         STA  DISKNOC
         MVI  C,DRIVE0
         ORA  A               ;drive 0?
         JZ   DSK1            ;if not, drive 1
         MVI  C,DRIVE1
DSK1.
         IN   SELECT          ,get values
         ANI  0F0H  ;mask off drive #
         ORI  C               ;new drive
         OUT  SELECT          ;SET THE LATCH to select the drive.
         XRA  A               ;SET A = 0.
         POP  D               ; restore DE
         RET                  ;RETURN FROM SELDSK.
,                             END CODE FRAGMENT
```

Real Computing

By Rick Rodman

32-Bit Systems

All Readers

Languages

Platform-independent languages: Calling conventions

Rather than cover the topics I promised last time, I have a burning desire to present technical nuts-and-bolts issues that could affect your choice of a platform-independent language.

The first issue is *calling convention.* You'll recall that the first-generation language was plain assembly with branches, and the second-generation language was assembly with subroutines. When you call a subroutine, there are three basic issues: how do you pass control to the subroutine and get it back, how to you pass parameters to the subroutine, and how do you get parameters back. A combination of choices, one from each group, is what we call a "calling convention." Normally the control flow is determined by the design of the processor hardware, but the latter two cases offer virtually limitless possibilities.

The calling convention is really independent of the language. Any language could use any calling convention. In fact, some compilers or interpreters have options, keywords or "pragmas" to allow other calling conventions to be used for particular calls. Nevertheless, in each language there is a "standard" calling convention.

In Forth, there are assumed to be two stacks, a control stack and a data stack. Calls and returns are placed on the control stack in the manner determined by the hardware. Data being passed is pushed onto the data stack in the order passed; the subroutine removes the data it needs from the stack, a process referred to as "unjunking the stack". Data returned by the subroutine is pushed onto the data stack in the order of return.

In Pascal, the control stack and the data stack may be the same. When calling a subroutine (function or procedure), data is pushed onto the stack in the order passed; the called subroutine does the "unjunking". Data being returned, if any, is returned in a CPU register. Since the register can only contain a single value, or "scalar", only one value can be returned.

Why this limitation? For two reasons. First, so the programmer can have the 'luxury' of carelessly ignoring the return value without any code being generated; second, because Pascal was originally a simple teaching language designed by Wirth and not intended for any practical use.

In C, the standard calling convention is even stranger. There is assumed to be only one stack. Parameters being passed to a function are pushed *in reverse order of call*, last parameter first, and the *caller* is required to unjunk the stack. Data being returned is placed in a register, as in Pascal.

The reason for the reverse order and caller unjunking is to allow a variable number of parameters to be passed. You see, if the parameters were pushed in the order passed, we couldn't easily find the first parameter passed without knowing how many parameters were passed. Also, the subroutine can't be expected to unjunk the stack if he doesn't know how many parameters the caller pushed.

Nearly every implementation of the languages I've mentioned use the calling conventions described. In other languages, such as Fortran and PL/I, implementers seem to have been allowed more freedom to choose a scheme they like. But what I'd like to point out is the fact that, even though the calling convention really is not determined by the syntax of the language per se, it has a "princess and the pea" influence on the implementation of the language: unseen, but painfully felt.

Historically, C's variable-arguments feature has been far more costly than anyone could have anticipated. But now that it's been "engraved in stone" by the ANSI and ISO committees, C, as a language, is stuck with it. Ironically, after the thousands, perhaps millions, of hours spent figuring out a standardized way it could be made to work, the syntax they developed is so cumbersome that almost nobody goes to the trouble.

Forth enthusiasts can point out that their language has the most powerful and convenient calling convention. Interestingly, Digital Research's PL/I, offered for the 8080 and the 8086 but now, unfortunately, off the market, used a stack-in, stack-out calling convention. Even strings were passed on the stack.

DR's PL/I-80 also beat every other CP/M compiler on the market in another area: *code quality.* But what do we mean by that?

Code Quality

In general, by "code quality" we mean, in a comparative sense, that code generated for a given statement is smaller and/or faster. The easiest way to explain this is with an example.

Consider the C statement: "i = qqq[x]", where all of the variables are stack-frame variables (that is, local variables implemented in stack memory, with the stack pointer moved down to make room). The code for the NS32 generated by Small-C looks like this:

```
ADDR 0(SP),R1        R1 points to i
ADDR 4(SP),R2        R2 points to qqq
MOVD 18(SP),R0        Get x in R0
ADDD R2,R0           Add R0 to R2,
                     indexing into qqq
MOVXBD 0(R0),0(R1)  Move the
                     qqq value to i.
```

But Phil Prendeville's C compiler generates much better code:

```
addr    -10(fp),r0   Point to qqq with R0
addd     8(fp),r0     Add x to R0
movxbd  0(r0),-16(fp) Move the qqq[x]
                      value to i.
```

The GCC compiler generates very similar code. Note that the Small-C compiler generates code for very small operations. Really, the problem is that Small-C internally operates on an 8080 CPU model. Let's see the 8080 code Small-C generates for the same statement:

```
lxi     h,0     Offset of i.
dad     sp      Add to stack pointer.
push    h       Push HL (points to i).
lxi     h,4     Offset of qqq.
dad     sp      Add to stack pointer.
push    h       Push HL (points to qqq.)
lxi     h,18    Offset of x.
dad     sp      Add to stack pointer.
call    ?gint   Load HL from
                 memory pointed to by HL.
pop     d       Pop address of qqq
into DE.
dad     d       Index into qqq with HL.
call    ?gchar  Get the byte pointed to
                by HL into HL.
pop     d       Restore address of i into DE.
call    ?pint   Store HL at memory pointed
                to by DE.
```

Would an assembly programmer have written anything like that? I doubt it. Of course, the 8080 doesn't have very good instructions for performing stack-frame variable work. The only reason for using stack-frame variables is to allow recursion; better performance can be obtained from the 8080 by using static variables for most temporary variables. By the way, the Z280 (aka Z800) has some new indexing instructions which would allow better code to be generated.

Looking at the 8080 code, one can understand why many assembly coders would view the use of any compiler with revulsion. This is why I presented the NS32 examples. Let's face it - the Z80, with all those registers and bit-manipulation instructions, was *designed* for assembly language. Most newer processors have added instructions which allow compilers to generate better code. Still, no machine will ever generate better code than a smart human assembly coder.

When discussing code quality, Forth requires special criteria. While the threaded code is extremely small, it is slow because every single instruction word involves at least two jumps and, in some cases, a call and a return. In most cases, the processor bandwidth loss is over fifty percent. Some Forth packages allow "inline" coding in which the interpreter overhead is reduced.

Should we be overly concerned about code quality? Only when it makes a difference - when memory is tight, or when speed is critical. In a dedicated processor that checks the temperature in a hot water heater once every five minutes, these considerations wouldn't apply. Small-C's syntax limitations are usually more of a problem than its code quality. It does an acceptable job, but if you have a better compiler, use it.

These, then, are the technical issues to keep in mind when comparing two different languages. Unlike the syntax issues, in which the "prettiest", most readable language always wins, the technical issues can be hard to see - but they are usually quite easy to quantify.

A novel approach to platform independence which didn't catch on was UAL, "Universal Assembly Language", in which assembly macros are used for all operations. This approach was used successfully in the original implementation of the SNOBOL interpreter. The problem of every assembler having different macro syntaxes, or no macro support at all, could be overcome by writing a separate macro processor using, say, the macro syntax of the M80 assembler (in my opinion, the best assembler ever written for any processor).

Modularity

For all the breath that's been wasted in shouting about Object Orientation, the real thrust of OO was always modularity: reusable packages of code. Funny, but I thought that's what Structured Programming was supposed to do for us. The real problem is not expressible in a syntax for a programming language, anyway - because the real problem is that the *environment in which the programs run does not support modularity.*

Take, for example, the PC environment - a worst case. We have four types of object code modules. First, there are device drivers, ".SYS" files. Next, there are "absolute" programs, or ".COM" files. Third, there are "executable" or ".EXE" files. Lastly, there are modular object files, or ".OBJ" files. Only the last of these corresponds on a one-to-one basis with a software module - and it is the only one which is *not* directly executable! Only a hard-bound "clump" of modules can be run.

How stupid, you might say. But this is true of *all* the common programming platforms that exist! True, AmigaDOS, Windows, OS/2, and some types of Unix do support dynamic linking, which does allow modules to be separately compiled and dynamically linked. But in each case there are substantial problems which prevent truly modular design from being done. Also, there's Forth. Forth's Achilles Heel is its linear dictionary, which only allows unloading ("forgetting") from one end; words in the middle can't be replaced.

In a truly modular programming environment, modules could be loaded, unloaded or replaced at will or as needed. There would be only one form of object code, whether for devices, program modules, or whatever. Nothing would be "hard-bound", even within the operating system - everything would be dynamically loaded and linked. Programmers would find such an environment delightful.

You wouldn't always need the complete environment. In a microwave oven controlled by an 8051, for example, the dynamic links used during debugging could be replaced by hard links in ROM. If the modules were developed in a platform-independent language, we could develop the code on a nice desktop machine, then recompile and burn into PROM for testing.

The outside world takes for granted we've had such an environment for years, of course. That's one reason that, when what we call "normal users" start using computers, the predominant reaction is disillusionment, followed by frustration, followed by resignation.

In truth, Object-Oriented Programming is just new buzzwords for old concepts. ("The PIP method copies one instance of a file object to another.") The old concepts are still valid, of course. It's sad how the computer world dashes madly from one fad to another without ever really completing any of its grand dreams.

More on the Internet

Gary Welles followed the instructions in #62 and got *this close* to actually getting a file from an FTP server. What makes it interesting is that he's accessing the Internet from MCI Mail through what we call a "gateway". Gateways in the E-mail world are much like customs officers in the Byzantine Empire. These are the seams where different electronic worlds ("cyberspaces") meet, and they are often unseemly.

It's usually easier to get through these things from the Internet side. Getting at them from the other side can be quite tricky, and fraught with incorrect or incomplete instructions, commands or addresses that don't work, etc. The easy way to go about E-mail is to start the correspondence from the Internet side. Then, the recipient can look at the address that the message says it's from - or just enter a Reply command, and you're off!

Gary Welles is on MCI Mail. From the Internet side, his address is "0001178863@mcimail.com"; presumably, 0001178863 is some kind of subscriber number. You can also express this in what is called "bang notation", where the part before the at-sign is moved to the end, separated off with an exclamation point ("bang"): "mcimail.com!0001178863". Compuserve accounts can be mailed to with the identical syntax, "<accountnumber>@compuserve.com".

AT&T Mail is different, because it uses X.400 internally. X.400, in case you haven't heard, is the new International Standard which the CCITT, under the auspices of the International Organization for Standardization (ISO), has developed for us since we need something to uniquely identify individuals for electronic mail purposes. You may have thought we already had a very nice system which works quite well, called SMTP (Simple Mail Transfer Protocol). That may be, however, Standards Bodies never simply bless an existing, working system when the opportunity arises to design Something Better from the ground up.

Anyhow, X.400 addresses, when written out in text form, have a bunch of sections separated by slashes in which the person's Administrative Domain, Private Domain, Organization, Surname, et al. are specified. They look something like "/A=US/P=Virtech/G=Rick/S=Rodman", or possibly worse.

At one time, I was able to send mail to an AT&T Mail subscriber by using an address "uunet!mhs.attmail.com!<company>/<X.400 stuff>" ("bang notation", notice). However, it's no longer working; I have no idea why. AT&T Mail subscribers can send mail to me with the address: "internet!virtech.vti.com!rickr/G=Rick/S=Rodman".

Next time

Next time we'll return to Minix, Linux, and maybe Uzi. Also, I'll discuss the dynamic linking in AmigaDOS, OS/2, and Sun Unix. In the meantime, keep your head high and your overhead low!

Where to call or write

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

SCSI BACK ISSUES

Here is a list of Back Issues that pertain to SCSI.

#22 - SCSI Introductory Column.
#23 - SCSI troduction to interface.
#24 - SCSI command Protocol.
#25 - Building a SCSI adapter.
#26 - Software for SCSI adapter.
#28 - SCSI for Real Time Control.
#31 - SCSI for General I/O.
#33 - SCSI for the S-100 Bus.
#48 - SCSI/Bernouli Drive for CP/M.

Issue 20 to 25 available in Volume 3, Otherwise all issues available separately. See pages 50/51 for list of back issues and how to order them.

Regular Feature

Kaypro Support

Power Supply Upgrade

Mr. Kaypro

By Charles B. Stafford

TODAY'S SURGERY

Wherein we solve a conundrum, perform a seemingly impossible transplant, and end forever, the cries for "More Power".

ODDS AND ENDS

In the days since we last met amongst these pages, several TRUTHS have been revealed to me via the magic of "snail mail" and the telephone. In keeping with my oath as a NEOPHYTE SCRIBE, I am honor bound to share them with you, and in fact I would anyway, since it's all NEAT STUFF.

TRUTH #1

When doing the K-II to K-IV upgrade, if you are nervous about soldering, even after all those hints and reassurances, you could just bend out the appropriate pins on the designated ICs and use those micro-clip test leads that Radio Shack sells to take the place of those soldered jumpers. They are really very solid when clipped on carefully, although, I suppose it would be possible to dislodge one or more if the machine travels very much. The drawback is that the ready-made test leads are really too long, and act as antennas for all the RF flying around inside the case. A better solution would be to buy just the clips themselves and make your own, so that they are as short as possible. A little slack won't hurt, but a lot will. You install the test clips by pulling up on the top, insert the wire, and push down exposing a tiny wire hook, which you put around the IC leg or connector pin, and which holds on for dear life when you release the pressure.

TRUTH #2

A "eutectic" solder (approximately 63/37) is superior to a 60/40 solder, both from the ease of use and the appearance standpoints. It melts more cleanly, i.e. for all practical purposes there is no "semi-molten" state, and the finished connection is brighter and smoother. An additional feature of eutectic solders is that they usually have small amounts gold and silver added to prevent them from dissolving trace amounts from the contacts you're trying to solder.

TRUTH #3

A couple of issues back, I mentioned that a good way to cure "screen jitters" was to implant a "wall wart" to supply power to the video circuit-board. I mentioned a "12v dc, 2 amp" rating. I have been reminded that the video circuit-board really only requires a 12v dc, 1 amp rating. It will probably be a lot easier to find one of these than the 2 amp variety.

These "TRUTHS", by the way, come to you courtesy of Lee Hart, an HCW, to whom I am indebted, AND who has a product called "Write Hand Man". It is similar to "Sidekick" (the DOS/OS2 product) and is at present ready for the HEATH/ZENITH platforms, as well as for Kaypros running the original CP/M operating systems. Lee can be reached at 323 West 19th, Holland MI 49423.

AND NOW, IN RING 3

This will probably be one of the most controversial, and down the road a piece, one of the most useful transplants to date. It isn't difficult or really fussy, but does require care and attention and is, I think, a very good example of "making do" with readily available components. The only critical part of this operation is paying attention to the color codes on the wires of your particular implant.

We are using "crimp-on" connectors for this transplant to simplify things and in the interest of both time and resource procurement. Not everyone has access to complete electronic supply houses, but there is a Radio Shack or auto parts store in almost every town. A "crimp-on" kit usually consists of the crimping tool, an assortment of connectors, and instructions and costs about $ 12.00 (cat # 64-409) at Radio Shack. There are other ways to make the connections, including new connectors, and solder and heat-shrink tubing for the purists and HCWs, but the "crimp-on" connectors are by far the most convenient for us amateurs.

As most Kaypro keepers are aware, the original power supplies were marginal, as far as capacity, and the wave soldering didn't always "glue" the connector pins to the power supply circuit-board very well. The results have varied from "unreliable boot" to "lunched hard drives" to "smoked" power supplies. The original power supplies were 65 watt versions, made by three different manufacturers, Cal DC, Boschert, and Astec, all good companies. Later supplies were 85 watt versions, mostly made by Astec. none of them had regulated 12v dc, which accounts for the "screen jitters", and none are currently available. I have found some 100 watt Astec supplies in the local surplus electronics store, which had the same size connector, but the order of pins is different, which means rearranging the original connector and the 12v is still unregulated.

About two years ago, I needed a fan for a Kaypro 10, that had suffered a lightning strike, and being too cheap to buy one at Radio Shack, I dismantled a dead PC-XT power supply just to get at the fan. It turned out that the fan wouldn't work for my application because it was a 12vdc fan and the Kaypro uses a 120vac fan, but LO and BEHOLD, the actual guts of the power supply turned out to be a single circuit-board. The more I looked at it, the better it looked as a candidate for this kind of transplant. It is shorter than the original Kaypro supply, but a little wider, has a plug-in connection for the dc fan, 2 line connections for ac power, a switch for 110/220v, 2 motherboard power connectors, and 4 (count 'em 4) drive power connectors. It is also rated at a Phenomenal 150 watts !!! The power supply for this project was procured from System Masters in San Francisco $ 14.00 [Phone # (415) 822-3779]. The first one arrived missing one of the drive connectors. When I called to complain, they mentioned an RMA number, but Steven Chang, the Proprietor, called back within a minute and a half to tell me to forget the RMA number, and that another supply was on the way. In fact, it arrived the very next day !!!

PRELIMINARIES

First assemble all the things you think you'll need. We're going to start by preparing the power supply and then put that aside while we prepare the patient. Here's what you'll need:

TOOLS

Soldering iron (don't be frightened yet, they're only wire connections)
Solder (see TRUTH # 2)
2 Phillips screwdriver
1 Phillips screwdriver
3 inch common screwdriver
Diagonal cutters
Pliers or very small adjustable wrench
Wire stripper or suitable substitute
Crimping Tool
Electric or Manual Drill
1/8th inch drill bit
2 or 3 inch Masking or Duct tape
Volt/Ohm Meter
1 Bath Towel

NECESSARY PARTS

1 PC-XT power supply
8 18ga crimp-on wire connectors (sometimes called "butt splice" connectors)

PREPARING THE IMPLANT

Before we start dismantling the XT power supply, we need to verify the wire color code. The easiest way is use it to power a couple of drives, (the ones in the Kaypro will do, just disconnect the ribbon cable and the drive power connectors and plug in connectors fron the XT supply) and then measure the voltages of the variously colored wires. Usually Black is ground, and Red is +5v. Beyond that nothing is certain. (When you look at the Kaypro, All the wires are White, real helpful, right?) The power supply must be loaded (supplying power to something) in order to develop the right voltages.

The common colors are:

BLACK....___V + - (NORMALLY GROUND)
RED.........___V + - (NORMALLY +5V)
YELLOW..___V + - (NORMALLY +12V)
GREEN.....___V + -
BLUE........___V + -
ORANGE..___V + -

Use this as a wor' 'heet, write in the voltage values and circle the + or - as appropriate. Write in any other colors that are on your power supply and their voltages. You are likely to find -5v and -12v as well.

1. Disconnect your test rig and proceed with the disassembly of the power supply. (this is the fun part, I guess I never got past the taking clocks apart stage). Remove the screws holding the metal box together, and carefully take the top off. The fan will come with it, but you'll have to unplug the fan power connector from the circuit-board. Some power supplies won't have a connector here, so you'll have to cut the wires midway between the fan and the circuit-board. The on-off switch is usually wired direct to the circuit board, so cut those wires as close to the switch as you can. There is one more switch, that selects the input voltage, 110/220v. Cut the wires to this switch, and then using your VOM (Volt Ohm Meter) determine if is closed or open when 110v is selected (usually closed). Now remove the circuit-board from the bottom of the case.

2. If the voltage selector switch is closed in the 110v position, unsolder one of those wires from the circuit-board, and use the other one as a jumper to permanently select 110v.

If the switch is open in the 110v position just unsolder both wires that led to it.

3. Put the power supply aside now, and we'll prepare the Kaypro

PREPARING THE PATIENT

1. Using the #2 Phillips screwdriver and a modicum of care remove the 10 screws securing the "hood" and then remove the "hood itself.

2. Disconnect all the connectors from the motherboard, remove the screws on either side of the parallel connector using a #1 Phillips screwdriver, and on either side of the DB-25 connector(s) using a small (tiny) wrench or very very carefully with a pair of pliers. Remove the 2 #2 Phillips screws on the back of the case and the 2 holding the motherboard to the long standoff insulators, and remove the motherboard.

3. Disconnect and remove the existing power supply, saving the screws (8) and standoff insulators (4).

4. Remove the carrying handle.

5. Fold a bath towel in quarters, put on the work surface and put the Kaypro face (screen) down on it.

6. Position the "new" power supply on top of the back, approximately over the place where the original was, with the heat sink (the big aluminum bar) toward what would normally be the top of the Kaypro. Looking at the Kaypro from the back, line up the top left holes of the new power supply and the case. Square up the new power supply so that the top

edge of the circuit-board is parallel to the top of the case, and mark the other three holes.

7. Set the new power supply aside again, using the duct/masking tape make "tents" under (on the inside of the case) the places where the new holes will be to trap any metal shavings or splinters, and using the 1/8th inch drill, drill new holes.

8. Carefully debur the outside of the holes, squash the "tents" to pick up anything in them, remove the "tents", and set the case back down on its feet.

9. Reinstall the standoff insulators in the new holes, and reinstall the carrying handle.

We're now going to trace all the power wires that DON"T GO TO DRIVES, identify them as to voltage and polarity, and cut them within an inch of the original power connector, so that we can reuse them with the new power supply. This is the time to be extra careful, we're going to hook these wires up to the new power supply, based on your identification of them.

10. Trace the 2 wires from video circuit-board connector back to the original power supply connector, and by matching the connector and the pins on the power supply, identify the +12v and ground connections, and tag them with a piece of masking tape appropriately marked. Now cut them about 1 inch from the long power supply connector.

11. Trace the wires from the motherboard connector back to the long power supply connector, identify and label these before cutting. The motherboard is marked as well, so the ends should be the same.

12. Trace the wires from the front panel LED, identify, label & cut.

13. The only wires left connected to the original power supply connector should be those that go to the power switch and those get the same treatment, identify, label and cut. In addition, they should be removed from the switch, by just pulling the spade connectors off the blades on the switch.

GET THE SHOTGUN PA, IT'S TIME FOR A WEDDING

Now that both the patient and the implant are ready, it's time for the main event.

1. Identify The two "motherboard" connectors on the cables attached to the "new" power supply, and cut them off as close to the connectors as you can.

NOTE
From this point on, when the word "connect" is used, it includes stripping about 1/4" of insulation from the wires involved, and crimping on a "butt connector" (except where noted) to splice the wires.

2. Find the two wires with the connector that used to carry power to the XT power supply fan, and respecting polarity, connect them to the two wires that go to the video board connector.

3. Find the original "motherboard" power connector and wires, and connect the appropriate wires to the ones that were cut off in step 1, just above. (i.e. the XT power supply motherboard power cables)

4. Find the two wires with the spade connectors, and connect them to the XT power supply wires that used to go to the line power switch, making sure that the combined length is great enough to reach the Kaypro power switch when the XT power supply is mounted in the case.

5. Mount the XT power supply on the standoff insulators, with the heat sink up, and the cables toward the right side of the Kaypro when viewed from the front.

6. Re-connect the small 2 conductor connector, now attached to the video circuit-board connector, to the XT power supply.

7. Re-install the "motherboard", push the spade connectors back onto the blades of the power switch, re-install the motherboard power connector, and carefully re-check all your work.

All that's left is to turn on the power and test your implant.

FINAL OBSERVATIONS

There is plenty of room for improvisation in this project. The PC-XT power supply is not the only one that can be used, there is now a "mini AT" desktop/ "micro-tower" power supply that is physically smaller, that could be used equally as well. I chose the PC-XT variety because it was significantly less expensive. I have seen "double-stick" foam tape used to mount circuit-boards effectively, and while I wouldn't recommend it for a power supply in a portable (luggable) machine, the internal modem in my own machine is mounted to the motherboard shield that way. If you anticipate traveling internationally with your machine, you could retain the 110v/ 220v switch by mounting it with a couple of appropriately sized washers, in one of the ventilation slots on the back panel.

One Caveat

If you decide to use a power supply other than a PC-XT compatible, there may be one or more small daughter boards inside the case. These are usually for electro-magnetic radiation control, and you'll have to figure out where and how to mount them.

PREVIEWS

I'd like to be able to tell you what we're going to do next issue, but I really don't know. If you have any requests, just drop me a line or tell Bill.

To contact Mr. Kaypro, check out Chuck's ad on the inside back cover.

TCJ Center Fold

Special Feature

All Users

XEROX 820

THEORY OF OPERATION

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

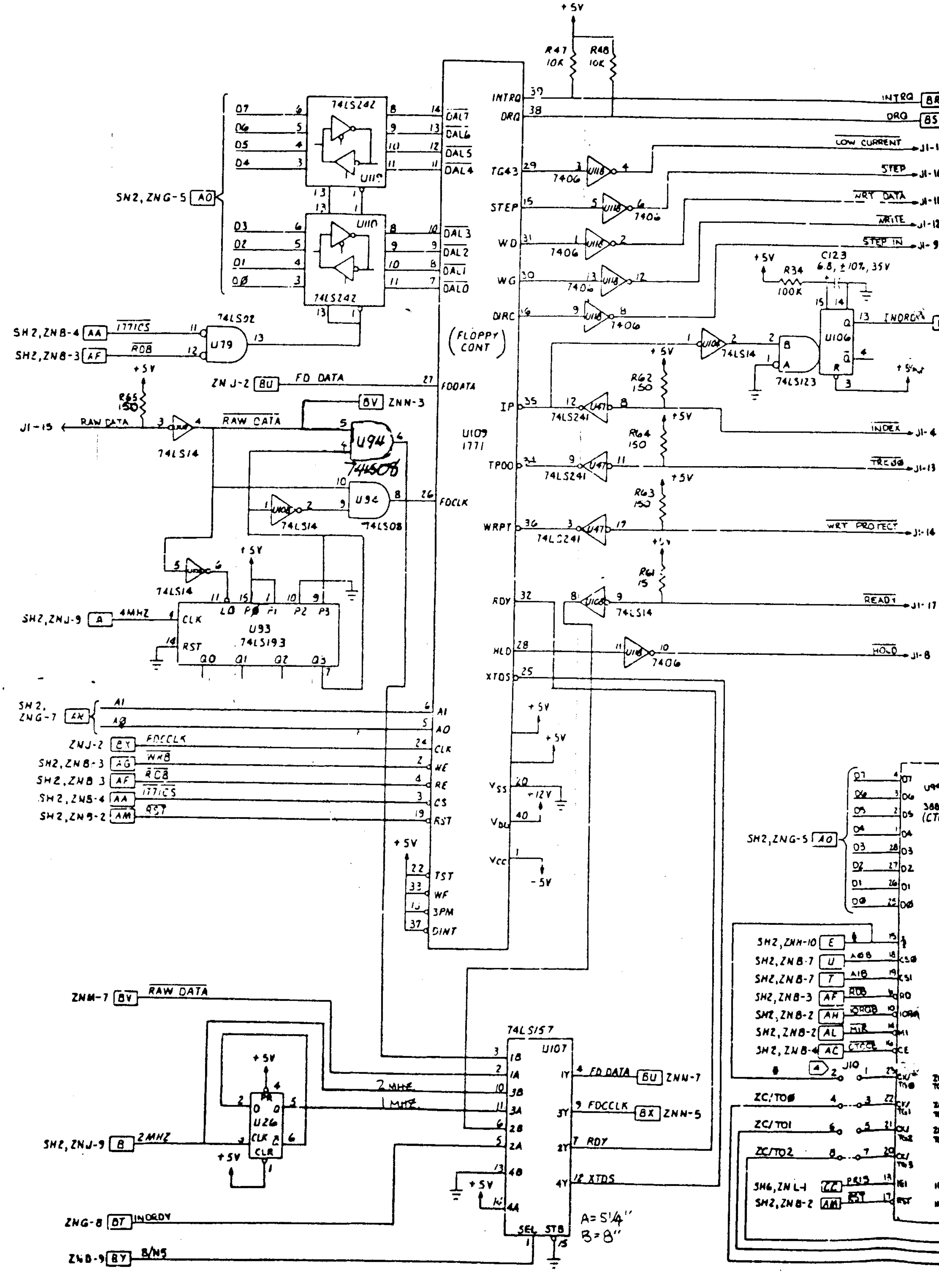
The devices that make up the CRT Controller provide interface for the display and CPU. The CRT Controller will convert data from the system data bus into Horizontal Sync, Vertical Sync and Video signals used by the display. The CRT Controller also handles the task of scrolling characters up the screen.

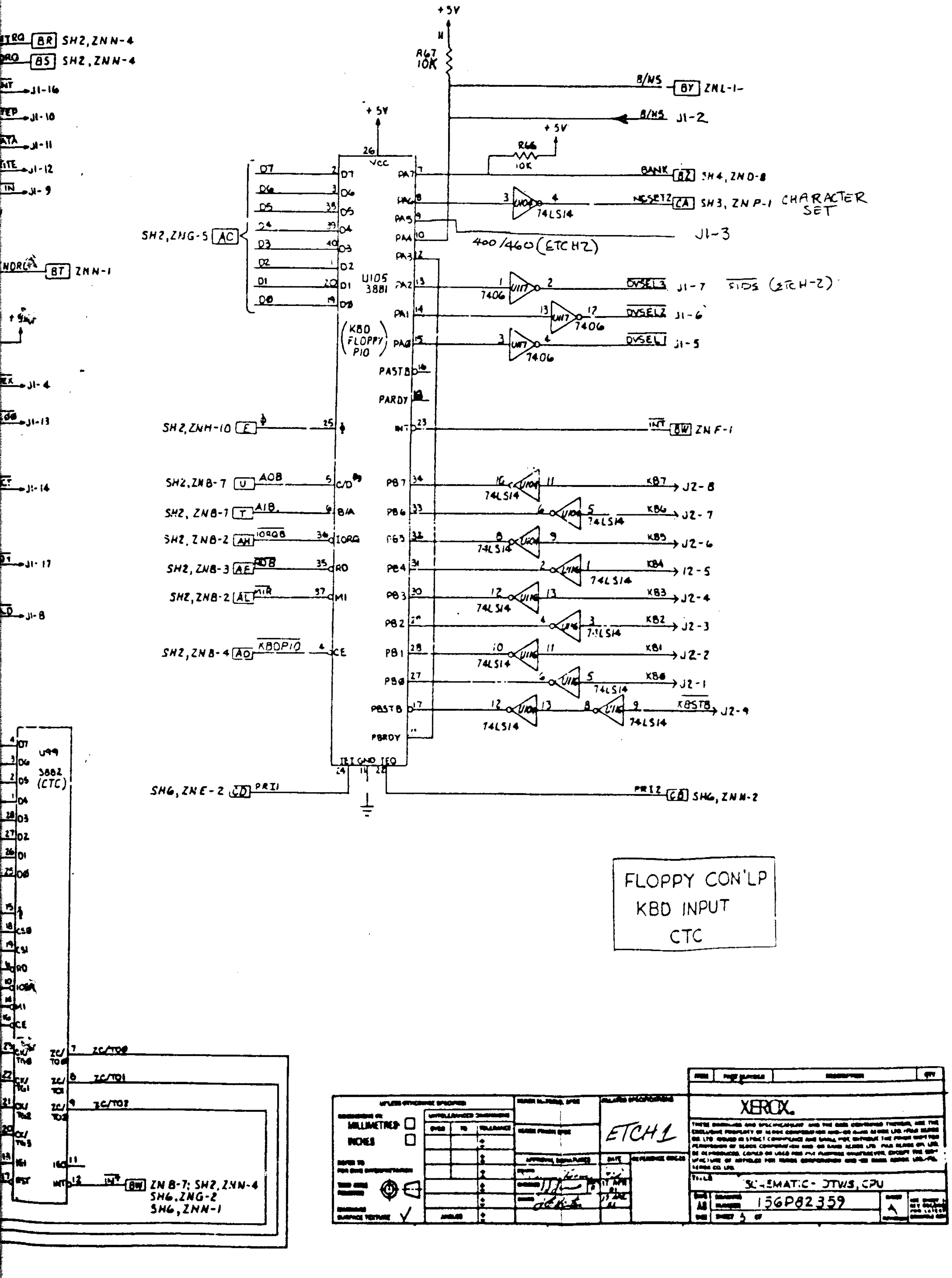
C. Timer Controller

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.


+5V
R47 10K
R48 10K
INTRQ 39
DRQ 38
INTRQ
DRQ
LOW CURRENT J1-1
STEP J1-1
WRT DATA J1-1
WRITE J1-12
STEP IN J1-9
TG43 29
STEP 15
WD 31
WG 30
DIRC 16
7406
U118
R34 100K
C123 6.8, ±10%, 35V
U106
74LS123
INDRDY
D7 D6 D5 D4
D3 D2 D1 DØ
74LS242
U119
U110
DAL7 DAL6 DAL5 DAL4
DAL3 DAL2 DAL1 DALO
SN2, ZNG-5 AO
SH2, ZNB-4 AA 1771CS
SH2, ZNB-3 AF RDB
74LS02
U79
(FLOPPY CONT)
ZN J-2 BU FD DATA
FDDATA 27
BV ZNN-3
J1-15 RAW DATA
R65 150
74LS14
RAW DATA
U94
74LS08
U109 1771
IP 35
TPOO 34
WRPT 36
RDY 32
HLD 28
XTDS 25
74LS241
U47
R62 150
R64 150
R63 150
R61 15
INDEX J1-4
TR00 J1-13
WRT PROTECT J1-14
READY J1-17
HOLD J1-8
U104
FDCLK 26
U93
74LS193
CLK RST LD PØ P1 P2 P3 Q0 Q1 Q2 Q3
SH2, ZNJ-9 A 4MHZ
SH2, ZNG-7 AN A1 AØ
ZNJ-2 BX FDCCLK
SH2, ZNB-3 AG WRB
SH2, ZNB 3 AF RDB
SH2, ZNB-4 AA 1771CS
SH2, ZN9-2 AM RST
A1 A0 CLK WE RE CS RST
VSS 20 +12V
VDD 40
VCC 1 -5V
TST 22
WF 33
3PM 16
DINT 37
ZNM-7 BV RAW DATA
74LS157
U107
1B 1A 3B 3A 2B 2A 4B 4A
SEL STB
2 MHZ
1 MHZ
1Y 4 FD DATA BU ZNN-7
3Y 9 FDCCLK BX ZNN-5
2Y 7 RDY
4Y 12 XTDS
U26
SH2, ZNJ-9 B 2MHZ
ZNG-8 BT INDRDY
ZNB-9 BY 8/NS
A=5¼"
B=8"
D7 D6 D5 D4 D3 D2 D1 DØ
SH2, ZNG-5 AO
SH2, ZNH-10 E
SH2, ZNB-7 U A08
SH2, ZNB-7 T A18
SH2, ZNB-3 AF RDB
SH2, ZNB-2 AH IORQB
SH2, ZNB-2 AL M1B
SH2, ZNB-4 AC CTCCS
J10
ZC/TO0
ZC/TO1
ZC/TO2
SH6, ZN L-1 ZZ PRIS
SH2, ZNB-2 AM RST


+5V
R67
10K
B/NS
BY ZNL-1-
B/NS J1-2
+5V
R66
10K
BANK BZ SH4, ZND-8
NSSET2 CA SH3, ZNP-1 CHARACTER SET
J1-3
400/460 (ETCH2)
U105
3881
(KBD FLOPPY PIO)
7406
DVSEL3 J1-7 SIDS (ETCH-2)
DVSEL2 J1-6
DVSEL1 J1-5
74LS14
SH2,ZNG-5 AC
SH2,ZNH-10 E
INT BW ZN F-1
SH2,ZNB-7 U A0B
SH2, ZNB-7 T A1B
SH2, ZNB-2 AH IORQB
SH2, ZNB-3 AE RDB
SH2,ZNB-2 AL M1R
SH2,ZNB-4 AD KBDPIO
KB7 J2-8
KB6 J2-7
KB5 J2-6
KB4 J2-5
KB3 J2-4
KB2 J2-3
KB1 J2-2
KB0 J2-1
KBSTB J2-9
SH6, ZNE-2 CD PRI1
PRI2 CB SH6, ZNH-2
BR SH2, ZNN-4
BS SH2, ZNN-4
BT ZNN-1
U99
3882
(CTC)
ZC/TO0
ZC/TO1
ZC/TO2
BW ZN B-7; SH2, ZNN-4
SH6, ZNG-2
SH6, ZNN-1
FLOPPY CON'LP
KBD INPUT
CTC
XEROX
ETCH 1
SCHEMATIC - DTWS, CPU
156P82359

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.

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 rece've 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 sto-ed 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 USA/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.

CONNECTING IDE DRIVES

by Tilmann Reh

Special Feature

Intermediate Users

Part 2: IDE Basics

Now it has been already one year since I described my 8-bit ECB-bus-based IDE interface here in TCJ. The delay in continuing with my description was caused by difficulties with the communication path between me and Bill Kibler. Since then, some questions have come up which were not covered by that article. So here is the missing information, I hope.

Remembering the Basics

Let us first have a short look at the drives we want to use. When discussing the different hard disk interfaces in my last article, I already pointed out that IDE drives of the AT-type, thus often called AT-bus drives (Bill Kibler calls them ATA drives, but this abbreviation is not the usual one, at least in Europe), are the ones with the best price/performance ratio one can get. This is even more the case now. So IDE drives still are the very first choice if you are looking for a good and cheap hard disk for your computer.

But what's special with those drives? I already mentioned that the IDE drive contains the complete hard disk controller. It is accessed with a system-bus interface compatible with the PC/AT (ISA) bus and offers control and data registers still compatible with the very first PC/AT hard disk controller (based on the WD 1010 controller chip). But even if those specifications come from something I don't like at all, why not use the low-price components for real computing (i.e., with a CPU280)?

Bringing the hard disk controller into the drive electronics offers some advantages. One of the main features is that you don't have a serial data stream with fixed bit rate between controller and drive. Thus, there's no need for conditioning the signals for the interface, and you can use any bit rate. As a result, the hard disk performance is limited by the drive technology, not by the interface's bit rate. This is one reason why today's drives are so much faster than the older ones. And technologies like Seagate's ZBR (Zone Bit Recording) are possible with hardware-independent interfaces.

There is another main feature of bringing the controller into the disk drive. Today's drives have very extensive checks for data security. They store error correction codes (ECC) together with the sector data and automatically correct single-bit errors, so the sector need not be re-read in those cases. Additionally, if a sector is found to be too unreliable, it is internally marked as bad and the data is mapped to a spare sector (usually there is one spare sector per track). All this is absolutely transparent to the user. So you now know the reason why today's intelligent drives don't have "defect lists" any more.

Since the PC's have such bad software (and hardware, too), there is another thing the integrated controller can do: translate virtual addressing information into physical. That means that the IDE drive is able to emulate another drive with different parameters (cylinder count, number of heads, and sectors per track). For the PC this is necessary because many PCs don't support drives with other than the historical 17 sectors per track, and many do not support free configuration of the drive parameters (only selection from a table is allowed). Also, some PCs mask off some bits of the cylinder number, since the first controller only had a 10-bit cylinder register -- so nearly every IDE drive still supports an emulation mode with less than 1024 cylinders and 17 sectors per track.

The IDE Interface

As mentioned above, the IDE interface is almost completely identical with a subset of the PC/AT expansion bus, so the drive can be connected (almost) directly to that. The only things required externally are two select signals (I/O address decoding). This gives us some information about how the interface works. In a PC the drive is accessed directly by the CPU via I/O accesses to registers internal to the drive. The disk data is transferred via the 16-bit data bus, but for compatibility to the older systems (again!) only 8 bits are used for command and status information. Besides the data bus, there are the standard Intel-type data strobe signals (/IORD and /IOWR), a few address lines, and some special signals. The connector is a 40-pin header, not to be confused with the XT-type IDE interface connector, which is also a 40-pin header but needs somewhat different hardware and totally different software!

The IDE interface allows connection of two drives with one cable. The second drive (slave) is then chained to the first one (master). However, I heard about problems when trying to connect different drives from different manufacturers. And the capacities of today's drive are so high that a single drive will always be enough for an 8-bit personal computer system! So, I never tried this option.

To understand the interface in detail, let's have a closer look at the IDE interface connector and its signals:

1	/RES	2	GND
3	D7	4	D8
5	D6	6	D9
7	D5	8	D10
9	D4	10	D11
11	D3	12	D12
13	D2	14	D13
15	D1	16	D14
17	D0	18	D15
19	GND	20	No Pin
21	/IOCHRDY	22	GND
23	/IOWR	24	GND
25	/IORD	26	GND
27	/IOCHRDY	28	ALE
29	No Connection	30	GND
31	IRQ	32	/IO16
33	A1	34	/PDIAG
35	A0	36	A2
37	/CS0	38	/CS1
39	/ACT	40	GND

The signals of the IDE interface can be collected in several groups: The general control signals are /RES (Reset) and / PDIAG (Passed Diagnostics). The data bus consists of 16 data lines (D0..D15). The access control lines are three address lines (A0..A2), the select signals /CS0 and /CS1 (Chip Select 0/1), and the strobe signals /IORD and /IOWR (and eventually ALE, the address strobe). The remaining signals (IOCHRDY, IRQ, /ACT, /IO16) are status signals.

Now Let's Go Into Details.

The reset signal normally is active-low. However, I heard about drives with an active-high reset signal, but I never saw one (or read such specifications). The /PDIAG pin carries a bidirectional signal used for chaining two IDE drives (master/slave). It normally can also be left open.

The data bus carries the 16-bit data words to and from the host. However, when accessing the control and status registers of the IDE drives, only data bits 0 through 7 are used (8-bit transfer). The data bus lines are tri-state lines that may be connected directly to the host's data bus. However, to meet the host bus specs and to avoid noise problems caused by the interface cable, a bus driver IC should be used to decouple the IDE bus and the host bus.

The drive is accessed using the selection signals /CS0 and / CS1. This also has historical (compatibility) reasons. Together with the three address lines, there could be two-times-eight addresses being occupied by an IDE drive. However, while the main register set really has eight registers and is accessed with /CS0 active, the other set (with /CS1) has only two valid addresses. We will have a deeper look at all the registers later. The data transfer is always strobed by the timing signals /IORD and /IOWR, for reading and writing, respectively. The address strobe (ALE) is often unused in the drive; it should be pulled high for static address lines (non-multiplexed busses).

The status signals are not absolutely needed to use IDE drives. Some of these signals are not commonly delivered at all (for example, /IOCHRDY (I/O Channel Ready), which is a WAIT signal for the host when the drive is much slower than the host processor in terms of interface access times). The /IO16 line informs the host of 16-bit transfers. Since we already know that data transfers are always 16-bit and everything else is always 8-bit, this is redundant (however, needed in the PC/ATs for their ISA bus). Line /ACT (Active) is an output which can be used for driving a drive-busy LED. Line IRQ is an interrupt request line that goes active on some internal events (if enabled by software).

Most IDE drives contain some jumpers that allow some options to be selected. This normally includes at least master/slave selection. Sometimes the /ACT signal may also be jumpered as an output signalling the presence of a second (slave) drive. The default state of the jumpers normally need not be changed (single drive, no special situation).

All interface lines carry CMOS-TTL-compatible signal levels. However, some signals (IRQ, /PDIAG, /IO16, /ACT) are able to drive higher currents. Those details should be looked up in each drive's specifications (for example, the /ACT output sinks 20 mA on my Conner drive, more than enough for an LED).

Accessing the drive is done with the following sequence of operations: First, the address lines and the chip selects must be set according to the desired register address. After some time (a minimum of 25 ns), /IORD or /IOWR is activated. This causes the data to appear on the data lines (when reading) or to be written to the drive (with the trailing edge of /IOWR, but there are setup and hold times to take care of). After a minimum of 80 ns, the strobe signal has to be removed. There are some more timing requirements, but these are the main ones.

The above timing details might differ from drive to drive. Always keep in mind that the IDE definition follows the PC/ AT system expansion bus and that official standards were not specified until two years ago, when the IEEE finally defined some specifictions (which many PC manufacturers are not following).

Unfortunately, I found that the drives do not match their own specs in every detail. For example, I found that the address lines of my Conner drive (a CP-3044 with 42 MB) must be kept stable for much more than the specified setup time. In addition, the drive is very sensitive to spike noise on the address lines, even if the noise appears long before an access is initiated. I spent a great deal of time struggling with such unlucky details (fixing other people's bugs).

IDE Interface Registers

Now that we've covered the interface signals and their meaning and usage, let's look at the registers of the interface. We saw that there are eight addresses being accessed through /CS0 and two addresses through /CS1. The following is a list of all the internal registers of an IDE drive:

/CS0	/CS1	A2	A1	A0	Addr.	Read Function	Write Function
0	1	0	0	0	1F0	Data Register	Data Register
0	1	0	0	1	1F1	Error Register	(Write Precomp Reg.)
0	1	0	1	0	1F2	Sector Count	Sector Count
0	1	0	1	1	1F3	Sector Number	Sector Number
0	1	1	0	0	1F4	Cylinder Low	Cylinder Low
0	1	1	0	1	1F5	Cylinder High	Cylinder High
0	1	1	1	0	1F6	SDH Register	SDH Register
0	1	1	1	1	1F7	Status Register	Command Register
1	0	1	1	0	3F6	Alternate Status	Digital Output
1	0	1	1	1	3F7	Drive Address	Not Used

The above addresses are those used in the PC/AT. Of course they are dependent on the decoding of the chip-select signals. The registers accessed via /CS1 might differ depending on the manufacturer of the drive. As far as I know, they don't always follow the compatibility principle with the first hard disk controller of the PC/AT.

The registers being accessed with /CS0 are also called the "Task File", so sometimes the IDE is also referenced to as "Task File Interface".

The error register can only be read. It contains valid information only if the error bit in the status register is set. Only five of the eight bits are used. They have the following meaning:

Bit 7: Bad block. This bit is set when the requested sector's ID contained a bad block mark (can be set when formatting the disk).

Bit 6: Uncorrectable data error. Set when the sector data can't be recreated (even with ECC).

Bit 4: Requested sector ID not found (wrong sector number).

Bit 2: Command was aborted due to drive status error or invalid command.

Bit 1: Track 0 has not been found when recalibrating.

The unused bits are always read as zero. However, I guess it's best not to rely on that!

The write precompensation register was previously used to set the starting cylinder for write precompensation (a slight shift of the serial data stream pulses to compensate for some magnetic effects on the disk surface). Since IDE drives handle all that internally, this function is not needed any more. Today, this register is often used as a parameter register for enabling or disabling look-ahead reading. We'll have a deeper look at that when talking about the various commands of IDE drives.

The sector-count register defines the number of sectors to be read or written with the next read/write command. A zero value causes 256 sectors to be processed, so the count varies from 1 to 256. This register is also used during drive initialization to specify the number of sectors per track (remember the emulation capability).

The sector-number register contains the starting sector number for any disk access. After a sector is processed, and after the command is completed, this register is updated. When an error occurs, this register contains the ID number of the erroneous sector. Normally, the sector numbers start with 1 and increase with each sector. However, by reformatting the disk, this order and the values may be changed.

The cylinder-low and cylinder-high registers contain the 10-bit cylinder number to be accessed. Since many drives have more than 1024 cylinders today, the cylinder-high register is often expanded to more than two bits. Like the sector-number register, these registers are updated after command completion and after errors. They are also used during drive initialization as the number-of-cylinders parameter.

The SDH register is a special register serving several functions. SDH is an abbreviation for "Sector size, Drive and Head". The bits of this register are arranged as follows:

Bit 7: Historical: Extension Bit. When zero, CRC data is appended to the sector's data fields. When set to one, no CRC data is appended. Since today's drives always use ECC error correction, this bit must always be set (no CRC).

Bit 6-5: Sector Size. Since today's drives always have 512-byte sectors (unchangeable by the user) because PCs are not able to support other sizes, these bits must always be 0-1.

Bit 4: Drive. This bit distinguishes between the two connected drives when using the master-slave chain. Single drives are always accessed with the drive bit set to zero.

Bit 3-0: Head number. These four bits contain the head number (that is, the disk surface number) for all following accesses. Similar to the cylinder and sector number, these bits are updated by the drive. The head number field is also used for drive initialization to specify the number of heads.

The read-only status register contains eight single-bit flags. It is updated at the completion of each command. If the busy bit is active, no other bits are valid. The index bit is valid independent of the applied command. The bit flags are:

Bit 7: Busy flag. When this flag is set, the task file registers must not be accessed due to internal operations.

Bit 6: Drive ready. This bit is set when the drive is up to speed and ready to accept a command. When there is an error, this bit is not updated until the next read of the status register, so it can be used to determine the cause of the error.

Bit 5: Drive write fault. Similar to "drive ready", this bit is not updated after an error.

Bit 4: Drive seek complete. This bit is set when the actuator of the drive's head is on track. This bit also is updated similarly to "drive ready".

Bit 3: Data request. This bit indicates that the drive is ready for a data transfer.

Bit 2: Corrected data flag. Set when there was a correctable data error and the data has been corrected.

Bit 1: Index. This bit is active once per disk revolution. May be used to determine rotational speed.

Bit 0: Error flag. This bit is set whenever an error occurs. The other bits in the status register and the bits in the error register will then contain further information about the cause of the error.

The command register is used to pass commands to the drive. There are many commands, not always using all parameters in the task file. Command execution begins immediately after the command is written to this register. Since this article is already quite long, I will cover the commands, their parameters, and their usage in another article, probably in the next TCJ issue.

The alternate status register contains the same information as the status register in the task file. The only difference is that reading this register does not imply interrupt acknowledge to reset a pending interrupt (as the main status register does).

The digital output register contains only two valid data bits. Bit 2 is the software reset bit, which causes a drive reset when being set, and bit 1 is the interrupt enable flag.

The drive-address register simply loops back the drive select bit and head select bits of the currently selected drive. This information normally is of no use for the programmer or user.

Last Words

Now that we had a look at the IDE interface, we also see the physical limits of this interface definition. With a fully expanded cylinder-high register, we are able to address up to 65536 cylinders, with up to 16 heads and up to 256 sectors per track. This results in a maximum addressable drive capacity of 128 gigabytes. I think this should be enough for microcomputing!! However, even if the PC/AT BIOS limitations are encountered, we could address 1024 cylinders with 16 heads and 64 sectors per tracks, giving 512 megabytes maximum capacity. This is also not bad, at least for small (8-bit) computer systems, where complete application software packages require only about 100 kilobytes of disk space.

Next time I would like to talk about the applicable commands of IDE drives and give examples of how to write software that accesses those drives. Perhaps I will also return to describing my IDE interface board for the 8-bit ECB bus in more detail. If you have questions or details about which you would like to read more, contact me at the following addresses:

Tilmann Reh
In der Grossenbach 46
D-57072 Siegen, Germany
e-Mail: tilmann.reh@hrz.uni-siegen.d400.de

List of Abbreviations:

AT	Advanced Technology	Class of PC's
BIOS	Basic I/O System	Hardware-dependent part of OS
CMOS	Complementary Metal-Oxid-Silicon	Semiconductor technology
CRC	Cyclic Redundancy Check	Error detection code, see also ECC
ECB	???	European standard 8-bit system bus
ECC	Error Correction Code	Additional data for security
IDE	Integrated Drive Electronics	Intelligent hard disk interface
IEEE	Institute of Electrical and Electronics Engineers	
I/O	Input/Output	(self-explanatory)
ISA	Industry Standard Architecture	PC/AT expansion bus
LED	Light Emmitting Diode	Optoelectronical component
OS	Operating System	Software which makes computers usable
PC	Personal Computer	Synonym for the worst computer architecture
TTL	Transistor-Transistor-Logic	Digital component standard (74xx series)
XT	eXtended Technology	Class of PCs, previous to AT
ZBR	Zone Bit Recording	Variable Density Recording Method

SCSI EPROM Programmer

By Terry Hazen

Special Feature

Intermediate Project

The Software

A Simple EPROM Programmer for the SCSI Bus, part II.

The SCSI Driver Software

Last time I presented the hardware design for a simple EPROM programmer board for the SCSI bus. This time we'll look at the software drivers required to communicate with the board. Our driver software controls the programmer board through the NCR5380 SCSI controller chip on the host computer board. Figure 1 shows the bit-mapped NCR5380 control registers used by the drivers.

I've written a complete EPROM programmer utility for this board that runs under ZCPR3 on Z80 and Z180/64180 computers. It's available as EPROG10.LBR on your favorite Z-Nodes. Since the source code for the whole program (included in the library) is too long to reproduce here, I've excerpted the SCSI driver code and presented it in commented Z80 assembly language form in Figure 2 so you can follow along as we look at each process.

Selecting the Programmer Board

The programmer board selection process starts by initializing the SCSI controller and setting it to target mode. Remember that all SCSI control and data lines are active low. The Current Bus Status Register is checked to see if the bus is busy (BSY\ asserted.) If the bus is busy, the SCSI controller is cleared and we quit with an error.

If the bus isn't busy, the programmer board ID bit is put on the data bus and SEL\ is asserted. Then we wait to see if the board answers, watching the Current Bus Status Register to see if BSY\ gets asserted. You'll recall from last time that when the board sees BSY\ deasserted and SEL\ asserted, it compares the state of the data bus with it's own ID and also makes sure the initiator's ID (7) is deasserted. If it makes the match, it sets its BUSY flip-flop, asserting BSY\.

After BSY\ has been asserted, we respond by deasserting SEL\, removing the ID from the bus and clearing the SCSI controller. Now we're ready to get to work.

Controlling the EPROM Address

The EPROM address is controlled by the outputs of a pair of cascaded binary counters, which have only two control inputs. The first input resets the counter to 0 and the second increments the current count by 1. Each of these inputs is controlled by a decoded combination of SCSI lines C/D\, MSG\, and I/O\ produced by U4. We reset the EPROM address by asserting MSG\ and I/O\ and strobing the REQ\ line to reset the counters. Incrementing the address is a similar process. We assert C/D\ and I/O\ and strobe REQ\ to clock the counters.

Reading the EPROM

When the bus phase is set to free (all control lines deasserted,) the board is in the read mode and the EPROM byte at the current address is on the data bus. All we have to do is read the byte from the SCSI Current Data Register.

Programming the EPROM

Programming is a little more complicated. We have several programming methods available. In the traditional method, you put the data to be programmed on the EPROM data bus and apply a 50ms negative pulse to the PGM\ pin to charge the EPROM's floating gate beyond a threshold condition. We'll look at this method first and then examine some faster possibilities.

The first step in programming a byte is to check if the byte is 0FFh, the erased state of the EPROM, with all bits set to 1. If it is, no programming is necessary and we can go on to the next byte. The next step is to set the write mode by asserting I/O\ and asserting the data bus bit in the Initiator Command Register so that the data bus is prepared to accept data. Now we can put the data byte in the Output Data Register, which places it on the data bus. We then write to the EPROM by asserting the REQ\ strobe line to place the programming pulse on the PGM\ pin, holding the strobe for 50ms, then deasserting it. Now we can deassert the data bus and read back the data byte to see if it was successfully programmed. If the operation was successful, we can go on to the next byte.

Fast Programming

There are several fast programming algorithms. Be sure to consult the spec sheets for the EPROM being programmed to get the details on the recommended fast programming method, programming voltages, the maximum number of programming cycles per byte and the maximum programming pulse width before you determine which fast programming method to use.

The basic fast programming method is to apply the program pulse for 1ms, then read the byte back to verify it. If the verification is unsuccessful, the programming write cycle is repeated. This is done some maximum number of times, often 20 or 25, until the byte can be verified or the retry count has been exceeded. Once the byte is verified, a final programming pulse is applied whose width is some function of the total programming time required to get verification. One method uses a pulse of the same length as total pulse width required for verification (1ms x number of 1ms write cycles.) Another method uses a final programming pulse of 3 times as much as the total pulse width required for verification (3 x 1ms x number of 1ms write cycles.)

Fast programming algorithms often specify an EPROM Vcc of 6v to make it easier to charge the floating EPROM gate to well over the threshold programming condition. Vcc switch S1 is provided to select between 5v and 6v.

Deselecting the Programmer Board

When we've finished and are ready to deselect the board, all we have to do is assert I/O\, C/D\ and MSG\, and strobe the REQ\ line to reset the board's BUSY flip-flop and deassert BSY\. When we see from the Current Bus Staus Register that BSY\ has been deasserted, we can clear the SCSI controller and we're done.

Testing the Board

EPROG10.LBR also contains SCSITEST.COM, a Z80/Z180 test utility to help you work out any problems you might encounter getting your EPROM programmer board up and running. It allows you to perform each of the driver operations separately and supports simple byte-by-byte programming.

EPROG10.COM and SCSITEST.COM both use a software timing loop with a ZCNFG-configurable time constant to create the programming pulse width. While the timing loop is automatically adjusted for the processor clock speed (as found in the ZCPR3 environment) and the different internal timing of processor types Z80 or Z180/64180, you can use SCSITEST to fine tune the pulse width in real time if you have an ocilloscope. With an empty EPROM socket, you can put an oscilloscope probe on the PGM\ pin, select the time constant function and measure the actual pulse width while SCSITEST provides a train of programming pulses. Pressing the '<' or '>' keys (or their unshifted equivalents) decreases or increases the time constant and displays its current value. When you are satisfied with the measured pulse width, you can exit the program and use ZCNFG to configure EPROG10's internal time constant to the displayed value.

Other SCSI Interface Applications...

Now that we've seen how easy it is to produce special-purpose SCSI boards, I hope some of you will be inspired to try your hands at designing useful SCSI boards. How about a simple SCSI 16L8 PAL programmer for us YASBEC users? Or a simple SCSI LAN?

What was that? My original YASBEC EPROM problem? Well, I used the EPROM programmer board and EPROG10 to copy the original YASBEC boot EPROM to a buffer and write it out to a binary file. I used ZP.COM to patch the file to add the few bytes of code needed to double the ZS180 clock speed to a DSEG area of the EPROM code. Then I used EPROG10 again with a fast programming algorithm to program a new (and faster) EPROM. Finally, I installed the new EPROM in my YASBEC, which now comes up very nicely at 18.4mhz snorting, stomping and pawing the ground! More Power!

```
Figure 1.  NCR5380 Control Registers

The SCSI EPROM programmer driver software uses the following five NCR5380
control registers to control the programmer board.  'SCSI' is the host
computer NCR5380 control port address.  All control register port addresses
are given as offsets from SCSI.  For more information on the NCR5380
registers, see TCJ#26, p12ff.

Current SCSI Data Register, Output SCSI Data Register (SCSI+0):

 Bit 7   Bit 6   Bit 5   Bit 4   Bit 3   Bit 2   Bit 1   Bit 0
+-------+-------+-------+-------+-------+-------+-------+------+
|  DB7  |  DB6  |  DB5  |  DB4  |  DB3  |  DB2  |  DB1 .|  DB0 |
+-------+-------+-------+-------+-------+-------+-------+------+

Initiator Command Register (SCSI+1) - Write Useage:

 Bit 7   Bit 6   Bit 5   Bit 4   Bit 3   Bit 2   Bit 1   Bit 0
+-------+-------+-------+-------+-------+-------+-------+-------+
|Assert | Test  | Diff  |Assert |Assert |Assert |Assert |Assert |
| RST   | Mode  |  En   | ACK   | BSY   | SEL   | ATN   | Data  |
+-------+-------+-------+-------+-------+-------+-------+-------+

Mode Register (SCSI+2):

 Bit 7   Bit 6   Bit 5   Bit 4   Bit 3   Bit 2   Bit 1   Bit 0
+-------+-------+-------+-------+-------+-------+-------+-------+
| Block | Target| Enable| Enable| Enable|Monitor| DMA   | Arbi- |
| Mode  |  Mode | Parity| Parity|  EOP  | BSY   | Mode  | trate |
| DMA   |       | Check | Int   |  Int  |       |       |       |
+-------+-------+-------+-------+-------+-------+-------+-------+

Target Command Register (SCSI+3) -

 Bit 7   Bit 6   Bit 5   Bit 4   Bit 3   Bit 2   Bit 1   Bit 0
+-------+-------+-------+-------+-------+-------+-----+-------+
|       |       |       |       |Assert |Assert |Assert|Assert|
|       |       |       |       |  REQ  |  MSG  |  C/D |  I/O |
+-------+-------+-------+-------+-------+-------+------+------+

Current SCSI Bus Status Register (SCSI+4):

 Bit 7   Bit 6   Bit 5   Bit 4   Bit 3   Bit 2   Bit 1   Bit 0
+-------+-------+-------+-------+-------+-------+-------+------+
| RST   | BSY   | REQ   | MSG   | C/D   | I/O   | SEL   | DBP  |
+-------+-------+-------+-------+-------+-------+-------+------+
; Figure 2.  Sample SCSI EPROM programmer board SCSI drivers.
;                         - Z80 assembly language -
; SELECT   - Select EPROM programmer board
; DSELECT - Deselect EPROM programmer board and remove it from the bus
; RDBYTE   - Read the EPROM byte at the current address
; WRTBYTE - Program the EPROM with the byte in A at the current address
; ADDRST   - Reset the EPROM address to 0
; ADDRSTP - Step the EPROM address by 1
;
; EPROM programmer board equates:
scsi        equu        58h         ; YASBEC SCSI port
target      equ         5           ; EPROM programmer board SCSI ID
;=================================================================
; Select the EPROM programmer board.  Once this routine is called,
; we must not perform any 'normal' SCSI operations until we call
; dselect.
; Exit: A=error code:
;              0=no error
;              1=SCSI bus occupied
;              2=target not responding (timeout)
;
select:     call        dstrobe     ; Set bus phase to free
            ld          a,01000000b             ; Set target mode
            ld          c,scsi+2    ; SCSI Mode Register
            out         (c),a       ; Target mode set
;
            call        status      ; Get current bus busy status
            jr          z,busfree   ; Bus is free
'
            ld          a,1         ; Else set SCSI bus occupied error
            jr          rstscsi     ; And return
;
, The bus is now ours...
busfree:    call        dstrobe     ; Set bus phase to free
            call        abus        ; Assert data bus
            ld          a,target    ; Put target ID on bus
            call        dataout
;
            ld          a,00000100b             ; Assert SEL\
            ld          c,scsi+1    ; SCSI Initiator Command Register
            out         (c),a       ; SEL\ asserted
;
; Wait for BSY\ to be asserted
            ld          b,10        ; Try 10 times
;
waitblp:    push        bc          ; Save counter
            call        status      ; Get current bus busy status
            pop         bc          ; Restore counter
            jp          nz,dbus     ; We've got the bus
'
            djnz        waitblp     ; Otherwise try again
;
timeout:    ld          a,2         ; Set target not responding error
            jr          rstscsi     ; And return
;=================================================================
; Deselect the EPROM programmer board, remove the interface from the
; SCSI bus and return the bus to normal operation.
; Exit: A=error code:
;              0=no error
,              4=target stuck on bus
;
dselect:    ld          a,00000111b             ; Assert I/O\, C/D\, MSG\
            ld          c,scsi+3    , SCSI Target Command Register
            out         (c),a       ; Release board from bus
;
            call        strobe      ; Assert strobe
            call        wait1ms     ; Wait 1ms for BSY to get off bus
            call        dstrobe     ; Deassert strobe
            call        status      ; Get bus status
            ld          a,0         ; Preserve flags and set good return
            jr          z,rstscsi   ; Not busy, so clean house and leave
            ld          a,4         ; Set target stuck on bus error
;
; Common SCSI return
rstscsi:    push        af          ; Save error code
            ld          a,00000000b             ; Deassert all control lines
            ld          c,scsi+1    ; SCSI Initiator Command Register
            out         (c),a       ; Bus phase set to free
            ld          c,scsi+2    ; SCSI Mode Register
            out         (c),a       ; Initiator mode set
;
            call        dataout     , Clear data register
            pop         af          ; Restore error code
            or          a           ; Set flags
            ret
;=================================================================
, Read byte from the EPROM at the current address.
; Exit: Byte returned in A
;
rdbyte:     push        bc          ; Save registers
            call        dstrobe     ; Set read mode (bus phase to free)
            ld          c,scsi+0    ; SCSI Current Data Register
            in          a,(c)       ; Get data byte in A
            pop         bc          ; Restore registers
            ret
;=================================================================
```

```
; Program EPROM with the byte in A at the current address.
; Exit: Z if programming was successful
;              NZ if unsuccessful
;
wrtbyte:push            hl          ; Save registers
            push        de
            push        af
            ld          (byte),a    ; Save data byte
;
            ld          a,00000001b             ; Assert I/O\
            ld          c,scsi+3    ; SCSI Target Command Register
            out         (c),a       ; Write mode set
            call        abus        ; Assert data bus
;
            pop         af          ; Restore data byte
            call        dataout     ; Put data byte on bus
            call        strobe      ; Initiate programming pulse
            call        wait50ms    ; Hold programming pulse for 50ms
            call        dstrobe     ; Terminate programming pulse
;
            call        dbus        ; Deassert data bus
            call        rdbyte      ; Read the byte back
            ld          hl,byte     ; Point to original data byte
            cp          (hl)        ; Same?
;
            push        af          ; Save flags
            call        dstrobe     ; Set bus phase to free
            call        dbus        ; Deassert data bus
            pop         af          ; Restore flags
            pop         bc          ; Restore registers
            pop         hl
            ret
;=================================================================
; Reset EPROM address to 0
;
addrst:     push        bc          ; Save registers
            ld          a,00000101b             ; Assert MSG\ and I/O\
            ld          c,scsi+3    ; SCSI Target Command Register
            out         (c),a       ; EPROM address reset
;
            call        strobe      ; Assert strobe
            call        dstrobe     ; Deassert strobe
            pop         bc
            ret
;=================================================================
; Step current EPROM address by one
;
addrstp:    push        bc          ; Save registers
            ld          a,00000011b             ; Assert C/D\, I/O\
            out         (c),a       ; EPROM address stepped
;
            call        strobe      ; Assert strobe
            call        dstrobe     ; Deassert strobe
            pop         bc          ; Restore registers
            ret
;=================================================================
 Subroutines getting data from a port:
. Get current bus busy status
;
status:     ld          c,scsi+4    ; SCSI Current Bus Status
            in          a,(c)       ; Get status
            and         01000000b ; Check BSY\ status
            ret
;---------------------------------------------------------
; Subroutines sending data out a port:
; Assert data bus
;
abus:       ld          a,00000001b             ; Assert data bus
            ld          c,scsi+1    ; SCSI Initiator Command Register
            out         (c),a       ; Bus ready to accept data
            ret
;
; Deassert data bus
dbus:       ld          a,00000000b; Deassert all control lines
            ld          c,scsi+1    ; SCSI Initiator Command Register
            out         (c),a       ; Data bus deasserted
            ret
;
; Put the byte in A on the data bus
dataout:    ld          c,scsi+0    ; SCSI Output Data Register
            out         (c),a       , Target ID is on bus
            ret
;
; Assert the REQ\ strobe
strobe:     ld          a,00001000b             ; Assert REQ/ - strobe
            ld          c,scsi+3    ; SCSI Target Command Register
            out         (c),a       ; Strobe asserted
            ret
;
; Deassert SCSI control lines REQ\, MSG\, I/O\ and C/D\.
dstrobe:    ld          a,00000000b             ; Deassert all control lines
            ld          c,scsi+3    ; SCSI Target Command Register
            out         (c),a       , Bus phase set to free
            ret
;---------------------------------------------------------
byte:       ds          1           ; Data byte storage
                                    ; end of software code
```

Regular Feature

Introductory Topic

6809 Operating Systems

Operating Systems

By Bill Kibler

In our ever on going quest for understanding about operating systems, we need to look at many different systems and their solutions. A number of our readers are using systems based on the Motorola 6809 CPU. A number of operating systems exist for these machines.

Back in the 70's Motorola produced the 6800 series of processors. At that time only the 6800 was available. Later 6801 and through to their power house version the 6809. A number of hardware and later software vendors appeared on the scene to support these machines.

The Hardware

The hardware came in many flavors, but the most popular for some time was the SS50 bus system. These systems are very similar to the concept of S-100 machines. The bus is a 50 vertical pin motherboard with corresponding female sockets on the plug in card. The I/O often is on the attached SS-30 bus.

The original products where made by South West Technical Products and later by GIMIX. The SWTP were the middle of the road in quality and price, while GIMIX was the Cadillac version in all respects. A number of other smaller vendors packaged products from either vendor while adding their own special cards. A few went so far as to make complete systems, and one or two of these vendors still have original stock for sale.

I contacted Jerry Koppel who ran AAA Chicago Computer Center and he indicates a few systems and some boards are still available. He also has some licensed copies of OS-9 still available for sale. His collection of boards ranges from a few assembled units to many unassembled bare boards. He recently moved and as such his stock is still in boxes and so he will need some time to find and check on what is available. Contact him by leaving a message on his phone at (708) 202-0150.

Jerry sold his own system under the Electra name which were a mix of his own cards with the cabinet and some cards from Hazelwood Computers. Mike Smith, the owner of Hazelwood Computers ((314) 236-4372) said he has a few 6809/SS50 boards available, but most would be bare boards. When questioned about selling them, like Jerry he is not really sure about what actual units and quantity is available. He is pretty sure he has 1Meg SS50 bare boards in large number and would sell them for about $10 plus shipping and handling (less than $25.00 for one including manual). Mike has been building systems for 15 years now; but since 1985 he has been producing only 68K based units. Based on Jerry's comments, Mike makes very high quality systems, both in design and operation.

One place that is also producing 68K boards as well as selling their older 6809 products is Peripheral Technology (404) 973-2156. Fredrick Brown has many 6809 single board units, some with Winchesters interfaces. He indicated he still sells one or two units a year of the 6809's, but mostly his 68K (single board and the PT-68K) are his current products. Fred sells OS-9 for the 68K units as well as REX (FLEX like with source code available) for his units. The interesting product is his ISA BUS compatible 68K mother board (called PT-68K that fits in standard PC/XT case and uses clone power supply and interface cards.) It was featured in August 1988 issue of Radio-Electronics (Computer Digest section) as a educational system (see SK*DOS products as well). For more details see his ad on the inside front cover.

Digital Research Computers of Garland Texas sold an English made system called "The 6809 Compacta UNIBOARD" which was similar in design to the Big Board Z80 system that produced the Xerox 820 and Kaypro systems. I have no idea if these products ever became successful or are still available. The major seller of single board systems however was the Tandy Computer company who made the Color Computer line of 6809 systems. They stopped selling them about three years ago, but new software and adapters are still being sold. I understand that calling the correct Tandy number will get you parts and software for the old CoCo's, even today.

All of these products could be bought with either FLEX by TSC or OS-9 from Microware. Both of which are not supported by their creators. The only operating systems worth getting is SK*DOS. SK*DOS is still being supported and is available for 6809 and 68000 systems.

FLEX

Flex by Technical Systems Consultants, Inc., of Chapel Hill, North Carolina was first on the market. It provided the normal compliment of disk utilities and standard operating system programming interface options. Like any operating system a programmer writes a program using the system interface calls. These calls are translated into the necessary internal operations and calls to external devices.

In reviewing FLEX documents it appears that most of the I/O options are more fixed and less user variable as was the case with CP/M. In CP/M the BIOS was designed and written by the maker of the system. Each vendor (and their were many vendors of CP/M systems) had their own BIOS and special enhancements as well as disk format. In the 6809 world there appeared to be mostly two vendors for most of the hardware. Those who came after these two, fashioned their work to look like the other two and thus the I/O or BIOS for these systems is more standardized than variable.

The FLEX manual describes the operating system as being composed of three parts, the file management system (FMS), the disk operating system (DOS), and the utility command set (UCS). From the manual " Part of the power of the overall system lies in the fact that the system can be greatly expanded by simply adding additional utility commands." We differ here from other DOS's in that there appears to be very limited internal commands.

Flex I/O is handled very much like PC CLONE systems in which you have the standard BIOS and are pretty much stuck with it's features. TSC is very clear about the FLEX systems not being user changeable and I believe that goes somewhat for the I/O options. Something similar to using TSR (Terminate and Stay Resident) programs is provided as an optional way of adding new features. In this option, ycu load a program that hooks the old calling address and replaces it with the new items entry point. Then when calling the old option, you are passed to the new program instead. This is an acceptable alternate way, but by far a better option is being able to simply change the actual BIOS as was done by CP/M users for years.

I contacted the current owners of FLEX and they are considering releasing the source code into public domain. The new company name is RTMX Inc. of Durham, NC and they are producing a true unix like real time operating system. I will be suggesting they submit their source to Walnut Creek CDROM as they are planning a orphaned CDROM. Walnut Creek has been collecting non-CP/M, NON-MAC, NON-UNIX, and NON-MSDOS files for this yet to be named disk. If you have any suggestions or orphaned programs (like FLEX programs), give Walnut Creek a call (800-786-9907).

Flex is simply a single user operating system providing reliable operation. UNIFlex and Miniflex were variations of Flex produced for multiuser or minimal systems respectively. These last two products never became successful or widely used. For more complex and multiuser operations OS-9 was the system of choice.

OS-9

During this period of time UNIX was being used by most colleges and as such was gaining popularity. Of importance was the multitasking features and device independent options of UNIX. UNIX provides pipes to direct data from one program or utility to another file, program, or device. These and other features influenced and controlled the design of OS-9. Unlike the Flex system which was available for 6800 and 6809 system, OS-9 was pr^^duced for the 6809 CPU only. The mu...asking or level II of OS-9 is for 6809 systems with a memory manager.

As we are starting to see the OS-9 system was much more advanced in both design and features than Flex. This same OS-9 system has been moved on to 68000 and 80386 platforms. OS-9 is also a real time operating system and is embedded in the CDI ROM systems currently on the market. OS-9 has been used by many large organizations including NASA and General Motors.

When used with memory manager and extra memory, multitasking systems can be built for the 6809 and level II of OS-9. Like Flex the system comes fixed for several of the standard 6809 hardware designs. Unlike Flex however, you can write your own device drivers and link them into the system. If you are a UNIX hacker you will find most of OS-9 very similar.

I have talked to the marketing director of Microware about the legal problems they have created by not supporting OS-9 for 6809 systems. Steve Johnson indicated they would not be releasing any source code (even though it is in 6809 assembler) since it gives away their design concepts which are still valid for the 68000 systems.

Steve indicated they are working at trying to find a way that helps the small user get legal systems and yet still protects the embedded system builder who is still selling and paying royalty fees. I hope to have an answer on this in the next issue.

SK*DOS

For the user putting together or buying a newer system, SK*DOS is the best option. Since you can not copy legally any OS-9 and I assume Flex systems, a source for new and supported operating systems is needed. Peter Stark has been selling his FLEX replacement for over TEN years. This is a mature and stable product that runs on both 6809 and 68000 systems. The two products are different but have the same disk format and close enough in other ways that the 68K SK*DOS has an SK*DOS-6809 emulator. So if you go to the newer 68K system, you can still use your investment of software on one system.

Unlike FLEX, SK*DOS is better at letting you do some of the work. The 68K version is really intended for you to do yourself. A separate BIOS is used and Peter has a configuration manual to show you how to do it. For the Micro Cornucopia reader, you might check out issues 46 and 47 where Karl Lunt describes how he brought up an SK*DOS 68K and created a multitasking system, all on surplus 68K boards.

When I talked to Peter, he made it very clear that he intends on continuing to support his product, and since his lively hood is NOT related to selling SK*DOS (he is a college professor), he can do so at reasonable prices as well. His SK*DOS goes for $75 for either 6809 or 68000 systems. Many standard configurations are available, so you might give him a call at (914) 241-0287 to see what he can do for your old system. His 24 page catalog lists many products as well as hardware platforms he can supply (even a 6802 running BASIC for $100!) He sells a "68000 Hardware Course" with a do-it-yourself approach, using the Peripheral Technology PT-68K mother board, and his $25 book (contact Peter for current pricing).

I pressed Peter for an article or a project using his system, and he indicated a desire to build a cheap and simple 68K single board system (bare boards to be made available) and show our readers how to bring up his system on it (a new variation of his older 68K Hardware Course). Since school is starting for him and he is not sure how many of our readers would be interested in such a project, Peter will have to do some thinking and planning before he decides to under take such a major project. A few cards and letters from those interested might help convince him to do that project (Star-K Software, Box 209, Mt. Kisco, NY 10549).

I might add that Peter has a number of other programs and ROM based products to support 6809/68K systems. All his prices are very reasonable and they all work as advertised. I believe he also has a program or two that will convert/ read other disk formats, so you can move data from your PC Clone to SK*DOS.

Some gotcha's

I have a GIMIX systems and it came with FLEX. The ROMs however are programmed to support both OS-9 and FLEX. One manual gives you the commands to boot the system using the W option. This states that it is a Jump to the Warmstart entry point. On first appearance you might think that is the correct command to start Flex, but it is not. If you check the Flex manual it indicates a U command for GIMIX Flex operation. Should the system crash and you are left at the monitor, then you can get back into Flex by using the W command. From power up Flex is not loaded and using the W command only jumps to empty memory. After Flex is loaded however, a program will be where W jumps and thus let you back into the operating system.

This problem got me several times, before I read more closely what the manual was saying. I first looked at it and just tried it with a system lockup being the results. Remembering that the system did work some time ago, I figured I was using the wrong command, which was the problem. Remember there are different ROMS for the Flex (simple BOOT) and the OS-9 (BIOS is in ROM code) operations and thus you must get to those parts differently using the 6809 CPU's monitor ROMs.

NEXT TIME

Next time we will look more closely insides SK*DOS and all the commands and programs available with it.

Available Software for SK*DOS/68K

The following partial list of SK*DOS software is coded as follows: UG = SK*DOS Users' Group; CSC = Computer Systems Consultants Inc.; PB = Palm Beach Software; SS = Spray Software; MC = Micronics Research Corp.; SKD = Supplied with SK*DOS System disk; BBS = Avaiable for downloading from the SK*DOS BBS at (914) 241-3307; ME = Michael Evenson BBS at (817)488-8398; OU = Other Users.

ACAT - Print alphabetized disk catalog (SKD)
APPEND - Append two disk files (SKD)
AREACODE - Finds telephone area codes (UG)
ASCII - Convert keypress into its ASCII code (SKD)
ASM - 68000/68010 Native assembler (SKD)
ASMK - Fast 68000 Assembler (PB)
ASMxx - Crossassemblers for Z80, 8048, 8085, many more (CSC)
BACKUP - Make a backup of a floppy disk (SKD)
BEEP - Add a beep to system prompt (SKD)
BIGCAL - Prints a big calendar (UG)
BUILD - Generate short text files without an editor (SKD)
C - Full K&R Compiler (UG or CSC)
CACHE - Cache program to speed up floppy disk operation (SKD)
CAL - Prints a calendar (UG)
CALLS - Does analysis of function calls within a C program (UG)
CAT - Print disk directory with additional data (SKD)
CCHECK - Check a C program indentation and comments (UG)
CHECKSUM - to checksum of disk contents (SKD)
CMODEM - Modem program for communications (UG)
CMON - Debugging monitor (OU)
CMP - File compare program by Jim Hughes (UG)
COMPAR - Compare two files (UG)
COMPARE - Compare two complete disks (SKD)
COPY - copy one or more files between disks, with options (SKD)
CURSOR - Adjust cursor type (OU)
CUT - Cut out columns from text files (UG)
DAMON - Display drive/trace/sector for each disk access (SKD)
DELETE - Delete a file from a disk (SKD)
DEVICE - Install a new I/O device on system (SKD)
DIFF - Display differences between two text files (SKD)
DIRS - Display current directories on a disk (P K Morrison/ BBS)
DIS - 68000/68010 Disassembler (UG)
DISKEDIT - Examine and edit disk files (UG)
DISKNAME - Display or change name or date of disk volume (SKD)
DOSPARAM - Display or change current DOS or device parameters (SKD)
DRIVE - Install or change active system drives (SKD)
ECHO - Echo command line with hex/octal/etc (from Unix) (UG)

EDDI - Screen Editor (PB)
EDLIN - Simple line editor (SKD)
ED - Simple editor, written in C, available in source (UG)
EDX - Unix "ED"-like line editor (UG)
ELIZA - Popular psychiatrist simulator game (SKD)
EMACS - Editor based on Micro Emacs (from Unix) (UG)
EZMODEM - Screen-oriented modem communications program (ME)
FDUMP - Hex file dump program to examine disk files (UG)
FIND - Find a text string inside a file (SKD)
FORMAT - Format a floppy disk (SKU)
FROMSDOS - Import a file from MSDOS to SK*DOS (SKD)
FTOH -Copy an entire floppy disk to a hard or RAM disk file (SKD)
FUNIQ - Remove duplicate lines from non-sorted file (does not sort) (UG)
GREP - Find strings within a file (from Unix) (UG)
HDFORMAT - Format a hard disk (SKU)
HECHO - Hex Echo program by Jim Hughes (UG)
HELP - Provides instructions to user on how to use SK*DOS (SKD)
HERC - Driver for Hercules monochrome board, emulates TVI 920C (UG)
HTOF - Opposite of FTOH above (SKD)
IOSTAT - Show I/O equipment currently recognized by system (SKD)
JINK - Relocatable Loader (OU)
KRACKER - 68000 file-to-file disassembler, text can be reassembled (PB)
LINK - Prepare a floppy disk for booting (SKD)
LIST - Display contents of a disk text file (SKD)
LOCATE - Display load addresses of a binary file (SKD)
LOGIC - Schematic drawing program (PB)
LONGSIDE - Prints two files side-by-side for comparison (UG)
LS - Similar to LIST command (from Unix) (UC)
MAKEMPTY - Generate an empty data file (SKD)
MEMTEST - Memory test program (UG)
MICRO-SPELL - Spelling checker (UG)
MODULA-2 - Compiler soon to come
MOZART - Music composition program (PB)
MSDIR - Display directory of MSDOS disk (ME,BBS)
MSREAD - Read an MSDOS disk (ME,BBS)
MSWRITE - Write an MSDOS disk (ME,BBS)
NCB - C beautifier progrrm to restructure a C program (UG)
NOBEEP - Oppsite of BEEP above (SKD)
NRO - "Runoff"-type text processor (from Unix) (UG)
PAGE - Unix-like progam to page through a text file (UG)
PARK - Park a hard drive before shutting it off (SKD)
PASTE - Paste data from multiple files together (UG)
PAT - Screen Editor (OU)
PDELETE - Prompted delete utility (SKU)
PEEK - Allow examining memory from SK*DOS command line (SKD)
POKE - Allow changing memory from SK*DOS command line (SKD)
POSITION - Position cursor on line (Frank Neuner /BBS)
PPR - Page printer program to format output to printer (UG)
PROMPT - Change SK*DOS prompt (SKD)
PROTECT - Change file attributes to delete protect etc. (SKD)
RAMDISK - Set up a RAM disk to speed up system (SKD)
RBASIC - Full Basic Interpreter (MC)
READ - Browse through text files backwards and forwards (SKD)
REDOFREE - Display disk parameters and rearrange free space (SKD)
RENAME - rename a disk file (SKD)
REPLACE - String replacement program (UG)
S4SKDOS - converts S4/S3 records to .com file (UG)
S4LOAD - convert S4/S3 records binary, for EPROM programmer (UG)
S4UNDOS - converts .com file to S4/S3 format file (UG)
S4UNLOAD - converts a binary memory image to S4/S3 (UG)
SC - Small C compiler (available in source and object) (UG)
SCAT - Sequenced catalog utility (SKD)
SCSIEVE - Sieve program for testing C compiler (UG)
SE - Screen editor (UG)
SEQUENCE - Change sequence number (for systems without clock) (SKD)
SFIND - String find program (UG)
SHAR - Archive program for combining files for transmission (UG)
SINSTR - Another version of SFIND (UG)
SKCC - C Compiler (CSC)
SLEEP - Time delay program for batch files (Frank Neuner/ BBS)
SORT - Sort a file by various fields (UG)
SPELLB - Spelling Checker (PB)
STEPRATE - Change default drive steprate time (SKD)
STONES - Awari-type game (UG)
STRINGS - Find text strings in text or command files (UG)
SUBCAT - Subdirectory Manager (PB)
SYSTEM - Change or display current system default drive (SKD)
TCAT - Display disk catalog with latest files on top (SKD)
TIME - Display or set clock/calendar setting (SKD)
TMODEM - Modem program for Xmodem protocol (from Unix) (UG)
TOLOWER - Change upper case files to upper/lower case 'SKD)
ıOMSDOS - Export files from SK*DOS to MSDOS (SKD)
TRACENAB - Set up a program for tracing (SKD)
UBASIC - Basic language interpreter (SKD)
UNCOMPR - uncompress an LZ compressed file done with compress (UG)
UNDELETE - Bring back a deleted file (SKD)
UNIQ - Delete duplicate lines from a sorted file (UG)
UNSQCMPR - Compare two text files for differences (from Unix) (UG)
VERIFY - Turn disk verify on or off (SKD)
VERSION - Display version number of a command file (SKD)
WC - Word count program (from Unix) (UG)
WHIMSICAL - Structured language compiler (SS)
WORK - Change or display current work default drive
XARC - File uncompress program (ME)
XREF - create a cross reference listing of a program file (C, asm. etc) (UG)
YASE - "Yet Another Screen Editor" (OU)

Special Feature

Intermediate Users

Part 2: Sharing the Bus

MULTIPROCESSING FOR THE IMPOVERISHED

by Brad Rodriguez

A TALKER PROGRAM FOR THE 6809

In our last episode, I supplied the design for a 6809 uniprocessor, and a simple test program to initialize the UART and exercise the serial port. Once that program is working, you can run some debugging software: a 6809 "talker" program (Listing 1). This program lets you examine and alter memory and registers, download code, and set breakpoints. The design of the talker follows my Z8 talker from TCJ #51 [ROD91]; the few changes are documented in the listing. You can use the host program from TCJ #51 for basic memory alterations, or you can modify the host program to allow editing of 6809 registers. The source code of Listing 1, and a 6809-tailored host program, are available from the Forth Roundtable on GEnie as file 6809TALK.ZIP.

GENERAL PRINCIPLES OF SHARED MEMORY

"You're either on the bus or off the bus."
- Ken Kesey, as reported in
The Electric Kool-Aid Acid Test

Figure 1 illustrates a simple multiprocessor system. The main bus (down the center of the diagram) is shared by all the CPU boards. Any memory or I/O board plugged into this bus is likewise shared. The bus carries address, data, and control signals between the CPUs and the memory or I/O -- just like the signals described in the previous installment which control data transfers between the 6809 and memory.

The important difference is: *only one CPU board can use the bus at any time.* If two CPUs try to send an address to memory simultaneously, they are likely to try driving each address line to different voltages -- with unpredictable, and possibly damaging results. The same is true of the data and control lines, of course. To avoid this, each CPU connects to the bus through a "three-state" buffer, so named because it can drive lines High (+5 volts), Low (0 volts), or *not drive them at all* (the third "state"). Only one CPU board can enable its three-state drivers at any time. We speak of this CPU "owning" the bus.

Obviously, we don't want the other CPUs to do nothing while they wait to use the bus. So each CPU has some "local" memory (the RAM and ROM shown for CPU #1). This memory can be accessed by the CPU chip, even when the three-state buffers are disabled. Similarly, each CPU board can have some "local" I/O. (The on-board address and data lines that interconnect the CPU chip, ROM, RAM, and three-state buffers are sometimes called the "local bus.")

So, each CPU runs normally until it tries to read or write to an "external" address, i.e., an address corresponding to one of the shared memory or I/O boards. (Each CPU decides for itself which of its addresses correspond to on-board memory, and which correspond to devices on the shared bus. This is a function of the address decoding logic.) When the CPU needs to use the bus, it asserts the REQUEST\ line, and then freezes all activity (enters a "wait" mode). When the bus becomes available to that CPU, its GRANT\ input will be asserted, telling the CPU that it can enable its three-state drivers and continue the memory cycle.

As long as the CPU continues to assert the REQUEST\ line, it keeps ownership of the bus. It is the responsibility of the CPU to de-assert the REQUEST\ line after a reasonable time, so that other CPUs may have a chance at the bus. The timing is shown in Figure 2. I use active-low REQUEST\ and GRANT\ signals; this is a matter of personal choice. Active-low signals are denoted in the text by a trailing backslash.

The problem of handling multiple simultaneous requests for the bus is left to the Bus Arbiter. I've shown this as a separate block in Figure 1. Many schemes exist for bus arbitration; I'll design a simple one in Part 3 of this series. For now, suffice it to say that the Bus Arbiter receives one or more REQUEST\ signals, and issues one and only one GRANT\ signal at a time.

REQUEST/GRANT LOGIC FOR THE 6809

In Part 1, the address range 4000-5FFF hex was reserved for references to the external bus, i.e., "off-board" references. Whenever the 6809 tries to read or write an address in this range, the signal CS4\ is asserted (pulled low) by the address decoding logic. This signal can trigger a request for the external bus.

Recall that the 6809 has an input line, MRDY, which tells the processor that the memory transfer is not yet complete -- as long as MRDY is held low, the memory cycle is "stretched."

This can "freeze" the 6809 while it waits for the external bus. (On the Z80, this signal is named WAIT\.)

We want the 6809 to "freeze" whenever:

a) it is trying to access an external address, and
b) it has NOT been granted the external bus.

The basic solution to this is shown in Figure 3. When CS4\ is asserted (low), the REQUEST\ for the external bus is asserted (low). While CS4\ is low and GRANT\ is high (bus not granted), MRDY will be held low. Since this freezes the address bus, CS4\ (and REQUEST\) will *remain* low. This continues until GRANT\ is pulled low, causing MRDY to go high, and allowing the cycle to complete. When the memory transfer is finished, CS4\ will go back high, de-asserting REQUEST\. And as long as CS4\ is high, the state of GRANT\ is a "don't-care." Note also that the three-state drivers are enabled when (and only when) GRANT\ is asserted.

Not so fast! Consider a memory read cycle. In Figure 3, when GRANT\ is asserted, the address and control are enabled onto the external bus, and MRDY is pulled high. But the 6809 interprets MRDY high to mean that *the cycle is complete and data is available.* There is no time for the memory to respond to the address and output its data!

The circuit in Figure 4 solves this by delaying the low-to-high transition of MRDY. When BUSWAIT\ goes low -- signifying a need for the CPU to wait -- MRDY goes low immediately (because of the active-low CLR inputs on the flip-flops). When BUSWAIT\ goes high, two clocks must occur before MRDY follows (because the high is clocked through two flip-flops). With a 4 MHz clock, this can be a delay of 250 to 500 nsec, depending on when in the clock cycle BUSWAIT\ goes high. Since we're making no assumptions about the Bus Arbiter, we have to assume that GRANT\ is an asynchronous signal -- that is, it can go high anytime during the 6809 clock cycle. Using two flip-flop stages guarantees a minimum of 250 nsec memory access time. Figure 5 illustrates the timing.

THE IMPORTANCE OF BEING INDIVISIBLE

We now leap forward to consider a software problem: how do we keep multiple CPUs from altering the same data structure simultaneously? This is the problem of "mutual exclusion" previously described in TCJ [ROD92]. The classic solution is the "semaphore," which works as well for multiple processors as it does for multiple tasks on a single processor...*provided* that the semaphore operators WAIT and SIGNAL are indivisible.

In a single-CPU system, a semaphore operator can be made indivisible by making it a single machine instruction. The 68000 provides the TAS (Test-And-Set) instruction for this purpose; but a memory rotate or increment/decrement can serve on the 6809 or Z80. The "gotcha" is that this one instruction requires *two* memory cycles: a read followed by a write. This is commonly called a "read-modify-write" (RMW)

WHY NOT PALS?

A TCJ reader asked why I'm designing with discrete logic instead of PALs. I have several reasons:

1. PALs -- especially 15 nsec or faster -- are still rather expensive. Jameco lists the most basic PALs at $2.50 each, and there is no way I could do the complete 6809 board with fewer than two. On the other hand, my junkbox is well stocked with 7400 series parts and wirewrap sockets.

2. Not everyone is equipped to program PALs. This is especially important for a journal like TCJ, which is trying to make its designs accessible to all readers. I have even had clients specify "no PALs," because they hadn't invested in the development and production hardware. (I, incidentally, do have a little-used PAL programmer.)

3. The logic design is "hidden" inside a PAL. This is aggravated when the minimal or "obvious" logic solution is not the PAL solution (frequently the case). Discrete logic, on the other hand, is easily understandable by any reader of TCJ (I hope).

4. Frankly, using PALs still feels like "cheating" to me. I remember the story told in *Soul of a New Machine*: when the engineers couldn't solve a hardware design problem, they just put in a block that said "PAL here." It feels like an admission of defeat.

5. I'm a member of the "old school" -- I still take pride in designing circuits with elegance and a minimum of logic. PALs only minimize the number of chips.

6. On the 6809 board, it wouldn't have saved that many chips. Excluding bus buffers and I/O drivers -- which PALs wouldn't replace -- there are only 5 "glue" IC's on the full board. Swapping that for two PALs isn't a great gain.

7. Perhaps most important, TCJ's editor specifically requested hardware articles that didn't use PALs!

Not all of these reasons would be valid in a commercial project. The one-time cost of PAL programmers is trivial for most companies. The added cost of PAL chips is offset by the reduced cost of assembly. "Cheating" to get a design out faster is simply good engineering, and the added flexibility for future design changes alone can justify the use of PALs. But this project is educational, written for impoverished newcomers to digital design. Here, I think discrete logic is best.

Comming Soon
Part III of 6809 CPU project

FIGURE 1. A THREE-PROCESSOR SYSTEM

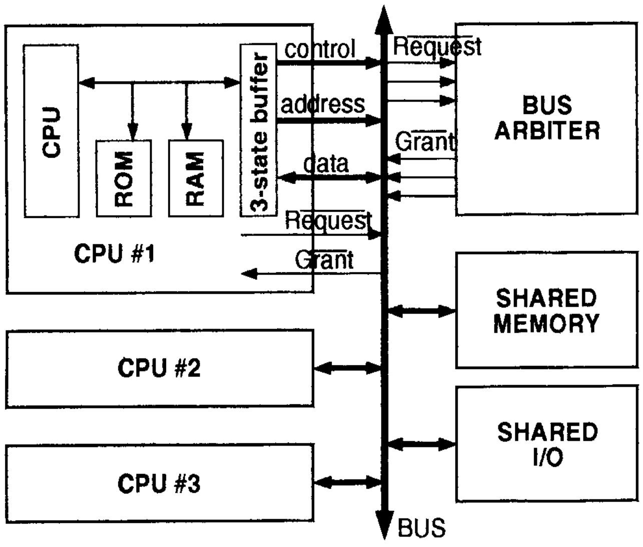


FIGURE 2. REQUEST/GRANT TIMING

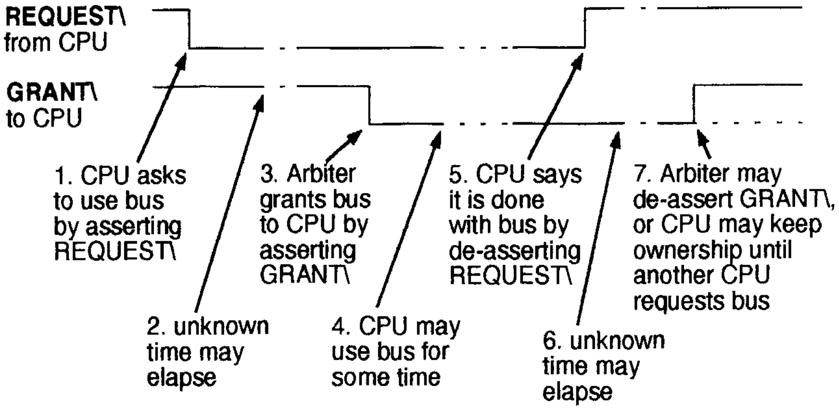

REQUEST\ from CPU
GRANT\ to CPU
1. CPU asks to use bus by asserting REQUEST\
2. unknown time may elapse
3. Arbiter grants bus to CPU by asserting GRANT\
4. CPU may use bus for some time
5. CPU says it is done with bus by de-asserting REQUEST\
6. unknown time may elapse
7. Arbiter may de-assert GRANT\, or CPU may keep ownership until another CPU requests bus

FIGURE 3. BASIC REQUEST LOC

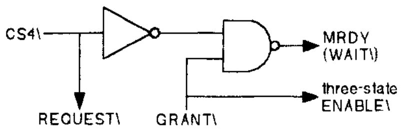


FIGURE 4. "FIXED" REQUEST LOGIC

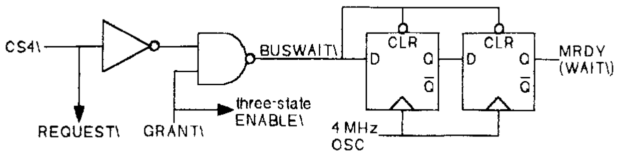


FIGURE 5. MRDY TIMING

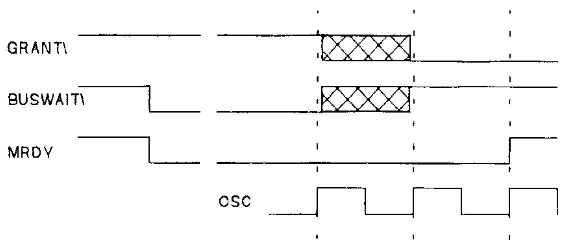


FIGURE 6. RMW LOGIC FOR Z80

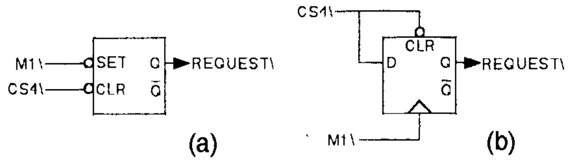


FIGURE 7. RMW LOGIC FOR 6809

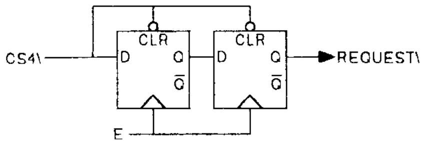
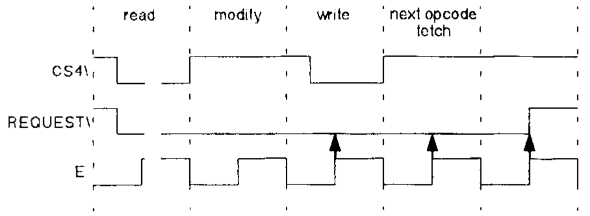


FIGURE 8. 6809 RMW TIMING



memory access: the memory byte is read, a bit is set (modified), and the byte is written back out to memory. Since "modify" takes some time, another CPU could obtain the bus between the read and write...and it's remotely possible that the second CPU would alter *the same sempahore* during that time. At megaHertz clock rates, even the "remotely possible" occurs too often!

The solution is obvious: the first CPU should not relinquish the bus until it has *completed* the read-modify-write operation. The REQUEST\ signal must be stretched accordingly. 68000s and 8086s provide a status signal for this purpose. For the 6809 or Z80, we must create our own.

RMW Logic for the Z80

The Z80 is the easier case. We can't necessarily tell when an RMW memory reference is complete, but we *can* tell when the next instruction begins. The Z80's M1\ signal is pulled low at the start of each new instruction (during opcode fetch).

Figure 6a shows a simple circuit to stretch REQUEST\. When CS4\ goes low (indicating an external access), it clears the flip-flop, forcing REQUEST\ low. Eventually the bus will be granted, releasing WAIT\, and the "read" part of the cycle will occur. During the "modify" part of the cycle, CS4\ may go high, but REQUEST\ will remain low. For the "write," CS4\ will go low again, but -- since this CPU still owns the bus -- GRANT\ is still low, and no wait states will be inserted. When the next instruction begins, M1\ will go low, forcing REQUEST\ back high and releasing the bus.

The circuit of Figure 6a will not work if opcodes are being fetched from external memory, since CS4\ and M1\ will be low simultaneously. (This usually will force both flip-flop outputs high.) One fix would use CS4\ to "qualify" the Set input to the flip-flop, so that when CS4\ is low the Set input is forced high. Another fix, (Figure 6b) uses the direct Set/Clear inputs to override the clocked data storage. The rising edge of M1\ will clock the flip-flop high, *unless* CS4\ is still asserted.

RMW Logic for the 6809

The 6809 doesn't provide a convenient M1\ signal like the Z80. Fortunately, the 6809 is completely deterministic, and its data sheet [MOT83] spells out the cycle-by-cycle execution of each machine instruction. For the shift, rotate, increment, and decrement instructions, the sequence is:

```
(varying number of cycles to get operand address)
cycle N-2:  output operand address, fetch data
cycle N-1:  output address FFFF, no data transfer
cycle N:    output operand address, store data
```

where each "cycle" is one cycle of the 6809's E clock. Thus, all read-modify-write operations on the 6809 place one "dead" cycle between the read and the write. So, RMW operations can be made indivisible by *stretching REQUEST\ by two extra E cycles.*

Figure 7 shows a circuit to do this. This is the same "stretching" circuit used previously, only it is synchronous -- the inputs always change at the same time relative to the E clock. Figure 8 gives an approximate timing diagram.

When the read is performed to an "external" address, CS4\ goes low and thus REQUEST\ immediately goes low. This read cycle may be stretched until the bus becomes available. (The E clock will also be stretched.) During the modify cycle, CS4\ will be high and a high level will be clocked into the first flip-flop. Before the high can be clocked into the second flip-flop, CS4\ will be pulled low for the write cycle. CS4\ must remain high for two consecutive cycles before REQUEST\ is de-asserted. This will occur sometime in the next instruction. (Unless, of course, the next instruction also accesses external memory. Note that this logic works properly if instructions are being fetched from external memory.)

OTHER CPUS?

The principles described here for the Z80 and 6809 can be adapted for most microprocessors. The CPU *must* have a WAIT input; thus the 8051 is not suitable. The CPU *should* have a memory-modify instruction (rotate, shift, or increment/decrement); otherwise, REQUEST\ has to be stretched over several machine instructions, and bus performance will suffer. This makes the 1802 a poor choice.

In the next installment I'll add this logic to the 6809 uniprocessor, design the bus arbiter, and add a few "frills."

REFERENCES

[MOT83] Motorola Inc., 8-Bit Microprocessor and Peripheral Motorola data book (1983).

[ROD91] Rodriguez, B. J., "A Z8 Talker and Host," The Computer Journal #51 (Jul/Aug 1991).

[ROD92] Rodriguez, B. J., "Forth Multitasking in a Nutshell," The Computer Journal #58 (Nov/Dec 1992).

```
        .command -ai    ; output in Intel hex format
;
;
;   MOTOROLA 6809 "MICRO-TALKER" MONITOR PROGRAM
,   (c) 1990, 1993 T-Recursive Technology
;   placed into the public domain for free and unrestricted use
;
;
;   A minimal monitor program for the Motorola 6809.
;   Used in conjunction with a "smart" host program to examine &
;   modify memory, and to set and execute breakpoints in machine
;   language and Forth.
;
;
;   The Talker program uses only registers for memory examine and
;   modify, and only a small amount of stack RAM for breakpoints.
,   Only about 300 bytes of PROM are used; no interrupts are used.
;   The Talker may be operated half-duplex over a bidirectional
;   serial line.
;
;
,   The program accepts characters from a Signetics 2681 DUART.
;   Characters 30h to 3Fh are treated as hex digits and are
,   shifted into a one-byte data register.
```

```
;    Characters 20h to 2Fh are command codes:
;      20-23 reserved for future use (ignored)
;     group 2: breakpoint / debug support
;      24 = set a Forth ''thread'' breakpoint
;      25 = set a Forth ''code field'' breakpoint
;      26 = set a machine language breakpoint
;      27 = fetch lo byte of address (adrs lo -> data & output)
;      28 = fetch hi byte of address (adrs hi -> data & output)
;      29 = read back data register  (data -> output)
;     group 1: minimal talker
;      2A = fetch byte & incr addr   (mem or reg -> data & output)
;      2B = store byte & incr addr   (data -> mem or reg)
;      2C = set lo byte of address   (data -> adrs lo)
;      2D = set hi byte of address   (data -> adrs hi)
;      2E = set memory ''page''      (no function on 6809)
;      2F = ''go''                   (jump to current adrs)
;
;    Internal register usage:
;      Y          = address
;      X (low byte) = data byte
;      D (A,B)    = working
;      S          = stack pointer
;
;    Revision history
;    v 1.0  23 Aug 89   original Super8 program
;    v 1.1  25 Feb 90   support for breakpoints and Forth words
;    v 1.2   7 May 90   function codes reassigned
;  Z8 v 1.0  2 Dec 90   modified for Zilog Z8
; 6809 v 1.0  25 Jul 93   modified for The Computer Journal 6809
;                 Uniprocessor
;
;
;
;    Macros for half-duplex communication on a bidirectional link,
;    e.g., a bidirectional RS-485 serial line.  Define these macros
;    to control the transceiver connected to the serial port.
;    If full-duplex is to be used (separate transmit and receive data
;    lines), define these as ''null'' macros.
;
;    N.B.: the PseudoSam Level I assembler does not support macros.
;    'TXON' and 'TXOFF' are commented out wherever used.
;
;    TXON - turns serial port transmitter on, and receiver off.
;    TXOFF - turns serial port transmitter off, and receiver on.
;
; txon   .macro
;        .endm
;
; txoff  .macro
;        .endm

;
;
;    STANDALONE INITIALIZATION
;
;    For use when the monitor is used as a standalone program
;    in a 6809 development board.  In this case, the monitor
;    is located in high PROM, to be started on reset.  The UART
;    registers are ''minimally'' initialized, to allow full-duplex
;    serial communication at 9600 baud, 8 bits, no parity.
;    The interrupts are vectored to a supplementary jump table
;    in RAM.
;
;    Expects:  reset state for all registers
;    Returns:
;    Uses:
;
;
            .org h'fed0

            .equ spsave,0       ; RAM location for saving S
            .equ vecs,2         ; RAM address of a jump table
            .equ stack,h'100     ; RAM address of bottom of stack

            .equ duart,h'6000    ; base address of 2681 DUART
            .equ txemt,8
            .equ txrdy,4
            .equ rxrdy,1

; 2681 Initialization Table.  Each word in this table contains
; register-number:contents in the hi:lo bytes, respectively.
;
initbl:     .dw h'022a ; Command Register A: reset rx, disable rx & tx
            .dw h'0230 ; Command Register A: reset tx
            .dw h'0240 ; Command Register A: reset error status
            .dw h'0210 ; Command Register A: reset MR pointer
            .dw h'0013 ; Mode Register A(1): 8 bits, no parity
            .dw h'0007 ; Mode Register A(2): 1 stop, RTS & CTS manual
            .dw h'01bb ; Clock Select A: tx & rx 9600 baud
            .dw h'0205 ; Command Register A: enable rx & tx
            .dw h'0a2a ; Command Register B: reset rx, disable rx & tx
            .dw h'0a30 ; Command Register B: reset tx
            .dw h'0a40 ; Command Register B: reset error status
            .dw h'0a10 ; Command Register B: reset MR pointer
            .dw h'0813 ; Mode Register B(1): 8 bits, no parity
            .dw h'0807 ; Mode Register B(2): 1 stop, RTS & CTS manual
            .dw h'09bb ; Clock Select B: tx & rx 9600 baud
            .dw h'0a05 ; Command Register B: enable rx & tx
            .dw h'0430 ; Aux Control Register: counter mode, xtal/16
            .dw h'062d ; Counter Upper, and
            .dw h'0700 ; Counter Lower: 2d00 hex = 50.000 msec
            .dw h'0d00 ; Output Port Configuration: all manual
            .dw h'0e03 ; Set Output Bits: OP0 and OP1 low
            .dw h'0500 ; Interrupt Mask Register: all disabled
endtbl:

; The program enters here on a reset.
; The stack pointer is initialized to a location in RAM.  This
; RAM is only needed if breakpoints are to be used -- the talker's
; memory examine & modify instructions don't use the stacks.

main:       lds #stack    ; set stack pointer to top of page zero
            clra          ; set direct page register to 0
            tfr a,dpr

; Initialize DUART from the table above.
;
            ldy #initbl
            ldx #duart
iloop:      ldd ,y++      ; fetch a:b from table
            stb a,x       ; store b at duart+a
            cmpy #endtbl
            bne iloop

            ; fall thru to talker

;
```

THE SCROUNGEMASTER II

Interested in a printed-circuit board for the 6809 multiprocessor? I've been exchanging email with TCJ reader Andrew Houghton, who wants to build a ''Poor Man's Transputer'' out of the 6809. This prompted some improvement and expansion of the original ScroungeMaster I design. Namely:

a) four RS-485 serial ports using two Z8530s, instead of the 2681
b) two parallel I/O ports, using a 6522
c) memory mapping logic for expanded memory: 32K on board PROM, 32K or 128K on-board RAM, and 384K of off-board (bus) address space

The driving principle is still Cheap Parts: I figure the increased cost of ICs over the ScroungeMaster I is about $9 (Jameco prices). No PALs.

Wire-wrapping one of these boards is enough of a headache -- I'd like to avoid wire-wrapping three more. If we can get commitments for twenty boards, I'll do the PCB layout and get boards fabricated at a local vendor. If interested, send me GEnie mail (B.RODRIGUEZ2), Internet email (b.rodriguez2@genie.geis.com), or drop a postcard to Brad Rodriguez, Box 77, McMaster University, 1280 Main Street West, Hamilton, Ontario L8S 1C0 Canada.

```
;
;    TALKER MAIN ENTRY POINT
;    MACHINE LANGUAGE BREAKPOINT ENTRY
;
;    This is the main ''talker'' program.  It performs the basic
;    character processing routine ''talk'' repeatedly, until a monitor
;    command transfers control to an application program.
;
;    This is also the entry point for machine language breakpoints.
;    A breakpoint consists of a software interrupt SWI3.  This
;    causes all registers to be pushed onto the stack.
;    Should it be desirable to have an interrupt cause a breakpoint
;    -- e.g., a ''break'' pushbutton wired to an interrupt input --
;    the same logic is used.
;
;    The breakpoint routine copies the saved flag values into the
;    talker's data register, and the saved return address into the
;    talker's address register.  An immediate ''go'' function will
;    resume with these saved values (as well as the saved rp).
;
,    Entering the breakpoint routine causes the '*' character to
,    be sent to the host.  Entering the monitor causes 'M' to be sent.
;    Note also that the main entry point puts its own address onto
,    the stack; this is so that we can ''go'' to a routine which ends
;    in a RET.  This ''normal termination'' can be distinguished from
;    a breakpoint by the 'M' character.
;
;    The stack frame built is:
;    CCR A B DPR Xhi Xlo Yhi Ylo Uhi Ulo PChi PClo TALKERhi TALKERlo
;    ^
;    |<-all of this is pushed by an interrupt,-> <-this is left->
;    S     and is popped by a ''go'' command.        when 'talker'
;                                                     is first entered.
;    Expects:
;    Returns:
;    Uses:  14 bytes of stack
;
;

talker:     ldd  #talker   ; push the address 'talker' under the 'go' frame
            pshs d
            pshs pc,u,y,x,dp,b,a,cc ; push a dummy return frame
            lda  #h'4d     ; 'M' ..talker program entry point
            bra t0


            ; Machine language breakpoints use SWI3, so they can use
            ; the same entry point as interrupt breakpoints.
            ; All registers are automatically saved in the return frame.
mbreak:     ; Machine language breakpoint entry
ibreak:     ; Interrupt breakpoint entry
            lda  #h'2a     ; '*'

            ; All entry points eventually land here
t0:         sts >spsave    ; save stack pointer where host can find it

t1:         ldb duart+1    ; transmit character in A
            andb #txrdy
            beq t1
            ; txon
            sta duart+3

t2:         ldb duart+1    ; wait 'til character finished
            andb #txemt
            beq t2
            ; txoff

            ldy 10,s       ; get return address in address register
            ldb ,s         ; get flags in data register
            clra
            tfr d,x
;
;
;    TALK - SINGLE CHARACTER PROCESSING ROUTINE
;
;    This routine processes one character received from the host.
;    If no character is received, it loops.
;    It may cause two characters to be transmitted to the host.
;
;    Expects:
;    Returns:
;    Uses:  D,X,Y,CCR
;
;

done:            ; talker is infinite loop
talk:       ldb duart+1    ; check for received character
            andb #rxrdy
            beq done

            ldb duart+3    ; get character
            clra           ; most operations require A=0
            andb #h'3f     ; mask off all but low 6 bits
            subb #h'30     ; if less than 30 -
            bcs cmd        ; - it's a command
```

```
;
; 30-3Fh: digit entry
;
digit:      andb #h'0f     ; 30-3f: hex digit, shift into lo nybble
            exg d,x        ; old MDR in D, for shifting
            aslb
            aslb
            aslb
            aslb           ; D=00n0  X=000n
            abx            ; add new to old, result in X
            bra done


;
; 24-2Fh: command codes
;    We use a jump table to select the appropriate function routine.
;    NOTE: Each routine is entered with Y=address, A=0, B=data.
;    D and X must be swapped back before going back to 'done'!
;    (The label 'cmdone' does this.)
;
cmd:        andb #h'0f     ; convert command codes to 0..0F
            aslb           ; *2
            exg d,x        ; table offset in X, data in D
            clra           ; many commands require this
            jmp [table,x]
                           ; function codes
table:      .dw cmdone     ; 20 - no op
            .dw cmdone     ; 21 - no op
            .dw cmdone     ; 22 - no op
            .dw cmdone     ; 23 - no op
            .dw settb      ; 24 - set thread breakpoint
            .dw setcb      ; 25 - set code field breakpoint
            .dw setmb      ; 26 - set machine code breakpoint
            .dw getlo      ; 27 - get low address
            .dw gethi      ; 28 - get high address
            .dw echo       ; 29 - get data register
            .dw fetch      ; 2a - read memory & send to host
            .dw store      ; 2b - store data at address
            .dw setlo      ; 2c - set low address
            .dw sethi      ; 2d - set high address
            .dw cmdone     ; 2e - set extended address (no op)
            .dw go         ; 2f - ''call'' address
;
; 24H: set a Forth ''thread'' breakpoint at the given address  (2 bytes)
;     this puts a the address of the ''breakpoint'' pseudo-word into
;     a Forth thread (a list of addresses of Forth words).  It is the
;     responsibility of the user to ensure that this is a valid thread
;     address, and to save the previous value.
;
settb:      ldx #tbreak
            stx ,y
            bra cmdone


;
; 25H: set a Forth ''code field'' breakpoint at the given address  (2 bytes)
;     this changes the code field associated with a given Forth word to
;     point to a machine-language breakpoint routine.  It is the
;     responsibility of the user to ensure that this is a valid code
;     field address, and to save the previous value.
;     >>>For the 6809 DTC Forth, this is the same as function 26H<<<
;
setcb:


;
; 26H: set a machine language breakpoint at the given address  (2 bytes)
;     this puts a machine-language SWI3 at the given address.  It is
;     the responsibility of the user to ensure that this is a valid
;     instruction address, and to save the previous value.
;
setmb:      ldx #h'113f    ; SWI3 instruction
            stx ,y
            bra cmdone


;
; 27H: copy low address byte to data register, and send to host
;     Note that this destroys the previous data register contents.
;     Use function 2Dh to save that value first, if needed.
;
getlo:      tfr y,d        ; address to data register
            clra           ; data register (hi) always zero!
            bra echo1      ; go send it


;
; 28H: copy high address byte to data register, and send to host
;     Note that this destroys the previous data register contents.
;     Use function 2Dh to save that value first, if needed.
;
gethi:      tfr y,d        ;
            exg a,b        ; address hi to data register (lo)
            clra           ; data reg (hi) always zero
            bra echo1      , go send it


;
; 29H: read back contents of data register
;
echo:
```

```
            .command -ai          ; output in Intel hex format
; ----------------------------------------------------------------------
; Simple test program for The Computer Journal's 6809 Uniprocessor
; B. Rodriguez  11 March 1993
; ----------------------------------------------------------------------
            .org h'ff00
            .equ duart,h'6000     ; base address of 2681 DUART
            .equ txrdy,4
            .equ rxrdy,1
; 2681 Initialization Table.  Each word in this table contains
; register-number:contents in the hi:lo bytes, respectively.
initbl:     .dw h'022a ; Command Register A: reset rx, disable rx & tx
            .dw h'0230 ; Command Register A: reset tx
             dw h'0240 ; Command Register A: reset error status
            .dw h'0210 ; Command Register A: reset MR pointer
            .dw h'0013 ; Mode Register A(1): 8 bits, no parity
            .dw h'0007 ; Mode Register A(2): 1 stop, RTS & CTS manual
            .dw h'01bb ; Clock Select A: tx & rx 9600 baud
            .dw h'0205 ; Command Register A: enable rx & tx
            .dw h'0a2a ; Command Register B: reset rx, disable rx & tx
            .dw h'0a30 ; Command Register B: reset tx
            .dw h'0a40 ; Command Register B: reset error status
            .dw h'0a10 ; Command Register B: reset MR pointer
            .dw h'0813 ; Mode Register B(1): 8 bits, no parity
            .dw h'0807 ; Mode Register B(2): 1 stop, RTS & CTS manual
            .dw h'09bb ; Clock Select B: tx & rx 9600 baud
            .dw h'0a05 ; Command Register B: enable rx & tx
            .dw h'0430 ; Aux Control Register: counter mode, xtal/16
            .dw h'062d ; Counter Upper, and
            .dw h'0700 ; Counter Lower: 2d00 hex = 50.000 msec
            .dw h'0d00 ; Output Port Configuration: all manual
            .dw h'0e03 ; Set Output Bits: OP0 and OP1 low
            .dw h'0500 ; Interrupt Mask Register: all disabled
endtbl:
; The test program enters here on a reset.
; This program doesn't use stacks, so the stack pointer doesn't
; need to be initialized.
entry:
; Initialize DUART from the table above.
echo1:                  ; txon - turn on transmitter (tx buf is empty)
            tfr b,a     ; transmit high nybble
            lsra
            lsra
            lsra
            lsra
            ora #h'30
            sta duart+3

tx1:        lda duart+1   ; wait 'til buffer ready for 2nd char
            anda #txrdy
            beq tx1

            tfr b,a     ; transmit low nybble
            anda #h'0f
            ora #h'30
            sta duart+3

tx2:        lda duart+1   ; wait 'til character finished
            anda #txemt
            beq tx2       ; txoff

cmdone:     clra        ; needed after 'echo', otherwise just in case
            exg d,x
            lbra done
;
; 2AH: fetch byte from memory and transmit
;
fetch:       ldb ,y+      ; fetch and increment address
            bra echo1     ; go send data
;
; 2BH: store byte in memory
;
store:      stb ,y+
            bra cmdone
;
; 2CH: set low address byte
;     D=00nn Y=hhll --> Y=hhnn D=00nn
setlo:      exg y,d
            clrb
            exg y,d      ; Y=hh00
            leay d,y     ; Y=hhnn
            bra cmdone
;
; 2DH: set high address byte
;     D=00nn Y=hhll --> Y=nnll
sethi:      exg y,d
            clra
            exg y,d      ; Y=00ll
            exg a,b
            leay d,y     ; Y=nnll
            exg a,b      ; D=00nn
            bra cmdone
,
; 2EH: set extended address byte (memory page)
; >>> not used on the 6809 <<<
;;
```

```
            ldy #initbl
            ldx #duart
iloop:      ldd ,y++     ; fetch a:b from table
            stb a,x      ; store b at duart+a
            cmpy #endtbl
            bne iloop
; Simple memory test, to check locations 2000 to 3FFF hex.
outer:      ; Memory test, one pass.
            lda #h'2e   ; '.' character means good
            ldx #h'2000 ; starting address
mtest:      ldb #1       ; rotate this through all bit positions
bittest:    stb ,x
            cmpb ,x
            bne bad
            aslb
            bne bittest
            bra good
bad:        anda #h'f5  ; error encountered: change '.' to '$'
good:       leax 1,x                  ; next address,
            cmpx #h'4000              ; and loop
            bne mtest
; Ouptut the character in the A register, over serial port A
tloop:      ldb duart+1
            andb #txrdy
            beq tloop
            sta duart+3
; Print all remaining ASCII characters up through 7F hex.
            inca
            bpl tloop
; Now wait for a received character, and echo it and all following characters.
rloop:      ldb duart+1
            andb #rxrdy
            beq rloop
            lda duart+3
            bra tloop
; The reset vector for the 6809 is located at address FFFE hex.
            .org h'fffe
            .dw  entry
            .end
; 2FH: go to given address (resume execution at given address)
;
; Note: if breakpoints are not being used,
;       go: jmp ,y
; is an acceptable substitute here
;
go:         sty 10,s     ; store program counter in return frame
            stb ,s        ; store flags in return frame
            orcc #h'80   ; force all regs popped on 'rti'
            rti           ; restore regs & go to given address

;
;    FORTH LANGUAGE BREAKPOINT ENTRIES
;
;
;    These are the entry points for Forth language breakpoints.
;    The first is the ''code field'' breakpoint.  This is an address
;    which can be stored in a Forth word's code field, to cause a
;    break whenever that word is executed.  This kind of breakpoint
;    can be set in any Forth word.
;    >>> In the 6809 Direct-Threaded-Code implementation, the
;     parameter field of every Forth word begins with machine code.
;     So, an ordinary machine-code breakpoint can be set in the
;     first byte of a word. <<<
;
;    The second kind is the ''thread'' breakpoint.  This is an address
;    which can be patched into a high-level ''thread'', to cause a
;    break when a certain point is reached in high-level code.  The
;    thread is a series of addresses of Forth words, executed by the
;    inner or ''NEXT'' interpreter.  So, we provide the address of a
;    dummy Forth word whose execution action is to invoke a machine
;    language breakpoint.  The Forth execution state can be deduced
;    from the registers.
;
;
;
            .equ cbreak,mbreak        ; the code field breakpoint is simply
; a machine language breakpoint set in a DTC ''code field'' what follows is the
; parameter and code field of a ''headerless'' Forth word, to invoke the breakpoint routine.
; (In DTC, this is simply a machine code fragment which does a breakpoint.)
tbreak: swi3                  ; the thread breakpoint is simply
                              ; a pointer to this code fragment

;                                                                    ;
;    6809 RESET AND INTERRUPT VECTORS
;

; Reset and SWI3 are used by the talker program.
; All other interrupts are vectored to a jump table in low RAM

            org h'fff2
            .dw  mbreak    ; SWI3 -  breakpoint
            .dw  vecs+0    ; SWI2
            .dw  vecs+4    ; FIRQ
             dw  vecs+8    ; IRQ
            .dw  vecs+h'0c  ; SWI
            .dw  vecs+h'10  ; NMI
            .dw  main      ; RESET - init & enter talker
            .end
```

REMINISCING and MUSINGS

By Frank Sergeant

Guest Feature

All Readers

Comment and Response

I'd always heard that potatoes can explode if baked without being stuck with a fork. I didn't doubt it was possible, but I never _really_ believed it until I ran a little experiment. It took many tries, but one day, Whoompf! potato _all_ over the oven!

So, until I finally try it, I probably won't _really_ believe an xyz table made with cheap threaded rod won't work.

Several people wrote in response to my article in #62. Thank you very much. I enjoyed the comments and suggestions. If you were on the verge of writing but didn't quite get around to it, please do write. I'd love to hear your thoughts on the various subjects I discussed last time.

One writer suggested printing PCB artwork on two transparencies and using them together to make the black lines dark enough. What a great idea! I had considered running the transparency through the printer twice to make it darker, but the registration isn't accurate enough for that. He thinks shooting a photosensitive board through a paper master won't work because of the amount of UV the paper would absorb. I mentioned last time that this _will_ work fairly well when shooting onto litho film.

Another writer emphasized the importance of the BIOS as "the software that knows the differences in the hardware." I knew that. I just said it was an urge. He also reminded me that the MS-DOS disk format is the standard, and must be supported. I knew that.

I've been doing a lot of thinking lately about what direction I might take my personal Forth, if it were to diverge from Pygmy. I have not come up with any satisfactory answers. I've considered a 32-bit Forth running in a large, flat address space. This would require a '386SX or better. Should it look like Pygmy? Should it run on the bare metal or require a DOS extender? Should it use DOS to start it and later save results, but otherwise run entirely out of RAM with no DOS services? This is how Charles Moore's latest "Forth" (his OK CAD system) does it. I haven't heard it mentioned, but it seems to me this last _requires_ an uninterruptable power supply (UPS). Imagine not only having to reset your VCR clock, but having to re-do several hours schematic or PCB layout work. Ooohh!

Just OK?

Chuck's OK system sounds very interesting. I saw it demonstrated several years ago at FORML, the Forth Modification Laboratory conference held every year over the Thanksgiving weekend in Asilomar, California. I was lost in the crowd, so I didn't get the best view. He was using a '386 PC clone with a VGA display and a 101-key keyboard, although he was only using 7 of the keys: ctrl, up, down, left, right, ins, del. It was entirely menu driven, so no text needed to be typed in. He says it is not suitable for word processing. He uses it to lay out the silicon for a Forth CPU he is designing. I've seen various figures on how many rows and columns he uses on the screen. I _think_ he had 15 rows and 20 columns. It was really pretty and I could _read_ it from a distance. I wish I were looking a characters that big and pretty right now as I type this article. However, he did not have lower case and did not have _all_ the upper case and punctuation.

'386 Assemblers

Surely you already know it, but the 80386 processor is a giant 32-bit machine hiding behind a 16-bit 8086 facade. Most PCs still use the '386 as just a fast 8086, without putting it into fancier modes. I've been writing '386 assemblers, trying to make one pretty. I haven't had much luck. As a comparison, the 8088/8086 assembler in Pygmy takes up about 22 blocks, without much crowding. The first of 3 recent attempts only generated 32-bit code, but fit in only 10 (fairly crowded) blocks. My goal is to make it fit in 3 blocks. The next two tries offered the option of generating either 32-bit or 16-bit code. Both took about 9 or 10 blocks. The 3-block solution seems completely out of reach as long as I insist on a full assembler. I think getting it down to 3 blocks will require an artist's eye to choose which instructions and addressing modes can be thrown away. All three assemblers run under Pygmy, a 16-bit Forth, but can generate 32-bit code. If they are destined for a large 32-bit address space, why should I care whether the source for the assembler takes 10 blocks instead of 3? Well, I can live with 10.

Compatibility

I keep returning to this. Maybe I'll just stick with Pygmy. It is already DOS and BIOS compatible and already works with the standard MS-DOS disk format. And, if I really need the 32-bit address space, I can use one of my 32-bit assemblers to temporarily change to 32-bit protected mode and then back again.

32-bit Protected Mode

In my Intel's house are many mansions. The '386 must be the 8th wonder of the world. I am thoroughly impressed that Man has designed and successfully implemented so massive a creation. Raise it from the miniature of silicon to a human scale and start strolling through its gardens, torture chambers, auditoriums, and gymnasiums. It is far larger than the Taj Mahal or all the Pyramids in Egypt put together. It has lots of hidden passages, with panels that open only if you press the secret buttons in the correct sequence. Otherwise, it's the mummy's curse for you, bud.

I finally worked out the kinks in getting into and back out of 32-bit protected mode. Getting _into_ 32-bit protected mode is fairly simple. It requires setting up an interrupt descriptor table and a global descriptor table, loading pointers to those tables, setting a bit in a control register, clearing the instruction prefetch queue, doing a long jump into a 32-bit segment, loading the segment registers, and setting up your new stack. (Did that sound simple to you?) Getting back out is what gave me a fit. I finally worked it all out, though. Leave out one little step and you get to reboot the computer and try again. And all this is easy compared to some of the things the '386 can do.

Southwest Texas State University

I survived the Unix programming course I mentioned last time. Too bad I can't say I enjoyed it. Once you've had a Human Factors course, every course becomes a Human Factors course.

The Computer Science Department secretary and the instructors of the the architecture and C courses keep talking like I'm really going to be teaching those labs in the fall. I think they are making a terrible mistake, but what do I know? Will I get into PLDs in the labs? Only time will tell. One correspondent highly recommends them, and points out that he needs to stock only two parts (16v8 and 22v10). Ok, building a PLD programmer just climbed a notch or two on my list of things to do.

FTP

Pygmy is now available via ''anonymous ftp'' from OAK.oakland.edu and other sites. Again, drop by the computer room of any local university and someone can probably show you all about ftp'ing files or even do it for you. Look for the file pygmy14.zip in the forth subdirectory.

Conclusion

I'd love to hear from you. If you have questions, suggestions, comments, solutions, or just want to set me straight, you can reach me at 809 W. San Antonio St., San Marcos, TX 78666, or via email to fs07675@academia.swt.edu on Internet or F.SERGEANT on GEnie.

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, hoe 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 check out ot Al Warsh, 2751 Reche Cyn Rd #93, Colton, CA 92324.

For Sale: ***Collector's Guide to Persnal Computers and Pocket Calculators.*** Prices and illustrations included. 336 pages. $14.95 plus $2.00 shipping. Fred Hartfield, Box 52466, New Orleans, LA 70152. Digitial Cottage BBS (504) 897-5514, help, support, sales, of old systems.

Now Avaiable: **96TPI Drives** for Kaypros, $12.00 + shipping. Mr. Kaypro, Chuck Stafford at (916) 483-0312 (eves).

Availabe: **Legal copies of CP/M** ($25), many bootable formats available. Disk copying, most formats including Apple CP/M. Manuals and more! Lambda Software Publishing. 149 West Hilliard Lane, Eugene, OR 97404-3057, (503) 688-3563.

Wanted: **User Manual**, 5.25 Boot disk and software, any Lazer Video Disks for a CP/M computer know as JOIN SYSTEM (ID 12428) made by C3 Inc. for military recruiting. Surplus in the 80's. Came with LDP1000 a Sony video player. Will pay reasonable amount for copies. DMT, PO Box 9064, Newark, NJ 07104.

Wanted: information about **Seequa Camilion** computer. Need boot disk and how-to information. This is a CP/M and MS/DOS machine. Edward Epps, 1000 Lenore St., Nashville, TN 37206.

Wanted: All **LCOMOTIVE** Software written in LOCASCRIPT. This is used on an AMSTRAD running CP/M+. Have several programs but need help getting started using the few programs that came with the machine. Looking for OTHELLO for Amstrad. Don Heeter, 484 Timothy Drive, Richmond, IN 47374-1009.

Classified ads are on a prepaid 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 subscribers who need to find old or missing documentation or software. Please limit your requests to one type of system.

The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535
B.Kibler@GEnis.com
GEnie = B.Kibler
CompuServe = 71563,2243
(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

TCJ *The Computer Journal*

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

Micro C Disks are $6.00 each plus shipping costs.

Shipping Cost to	U.S.	Canada/Mexico		Europe/Other	
		Surface	Air	Surface	Air
Added these costs	$1.00	$1.00	$1.25	$1.50	$2.50

Shipping costs are for GROUPS of 1 to 3 disks.

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.**

Volume Number 3:

- **Issues 20 to 25**
- Designing an 8035 SBC
- Using Apple Graphics from CP/M
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K
- Extending Turbo Pascal: series
- Unsoldering: The Arcane Art
- Analog Data Acquisition & Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC
- NEW-DOS: series
- Variability in the BDS C Standard Library
- The SCSI Interface: series
- Using Turbo Pascal ISAM Files
- The Ampro Little Board Column: series
- C Column: series
- The Z Column: series
- The SCSI Interface: Introduction to SCSI
- Editing the CP/M Operating System
- INDEXER: Turbo Pascal Program to Create an Index
- Selecting & Building a System
- Introduction to Assemble Code for CP/M
- Ampro 186 Column
- 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

The Computer Journal Back Issues

Issue Number 45:
- Embedded Systems for the Tenderfoot: Getting started with the 8031.
- The Z-System Corner: Using scripts with MEX.
- The Z-System and Turbo Pascal: Patching TURBO.COM to access the Z-System.
- Embedded Applications: Designing a Z80 RS-232 communications gateway, part 1.
- Advanced CP/M: String searches and tuning Jetfind.
- Animation with Turbo C: Part 2, screen interactions.
- Real Computing: The NS32000.
- The Computer Corner.

Issue Number 46:
- Build a Long Distance Printer Driver.
- Using the 8031's built-in UART for serial communications.
- Foundational Modules in Modula 2.
- The Z-System Corner: Patching The Word Plus spell checker, and the ZMATE macro text editor.
- Animation with Turbo C: Text in the graphics mode.
- Z80 Communications Gateway: Prototyping, Counter/Timers, and using the Z80 CTC.

Issue Number 47:
- Controlling Stepper Motors with the 68HC11F
- Z-System Corner: ZMATE Macro Language
- Using 8031 Interrupts
- T-1: What it is & Why You Need to Know
- ZCPR3 & Modula, Too
- Tips on Using LCDs: Interfacing to the 68HC705
- Real Computing: Debugging, NS32 Multi-tasking & Distributed Systems
- Long Distance Printer Driver: correction
- ROBO-SOG 90
- The Computer Corner

Issue Number 48:
- Fast Math Using Logarithms
- Forth and Forth Assembler
- Modula-2 and the TCAP
- Adding a Bernoulli Drive to a CP/M Computer (Building a SCSI Interface)
- Review of BDS "Z"
- PMATE/ZMATE Macros, Pt. 1
- Real Computing
- Z-System Corner: Patching MEX-Plus and TheWord, Using ZEX
- Z-Best Software
- The Computer Corner

Issue Number 49:
- Computer Network Power Protection
- Floppy Disk Alignment w/RTXEB, Pt. 1
- Motor Control with the F68HC11
- Controlling Home Heating & Lighting, Pt. 1
- Getting Started in Assembly Language
- LAN Basics
- PMATE/ZMATE Macros, Pt. 2
- Real Computing
- Z-System Corner
- Z-Best Software
- The Computer Corner

Issue Number 50:
- Offload a System CPU with the Z181
- Floppy Disk Alignment w/RTXEB, Pt. 2
- Motor Control with the F68HC11
- Modula-2 and the Command Line
- Controlling Home Heating & Lighting, Pt. 2
- Getting Started in Assembly Language Pt 2
- Local Area Networks
- Using the ZCPR3 IOP
- PMATE/ZMATE Macros, Pt. 3
- Z-System Corner, PCED
- Z-Best Software
- Real Computing, 32FX16, Caches
- The Computer Corner

Issue Number 51:
- Introducing the YASBEC
- Floppy Disk Alignment w/RTXEB, Pt 3
- High Speed Modems on Eight Bit Systems
- A Z8 Talker and Host
- Local Area Networks--Ethernet
- UNIX Connectivity on the Cheap
- PC Hard Disk Partition Table
- A Short Introduction to Forth
- Stepped Inference as a Technique for Intelligent Real-Time Embedded Control
- Real Computing, the 32CG160, Swordfish, DOS Command Processor
- PMATE/ZMATE Macros
- Z-System Corner, The Trenton Festival
- Z-Best Software, the Z3HELP System
- The Computer Corner

Issue Number 52:
- YASBEC, The Hardware
- An Arbitrary Waveform Generator, Pt. 1
- B.Y.O. Assembler...in Forth
- Getting Started in Assembly Language, Pt. 3
- The NZCOM IOP
- Servos and the F68HC11
- Z-System Corner, Programming for Compatibility
- Z-Best Software
- Real Computing, X10 Revisited
- PMATE/ZMATE Macros
- Controlling Home Heating & Lighting, Pt. 3
- The CPU280, A High Performance Single-Board Computer
- The Computer Corner

Issue Number 53:
- The CPU280
- Local Area Networks
- Am Arbitrary Waveform Generator
- Real Computing
- Zed Fest '91
- Z-System Corner
- Getting Started in Assembly Language
- The NZCOM IOP
- Z-BEST Software
- The Computer Corner

Issue Number 54:
- Z-System Corner
- B.Y.O. Assembler
- Local Area Networks
- Advanced CP/M
- ZCPR on a 16-Bit Intel Platform
- Real Computing
- Interrupts and the Z80
- 8 MHZ on a Ampro
- Hardware Heavenn
- What Zilog never told you about the Super8
- An Arbitary Waveform Generator
- The Development of TDOS
- The Computer Corner

Issue Number 55:
- Fuzzilogy 101
- The Cyclic Redundancy Check in Forth
- The Internetwork Protocol (IP)
- Z-System Corner
- Hardware Heaven
- Real Computing
- Remapping Disk Drives through the Virtual BIOS
- The Bumbling Mathmatician
- YASMEM
- Z-BEST Software
- The Computer Corner

Issue Number 56:
- TCJ - The Next Ten Years
- Input Expansion for 8031
- Connecting IDE Drives to 8-Bit Systems
- Real Computing
- 8 Queens in Forth
- Z-System Corner
- Kaypro-84 Direct File Transfers
- Analog Signal Generation
- The Computer Corner

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

Issue Number 62:
- SCSI EPROM Programmer
- Center Fold XEROX 820
- DR S-100
- Real Computing
- Moving Forth part III
- Z-System Corner
- Programming the 6526 CIA
- Reminiscing and Musings
- Modem Scripts
- 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		Europe/Other	
Subscriptions (CA not taxable)		(Surface)	(Air)	(Surface)	(Air)
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

Regular Feature

Editorial Comment

CP/M and LANS

Computer Corner

By Bill Kibler

TOO MANY ZX81's

Last issue I made a response to a readers request and opened up an endless supply of schematics. Yup folks, it seems the interest in ZX81 is still very active. Even seems some people have not given up on them. Several of the letters talked rather fondly about the product and what building one taught them.

Lets see, we have a letter from Michael Dantzer-Sorensen of Denmark who sent his German language copy of the schematic and assembly instructions. Seems the ZX81 was a kit product. Michael also mentions the Jupiter Ace which was a Forth in ROM version. I thought about getting one once until I saw the $300 price tag.

Ken Smyth sent his copy of ZX81 schematics and reminded me that Sinclair also made a 68008 version. He also indicated that the keyboard is a simple matrix and should be easy to adapt other keyboards. Ken is also looking for a copy of the "ASZMIC ROM", it turned the ZX81 into a assembly development platform. I'll settle for a copy of the Jupiter Ace ROM or better yet, source code?

Eric Sakara also sent me a copy, saying how he "cut his teeth" on the ZX81. He indicated that the Vancouver Sinclair Users Group was very advanced, even to the point of creating some multi-user software for the machine.

Tom Poulin wrote to say he had written an article some time ago, on repairing the keyboards. Maybe I will get Tom to review it with an eye toward repairing any membrane keyboard. What our readers probably would be more interested in is Tom's article describing elementary stepper motor circuits. Teaching stepper motor basics has been one of my long time projects. So send me a copy of the article, Tom!

Hetz Goldseger's letter (along with his copy of the schematic), pointed out some variations in the sockets, due to different RAM devices that could be used. He also modified a B&W TV for direct video. Hetz also appreciates the Xerox Schematics, as it is solving some of his problems.

Well that is it for today, at least as far as getting schematics! When I print the Schematic (issue 65 or 66), I will print all your letters in full detail. So if you have any more comments, don't need any more schematics (please!), send them along. I am especially interested in knowing about how well the ZX81 emulator works. Also information on the Vancover Sinclair Users Group would be nice to see.

Emulators???

Speaking of emulators, I got the Simtel20 CDROM from Walnut Creek and was amazed to see all the emulators it contained. Besides the Sinclair versions(4), there was 6800 with real time operating system, CP/M (7 + one CP/M86), two 68HC11 emulators, an Apple][, and a Commodore 64 as well. With the cheap price of clones, you can sure see why having access to old software on the new machines is popular.

Since I am running out of space, next time I will review emulators in more detail. I guess it is a great way to use new hardware with your favorite but old or broken hardware/software system.

Send those letters and cards to me, Bill Kibler, TCJ, BOX 535, Lincoln, CA, 95648.

Advertising Rates For *The Computer Journal*

Size	1 Insertion	2-3 Insertions	4+Insertions
Full	$400	$360	$320
1/2 Page	$240	$215	$195
1/3 Page	$195	$160	$145
1/4 Page	$160	$120	$100
Market Place	$50	$35	$35

Discover

The Z-Letter

The Z-letter is the only publication exclusively for CP/M and the Z-System. Eagle computers and Spellbinder support. Licensed CP/M distributor.

Subscriptions: $18 US, $22 Canada and Mexico, $36 Overseas. Write or call for free sample.

The Z-Letter
Lambda Software Publishing
149 West Hilliard Lane
Eugene, OR 97404-3057
(503) 688-3563

Advent Kaypro Upgrades

TurboROM. Allows flexible configuration of your entire system, read/write additional formats and more, only $35.

Replacement Floppy drives and Hard Drive Conversion Kits. Call or write for availability & pricing.

Call (916)483-0312
eves, weekends or write
Chuck Stafford
4000 Norris Ave.
Sacramento, CA 95821

***TCJ* MARKET PLACE**

Advertising for small business

First Insertion:	$50
Reinsertion:	$35

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

Mail ad or contact
The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535

CP/M SOFTWARE

100 page Public Domain Catalog, $8.50 plus $1.50 shipping and handling. New Digital Research CP/M 2.2 manual, $19.95 plus $3.00 shipping and handling. Also, MS/PC-DOS Software. Disk Copying, including AMSTRAD. Send self addressed, stamped envelope for free Flyer, Catalog $1.00

Elliam Associates
Box 2664
Atascadero, CA 93423
805-466-8440

8 BITS *and Change*

CLOSING OUT SALE!

All 12 Back Issues

for only $40

Send check to

Lee Bradley
24 East Cedar Street
Newington, CT 06111
(203) 666-3139 voice
(203) 665-1100 modem

S-100/IEEE-696

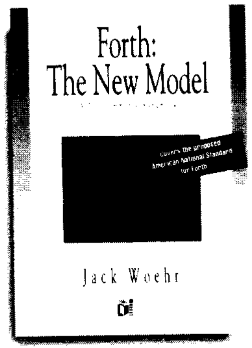
IMSAI Altair
Compupro Morrow
Cromemco
and more!

Cards • Docs • Systems

Dr. S-100

Herb Johnson,
CN 5256 #105,
Princeton, NJ 08543
(609) 771-1503

New from
M&T Books!



$44.95 1-55851-277-2

M&T BOOKS
Technical Books for
Technical Times

Available at bookstores
everywhere
or call 1-800-688-3987
RCJ3

PCB's in Minutes From LaserPrint!*

8 1/2" x 11"
Sheets
100% MBG

* Or Photocopier
Use household
iron to apply.

PnP BLUE or **PnP WET**

For High Precision
Professional PCB Layouts
1. **LaserPrint**
2. **Iron-On**
3. **Peel-Off**
4. **Etch**
An Extra Layer of Resist
for Super Fine Traces

Easy Hobby
Quality PCB's
1. **LaserPrint**
2. **Iron-On**
3. **Soak-Off w/ Water**
4. **Etch**
Transfers Laser or
Copier Toner as Resist

20Sh$30/40Sh$50/100Sh$100 Blue/Wet (No Mix)
Sample Pack 5 Shts Blue + 5 Shts Wet $20
VISA/MC/PO/CK/MO $4 S&H -- 2nd Day Mail
Techniks Inc. P.O. Box 463 Ringoes NJ 08551
(908)788-8249