Providing Support Around The World

The Computer Journal

Issue Number 69 September/October 1994 US$4.00

ISSN # 0748-9331

TCJ - For Having Fun With Any Computer!

Peripheral Technology Specials

486SLC 33MHZ Motherboard w/ CPU	**$139.00**
486SLC/50MHZ IBM,ISA,CPU, 0K	**$199.00**
486SLC/66MHZ IBM,VESA,CPU,Math	**$299.00**

IBM boards - Made in USA - 3YR warranty

1MB SIMM 70ns DRAM	**$47.00**
356MB Samsung IDE Drive	**$229.00**
420MB Connor IDE Drive	**$289.00**
546MB Maxtor IDE Drive	**$379.00**
IDE/Floppy/Serial/Parallel	**$24.95**
1.44MB TEAC Floppy	**$49.95**
Mini Tower, 200W, LED readout	**$79.00**
Panasonic Dual Speed CD ROM	**$169.00**
VGA Card ET4000-1MB,1280x1024	**$99.00**
VGA Monitor WEN .28mm 1024x768	**$249.00**

UPS Ground $7.00 on most items. Tower & monitor $12.00.

1250 E. Piedmont Rd. 404/973-2156
Marietta, GA 30062 FAX: 404/973-2170

Cross-Assemblers as low as $50.00
Simulators as low as $100.00
Cross-Disassemblers as low as $100.00
Developer Packages
as low as $200.00(a $50.00 Savings)

A New Project
Our line of macro Cross-assemblers are easy to use and full featured, including conditional assembly and unlimited include files.

Get It To Market--FAST
Don't wait until the hardware is finished to debug your software. Our Simulators can test your program logic before the hardware is built.

No Source!
A minor glitch has shown up in the firmware, and you can't find the original source program. Our line of disassemblers can help you re-create the original assembly language source.

Set To Go
Buy our developer package and the next time your boss says "Get to work.", you'll be ready for anything.

Quality Solutions
PseudoCorp has been providing quality solutions for microprocessor problems since 1985.

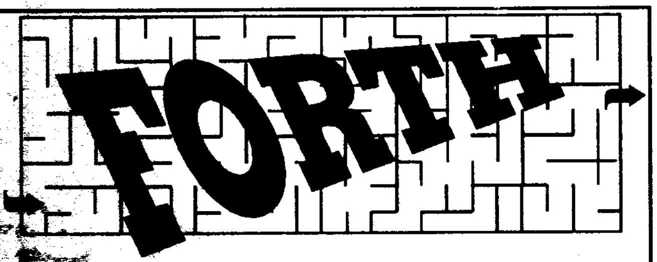
BROAD RANGE OF SUPPORT

- Currently we support the following microprocessor families (with more in development):

Intel 8048	RCA 1802,05	Intel 8051	Intel 8096
Motorola 6800	Motorola 6801	Motorola 68HC11	Motorola 6805
Hitachi 6301	Motorola 6809	MOS Tech 6502	WDC 65C02
Rockwell 65C02	Intel 8080,85	Zilog Z80	NSC 800
Hitachi HD64180	Motorola 68000,8	Motorola 68010	Intel 80C196

- All products require an IBM PC or compatible.

So What Are You Waiting For? Call us:
PseudoCorp
Professional Development Products Group
921 Country Club Road, Suite 200
Eugene, OR 97401
(503) 683-9173 FAX: (503) 683-9186 BBS: (503) 683-9076



Journey with us to discover the shortest path between programming problems and efficient solutions.

The Forth programming language is a model of simplicity: in about 16K, it can offer a complete development system in terms of compiler, editor, and assembler, as well as an interpretive mode to enhance debugging, profiling, and tracing.

As an "open" language, Forth lets you build new control-flow structures, and other compiler-oriented extensions that closed languages do not.

Forth Dimensions is the magazine to help you along this **journey. It is one of the benefits you receive as a member of the non-profit Forth Interest Group (FIG). Local chapters, the GEnie™ Forth RoundTable, and annual FORML conferences are also supported by FIG. To receive a mail-order catalog of Forth literature and disks, call 510-89-FORTH or write to:**
Forth Interest Group, P.O. Box 2154, Oakland, CA 94621. Membership dues begin at $40 for the U.S.A. and Canada. Student rates begin at $18 (with valid student I.D.).

GEnie is a trademark of General Electric.

SAGE MICROSYSTEMS EAST

Selling and Supporting the Best in 8-Bit Software

Z3PLUS or NZCOM (now only $20 each)
ZSDOS/ZDDOS date stamping BDOS ($30)
ZCPR34 source code ($15)
BackGrounder-ii ($20)
ZMATE text editor ($20)
BDS C for Z-system (only $30)
DSD: Dynamic Screen Debugger ($50)
4DOS "zsystem" for MSDOS ($65)
ZMAC macro-assembler ($45 with printed manual)

Kaypro DSDD and MSDOS 360K FORMATS ONLY
Order by phone, mail, or modem and use Check, VISA, or MasterCard. Please include $3.00 Shipping and Handling for each order.

Sage Microsystems East
1435 Centre Street
Newton Centre MA 02159-2469
(617) 965-3552 (voice 7PM to 11PM)
(617) 965-7259 BBS

TCJ The Computer Journal

Issue Number 69 September/October 1994

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Bill D. Kibler

Technical Consultant
Chris McEwen

Contributing Editors
Herb Johnson
Charles Stafford
Brad Rodriguez
Ronald W. Anderson
Helmut Jungkunz
Ron Mitchell
Dave Baldwin
Frank Sergeant
JW Weaver
Richard Rodman
Jay Sage
Tilmann Reh

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.

Entire contents copyright © 1993 by *The Computer Journal* and respective authors. All rights reserved. Reproduction in any form prohibited without express written permission of the publisher.

Subscription rates within the US: $24 one year (6 issues), $44 two years (12 issues). Send subscription, renewals, address changes, or advertising inquires to: *The Computer Journal*, P.O. Box 535, Lincoln,CA 95648.

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used trademarks are acknowledged, and we apologize for any we have overlooked.

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

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

EDITOR'S COMMENTS

Welcome to issue 69, a summer feast of articles. Since many of you were gone or busy having fun this summer, *TCJ* is playing catch up with our on going articles.

Your letters and comments start on the next page. A real mixed bag of comments this time. Certainly something for everyone.

Ron Anderson gets started on doing 6809 assembly for the Flex system. Since there are many readers still using Flex and many more wanting to better understand assembly, Ron is stepping in to help fill that space between ones ears.

Frank Sergeant finishes his recent XT travails by explaining how his Pygmy Forth helped him do some stepper motor activities. As usual we welcome your versions of Frank's Forth program steps in other languages. However as you will see, Frank makes it seem too simple to be true.

Ron Mitchell steps up next with his second installment of leading one into ZCPR for the first time. Like Ron's last entry in the journal, this should be educational and entertaining for all our readers. Now Ron, it's A> in the USA, EH> in Canada, but what about Europe and Australia?

Dr. S-100 drops in with his mail bag and tries to update our readers on the S-100 goings on. Herb doesn't stop there, as he co-authored (well mostly edited) the Centerfold section. We talked about it and now it is here, the S-100 IDE single chip project. This elegant design deserves your study and comments. Claude Palm will have more to say next issue on his newest single board Z180 project which incorporates a version of the single chip IDE interface. This shows of course how using more current, but expensive devices, makes multiple variations, simple and usually cheaper to do than redesigns using individual devices. Nice work Claude and thanks.

Getting another big thanks for ever struggling to keep our user's Kaypros running like new, is Charles Stafford as Mr. Kaypro. Chuck shows you how to layout the Advent Decoder board and start soldering. So get those soldering irons hot and start wiring and burning.

Rick Rodman's help with tying all our systems together get some assistance as he comments and shows us Tilmann Reh's RS232 to RS-485 interface. Tilmann's design is very nice since it isolates the signal electrically and thus prevents damaging your system, should another in the group go south for the winter. I know of one case where the power supply ground was not right and when the serial cables between this and another unit were touched together (well actually the person didn't even get them to touch) arcs and sparks went everywhere. Seems the grounds were somehow about 115 Volts different (smarts a bit) and thus blew most of the chips in the system. Nice addition to the design Tilmann!

Brad Rodriguez is still moving on with installment number 6 of his Moving Forth series. This installment completes the Z80/CPM version of his brand new Camel Forth. Next time Brad promises to give us the 8051 version, which I know many have been waiting for.

Since I received some comments on my Forth support, I felt it is about time for a little explanation of my interest in it. So the Computer Corner is mostly on what it is about polyFORTH I think you should consider knowing about. I also throw in some other tid bits of facts and fodder for your consideration.

So that is it for issue 69. Packed as always with goodies for your consumption.

A Death in the Family!

As many of you know by now, the creator if you will, of CP/M is dead. Gary Kildall died Monday, July 11, 1994. His death appears to be a side effect of not seeking medical help immediately after suffering an head injury.

I had considered commenting more on his work until I received *The Z-Letter.* David McGlone reviewed his life and activities so well I rather everyone just got a Z-letter.

Catching Up...

I am still catching up from being gone for most of two months. I believe I have sent all back issues and should have all renewals updated by the time you get this. Should you think your request has fallen through the cracks, please call and see if I stuck your request in the wrong "to do" pile. With that said, I am also trying to get ahead and want any articles to be here by the last week of October, else it will be in issue 71, not 70!

So lastly..Enjoy! Bill Kibler.

The Computer Journal
PO Box 535
Lincoln, CA 95648-0535

(916) 645-1670
(800) 424-8825
Genie as B.Kibler
Compuserve: 71563,2243
E-mail: b.kibler@Genie.geis.com

READER to READER

Letters to the Editor

All Readers

MINI Articles

Dear Bill,

Look forward to many more issues of *TCJ*. Debit my piece of plastic again, another years worth please. (Airmail rates please)

This (US$44-00) will run out at about NZ$90-00. Hurts a bit I confess; but then I'm paying by choice and figure that if that's what it costs to support an old habit, so be it. (I can think of many more expensive pursuits. Most if not all, far less intellectual challenging/stimulating.)

Read the article (Part 1 *TCJ* #64) 'Small-C?' with great interest. While most is out of my league; I think I grasped more about the scope of 'C' and its variations; than I'd previously read in any one small piece of print. Really looking forward to the follow ups.

I see no mention yet, in JW Weaver's spot, detailing the Morrow user group, or the support BBS. Both still alive and well. Guess I might have to supply from way down here if one of you, being closer; don't do so very soon.

Best wishes for '94, Many Thanks, Paul MacDiarmid, Rotorau, New Zealand.

Thanks Paul for your long distance support. Yup, changes in dollar sure can make TCJ costly. I had to raise the rates on all readers to make us break even. Yet if I do too large of a issue I still can loose money on the mailing costs overseas.

As to the Morrow user group, haven't heard from them in long time, so if you talk to any members, get them to send me the latest information.

So far all the 'C' follow up has been our regular writers expressing their views and comments. May start a beginning section on 'C' and start requesting more comments on where to go next. So far seems like most readers just overly saturated with the topic and are letting it simmer for a while before starting on it again. Time will tell. Thanks again for hanging in there, Paul! Bill.

Dear Bill:

I program and use 'antique 8 bit computers' for many things. They are an Apple][e and a Franklin Ace 1200. In my opinion, they are far superior to IBM/MSDOS machines. MSDOS systems are just CP/M systems that work with CP/M 86, a modified CPM 80 system for 16 bit machines, and NOT a gung ho advanced system as almost everyone believes. When I want to, I can degrade to a Z80 system simply by rebooting my computers. I run Apple Pascal, DOS, ProDOS and CPM2.2/CPAM 4.0 on these.

In *TCJ* issue #64, Jim Moore wrote you a letter I would like to comment on. First, I have always agreed 100% with him about new versions of programs, wasted space (on any CPM system), and so forth. This, as well as being harder to program, is why I will never degrade myself by 'upgrading' to a new system. (I call it downgrading.)

Second, on the subject of CP/M software. The Apple CP/M format is the same as the TRS 80 (Radio Shack) format, which is an 8 bit format called CPM 80. The most common version of this is 2.20 and higher. I have found some of this software in 1) The Public Domain Exchange, 2) a Colorado user club - Aces High, 3) a local software liquidator (California's largest) and now at 4) Lambda Software (advertised in your magazine).

Yours Truly, Daniel E. Wallace, Mountain View, CA.

Thanks Daniel for those words, and do you have the address for Aces High, would like to put them in the support group section if they are still active.

Yes upgrading on IBM clones can be a nightmare, without any guarantee of speed or utility improvements. It still amazes me how often people think that CP/M never really existed, or better yet that DOS will be the first multi-user system, whoops, what happen to MP/M and CP/NET, guess just figments of our imagination.

Oh well, some people never learn, Thanks. Bill.

Dear Bill:

Here's a column from May *Computer Shopper*. Seems like this is something you should know about; I sent your address to Stan Veit.

Sincerely, Steve Brown, WSB Enterprises, Houston TX.

Thanks for the copy Steve, I let my Shopper subscription go many years ago when it became a PC sales only magazine. The article you indicated is Stan Veit's TECH SECTION where he talked about "Endangered Software" and mainly how people are forgetting that other systems are used by many people like us. I think your interest was the fact that TCJ was not mentioned by Stan. Well, I am a little surprised too, since he and I talked *last year about him buying TCJ. We didn't do the deal, because his objective was to turn TCJ into a "beginners PC user" magazine and thus killing off the only magazine supporting these older systems which he says no one is supporting.*

Maybe Stan is having second thoughts about these older systems, since I have not closed down TCJ as he thought I would if I didn't go mostly PC based articles. Proably, the problem really is his contract with Ziff Davis and their editors unwillingness to let him really say what he wants, which I hope would be more about us, but then Elliam Assoc, which he did mention, sends a flyer telling about us with each of his orders. To add more fuel to this mess, I have started advertising in the classified section of the shopper, where we will see if anybody reads that part anymore.

Again, thanks and let me know if you find anymore areas I need to know about. Bill Kibler.

Dear Bill,

Enclosed please find a check for $44.00 to cover the next 12 issues (two years) of *The Computer Journal*. I first subscribed to *TCJ* at the urging of Chuck Stafford, CKG (Chief Kaypro Guru) and HPHCW (High Priest amoung Highly Certified Wizards); it was one of the many pieces of sagely advise I got from him.

I think providing support for the older PC/XT platforms is a good move. These systems, like their SS-50 and S-100 bus predescessors and compatriots, have, in turn, been abondoned by their manufacturers. A great many people still depend on these machines and their relative abundance and low-cost makes them great "hobbyist" systems.

The Computer Journal is a great collection of material. It reflects the care and varied interests of each of the Contributing Editors. Having served as Bulletin/Newsletter editor for various archaeological organizations, I know how much work is required to produce something as large as *TCJ* as a one-person show. Ignore the occasional Letter to the Editor complaining about typos, punctuation, and the choice of colors for the covers. These comments, although I'm sure well meant, miss the mark. It's the information and ideas that are important, not the occasional typos or misplaced quotation mark.

Cheers, Richard Estabrook, Tampa, FL.

Thanks for those words of support Richard. You are correct on the work load around here and hopefully Chuck will be able to help me out soon. Running TCJ really is a full time job, yet the money is not there to do that. So I try as hard as I can, and sometimes must plain old stop and send the issue to the printer. I do that knowing it is the only way to get six issues a year done and still handle all the address changes, new subs, and back issue orders. The paper work side easily consumes 60% of my time. If our readers renewed on time and I didn't have to send out so many notices and flyers to get them to renew, I would proably be able to catch a few extra mistakes and typos.

On Chuck being a CKG; well he tries hard to help our Kaypro users out. Like me, Chuck does other things and can't always be the perfect source of information and services, but at least he is trying. Yes, the PC/XT has been abandoned and interestingly enough their cost are now less than CP/M systems. At the last swap I saw complete XT's going for $5 and $10 each (monitors extra). So for collectors of old systems, stay with anything but XT's right now, CP/M systems are worth more.

And thanks for the letter and renewal, Richard. Bill.

Dear Editor:

This letter is a follow up to my call on your 800 number. I built a data link that I operate between a Color Computer 3 and a PC. Evidently just about every PC made can operate COM1 at high speed, which the commercial system "laplink" uses to transfer files between PC's. When set to a 2 Mhz clock rate the CoCo-3 will reliably communicate at 57.6Kbaud. The next fastest PC rate is 115.2 Kbaud, which is just too much for the CoCo-3 bit banger serial port. I put the link software in an unused CoCo cartridge and made a serial cable, making a very tidy package compared to the usual CoCo setup.

NOTE: Since the CoCo serial port does not use standard RS232 connector you must make an adapter if you use a PC serial cable.

The main limitation is that 2 stop bits are needed, slowing transfer to a little over 5,200 bytes/second. By wiring the CoCo serial data input and interrupt pins together I have the ability to do serial port interrupt servicing. Even though you need one character to initiate an interrupt it represents a delay of less than 200 microseconds. The CoCo-3 starts at low speed when power is applied, so the cartridge software relocates itself and sets things up to full RAM and high clock speed. The routines take about 2 Kbytes, but the benefit is that downloading software from the PC is very fast.

I like machine language programming on th CoCO. The 6809 CPU is easy to use and the computer has no cycle stealing. What you see is what you get. I wrote a small monitor that fits up against the I/O routines at the end of memory and modified the Motorola freeware PC to 6809 cross compiler to turn out a true binary file.

There is still a lot of room for improvments, but I achieved my goal, which was to prove to myself that there is no real barrier to using an obsolete computer like the CoCo with the more modern PC. With a fast data link it is almost possible to forget that you are using another computer with the PC acting as an intelligent terminal. The trick is in eliminating the pshychological barrier presented by long download times.

Yours: Frank Wilson, Tomales, CA.

Your serial interface is an excellent example of what many of us here at TCJ have been saying about PC's and Older systems. You showed how learning as-

sembly and using it on an older system really isn't very hard to do. You also provided a good use for PC's, mainly as file servers, a good compromise of products.

I am not as sharp on CoCo's as I would like to be, but I got the impression they use a standard serial port device and weren't bit banging I/O lines, am I wrong? Maybe you could consider a fuller article with your assembly code for the CoCo and what are you running on the PC? I know you must be using something of your own, since most standard PC based serial programs limit speeds on the slower XT to 9600 MAX (althought they can do higher as you said.) Thanks for the food for thought and good luck with your next assembly project. Bill.

Hello Bill;

I look forward to receiving *TCJ* to see if there are any new 8051 articles. While not the most powerful controller around, its ease of use rate high with me and I've been tinkering around with it. One of the big hassles is EPROM programing. My code is never right the first or second times anyway and it kept me from begining many rpojects. I put together the accompanying circuit to bypass this annoying effort.

I used 0.1" perfboard and soldered wire-wrap wire (it goes faster than you think). It is mounted and wired to an existing card in the PC. The cable is 3 ft. (24 conductors) of the grey flat stuff (an old hard drive cable). On the 8051 Single Board Controller end of the cable, I soldered the wires to the top of a 28 pin socket. It plugs in place of the EPROM. I brought the 5 volts through the cable so I don't need a power supply for the SBC. A reset button and a 5 volt (470ufd/16V) line filter cap were added too. The IC's are 74LS off those junk boards. If you need less than 256 bytes of RAM-EPROM, you can eliminate all the A8, A9 and A10 pins on the RAM (I'm using a 6116 RAM) to + or gnd (or a switch and use different blocks of RAM).

To use the RAM-EPROM run DEBUG, Load a program, like FLASH.BIN, Move it to D800:00 (M 0100 xxxx D800:00 where xxxx is the prog length) and push the SBC reset button. A snap !! the E command lets you change bytes instantly. Other Debug commands, like the Function keys, save you keystrokes, and permit you to save changed programs to disk. You have to Move the program back to the Debug load location in order to save it.

I look forward to playing with the instruction set without it being a BIG deal, and have many projects in mind. I really believe others would enjoy this also.

By the way, did I miss Tim's "data acquisition system" as mentioned in #47? Could you tell me where to find it?

Have fun, Ken Willoughby, Larkspur, CA.

Thanks for the circuit and project, Ken. Well I can also state how great it is not to burn EPROMs. I built one for my S-100 system based on a Microcomputer Journal article some years back. You can do the same things using CP/M as with DOS and I am sure with any other of the small systems. I even have an EPROM burner that has the option to act as an EPROM emulator. Sure makes trying out little ideas a snap!

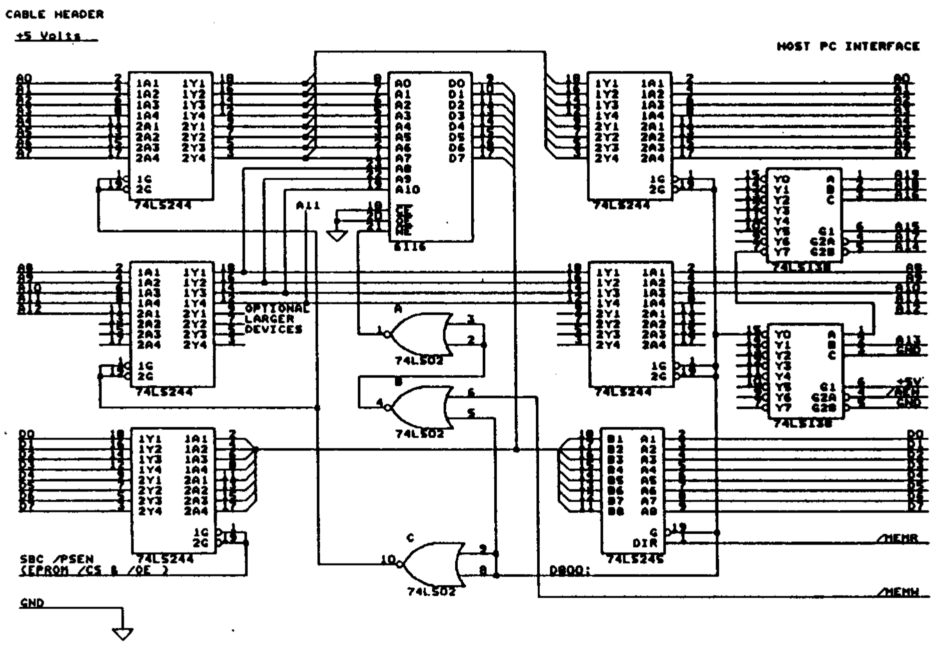
As to finding Tim's article, I think it got lost in my taking over and Tim getting tons of work dumped on him. Tim promissed to start suppling articles just as soon as his new house gets done being built (think he is the contractor on this project). If his house is going as well as some friends did, we might see an article this Christmas (building your own home never seems to end). I have been trying to get others to cover the 8051 and 6805 field, but all too busy to do any writing.

So thanks again and what is your 8051 project? Bill.

Dear Bill,

Several items to cover in this letter. First, and most important!, here's my renewal.

Next, I noticed your request(s) for stuff for the Microlog CPM card. There were at least two models made, which I will call I and II. The I card has only 64K of ram on it. The II card has room for 384K and a battery backed clock/calendar. I have the manuals for both, and software for the II. The II software has all separate programs/drivers for the extra features that the I card lacks (clock, PC ramdisk, etc.), so I would guess that the actual CPM part should work on both. Whichever model you have, I'd be glad to copy what I have for you. HOWEVER, you may not want to waste your time. I use Z80MU, a shareware type of emulator, occasionally. I also have ZPEM, an



emulator (which specifically emulates the Kaypro, Heath H89, and the Osborne). ZPEM came with MEDIA MASTER, a poor imitation of Uniform. It (Media Master) barely gets the job done — I wish I'd bought Uniform instead. I haven't tried ZPEM, but authors are extremely impressed with it. There are others that look promising. One, ZSIM, actually loads (boots) off of real CPM disk if you give it one.

To get back to the Baby Blue card. In the docs of one of my emulators there is a scathing criticism of " some systems actual require that you bind a header to the CPM programs before you can run them...". They are referring to the Microlog card. In order to run a program you must "bind" a header to it. Actually the header is a shell that handles the emulation. It turns your CPM COM file into an EXE file. As soon as the program is finished, you're back in MSDOS — you never see CPM! Strictly speaking, it's more of a converter than an emulator. When I read that in the manual, I just put it back on the shelf (got it at the microscopic local ham fest). Like I said, you're welcome to whatever. But Z80MU is quite good. It even has REZ-like disasm built in (same commands as Ward C's Resource). If you run it on a, say 33 Mhz AT, you get respectable performance. My 12 Mhz At gives a Z80 speed of 1.8 Mhz.

A possibly viable alternative is for someone to write/adapt a real time CPM BIOS for the card. It doesn't look hard, since it uses dual ported RAM for communications between the PC and the Z80. Considering the very heavily documented CBIOS's such as the Xerox 820 I & II, it should actually be quite easy for some whiz out there. THEN up the speed of the Z80 chip, and you'd have a real screamer.

Next item. Some years ago, I hooked up a Western Digital 1002-05 up to a Commodore VIC-20, then a C64, then a C128. I wrote a low level formatter in basic and formatted a 10 meg hard drive. The interface is very simple, one chip, since the 1002 series hard disk controllers are meant to be hooked directly to a bus (they use tri-states). All I did was use a few inches of ribbon cable, an old game catridge for the card edge (game cartridge still works), and a single inverter chip. Seeing as how 1) my Kaypro 10 uses the same controller (1002-HD0), 2) since subscribing to Genie I've noticed a resurgence of interest in CP/M on the C128, I've in mind writing an article on the above (C64/C128-1002-05). What do you think? I'd been holding off writing you but the article on the 6526 in the latest issue kind of broke the ice.

I've been using a C64 CPM catridge for years, and with a very small mod to the cartridge and a minor mod to the software I'll describe below, people with the CPM catridge can be up an running with up to four hard drives of any size. It sounds weird to front end a C64, but the CPM catridge runs a 1 Mhz Z80, just as fast as many older CPM systems still in use now. (Apple Z80 cards are still in use.)

Some years back I purchased a kit from Emerald Microware, called the Winchester Connection (Wincon). I don't know if they still sell it (I'll check). The kit allows you to hook up a hard drive to almost any CPM system (CPM 2.XX only). It works quite well and I've sold several of my friends on it in the past (naturally they made me do the installation). But I've never seen an article on it. There was a review on it, then an article on a home brew S-100 look alike (no drive though...) in *Micro Cornucopia*, but never an article on how to install the Emerald Microware kit. This kit is a real life saver to old CPM systems! If you're interested I'd like to write one, because even if it isn't made any more, the kit includes a schematic, and since I bought both bare and stuffed boards — quite a simple interface — something might be worked out on a source (vis a vis the copyright thing). Also it looks possible to load the modified Emerald Microware driver as CPM Plus (3.XX) RSX. I saw a piece of software on Genie to do just that — load a CPM 2.XX program as a CPM Plus Resident System Extension, thereby allowing users of CPM Plus (such as the C128) to take advantage of this jewel. A special note though, users of anything other than a Z80 will need a modified driver (re 8080, 64180, 8085, etc.) I've done some of the work already, having rezzed the Wincon low level format utility.

Some other possible — shorter — subjects are: using the spare parallel port on the Xerox 820-II, using the SASI host adapter on the same, for other than SASI. By the way, the Xerox 16/8 (an upgraded 820-II) also uses the 1002 series controller, as does the Panasonic Senior Partner and the Seiko 8650 multiuser computer.

Thanks for putting out my favorite magazine (since Micro Cornucopia) — thanks for all the hard work.

Douglas Ross.

I have had several conversations with Chuck Stafford on this very topic. He has been looking for ways to replace failed Kaypro interface cards. Our series on IDE drives was also to get people to consider repairs using them instead of the harder to find 1002 cards. I guess what I want to know (and our readers too) is how did the kit work? What software changes were needed? But please check on their availabiltiy and rights to redo their product if they aren't willing.

Your suggestion is just what I had in mind for changing the Baby Blue software. I saw the header stuff and said what a waste. Your right too about emulators, and I am sure by now you have tried MYZ80 which works very well. Jay Sage is planning on writing a review of Z80 emulators when he can get the time free to do so, but it appears your experiences might be nice to see in writing as well.

Any case Douglas, thanks for the comments, and how about those articles now that you got my interest (and attention)? Bill Kibler.

From: K.OWEN2
To: B.KIBLER
Sub: The possible article...

A friend at work brought me a computer, a Chameleon by Seequa of An-
Continued on page 47

Small System Support

By Ronald W. Anderson

Regular Feature

68xx/68xxx Support

6809 Assembly & Flex

I thought I would start this time with some honest to goodness 6809 topics. First let me give you the name and address of a staunch 6800 and 6809 user that I have been trying to help with an old 6809 system that refused to work.

John Fiorino
518 - 85th Street
Brooklyn, NY 11209

John tells me that he wrote to all of the advertisers from old copies of *'68' Micro Journal*, and received only a few replies, mostly saying the companies don't support FLEX anymore. John would be interested in corresponding with anyone who has similar systems and wants to discuss them.

John found and repaired a problem in his serial port board, but the system wouldn't run, though the serial board runs in another system now. He sent me his MP09A processor board several days ago. I spent an hour after work (where I have a 6809 system) trying to find a problem. After noting that there were not any chip select pulses getting to the Monitor ROM, I found two or three feed-throughs (vias in the parlance of PC board manufacturers) that didn't conduct. I soldered wires through them and filled all the others with solder making sure it flowed on both sides of the board. Still no operation, though all the address and data lines seemed active and there were now chip selects getting to the ROM with the monitor program.

Still no monitor prompt on the terminal. I spent a lunch hour swapping chips from a working identical board. When I finished swapping every chip between the two boards (one at a time with a test of the working board each time) the working board still worked with all the chips from the non-working one and the non-working one still did not work with all the chips from the working one. I ran out of time but brought the board home for a good visual inspection to look for foil breaks and bridges. Continuity testing has not found any more open connections, but then all it takes is one! A little while later I decided to get out an old toothbrush and the isopropyl alcohol (92% Isopropyl rubbing alcohol from the local drugstore) and clean the rosin off of the board (the solder flux). I found a solder bridge between two pads. Unfortunately the board still didn't work. Later I noticed that the molex connectors had been damaged a bit. The contacts didn't spring enough to close the openings in the connectors, and I thought perhaps they didn't make good enough contact with the molex pins on the motherboard. I used up a roll of Solder Wick carefully removing the connectors and replacing them with some unused ones that I had. Still "nada". Ay caramba! Porque no trabaja?

Of course in the process of testing the boards I turned my system on and off dozens of times and then discovered that I had blown my system disk that was left in a drive with the door closed. I wonder frequently why I am so dumb! I didn't think of it because I wasn't trying to get farther than the SBUG-E monitor prompt. No matter. I restored it to working order from a backup. The disk needed to be cleaned up anyway. I sure wish I could find the bad connection though. By the way, though the PC family won't trash a disk left in a drive if you turn the power off and back on again, that procedure is a definite no-no with the old SWTPc systems. Somehow, on power up, the drive can write a byte or two in the middle of the directory or system information sector and the disk is basically scrap until it is reformatted.

Assembler - 6809

I've been putting this off for months, but here I am and the subject this time is going to be assembler programming on the 6809 running the FLEX or SK*DOS operating system. Early versions of SK*DOS for the 6809 were called STAR DOS and you might run across an old disk or manual that references it that way. Peter Stark had to change the name because someone else had used it previously. Obviously the name came from STARk, so the duplication was perfectly innocent.

Flex was supplied with a user manual and an "Advanced Programmer's Guide". It is the latter that will concern us here since writing assembler programs requires interfacing to the operating system unless you want to write your own code to deal directly with a serial or parallel interface. In the present case, after thinking long and hard about how to go about presenting the whole instruction set of the 6809 and all the addressing modes, I decided that would not be the best way to go. Instead, I am going to start writing some simple programs that do something useful.

As we move along, we'll look at the way you interface with FLEX and at how various addressing modes work, but this way we can learn a little at a time and it won't all be boring theory for 8 months before we get around to doing something useful. Besides, I think from my own experience, I learned more from looking at assembler code written by someone else and first trying to understand it, second making small modifications to see if I could make them work, and finally getting brave enough to try writing a program for myself from scratch.

As I write this, I am using a new old toy. I have been doing some consulting for a customer that I have been associated with for so long, that I think of him more as a friend than as a customer. In late April, bill was in Detroit for a conference, and I reminded him that Ann Arbor is only a 45 minute ride from Detroit, so he came for a visit and we had a nice evening out for dinner. At any rate, Bill has had me working on getting his old 6809 programs and a very large set of data files (over 100 Megabytes) moved over from old 6809 8" floppies to his PC on 3.5" floppies, doing some translating on the way.

Since that project is about done, Bill has some 6809 hardware that he is winding down and won't need anymore. He promised to send me one of his old 6809 systems for, as he put it, "my museum". He has done that, and I added a pair of 5" drives and connected my trusty old Radio Shack TRS-80 Data Terminal (that is exactly what the label says). This terminal was bought at a R. S. clearance sale for $50, and it has the best and sharpest green monitor I have ever seen on a serial terminal. It is running 9600 baud (won't quite run reliably at 19,200).

With this neat and reliable old computer setup, I am all set to do the series on assembler programming. I have the equipment at work to transfer disk files to my PC so I can get them into form for the column. The 6809 disk is readable on the Peripheral Technology PT68K-4 system at work. That has a utility called MSWRITE to write it to a disk formatted on the PC, and then I can read it in as a text file and include it in my column. Before I transfer it to the PC disk, I run it through a filter program that adds a linefeed after each CR, since FLEX terminates a line with a CR only and MS-DOS likes CR/LF.

First thing I did was to try to set up a system disk and add a few little utilities. I don't like to go away for supper or a cup of coffee or whatever and leave the terminal's screen full of clutter, so I tried the CLS (CLear Screen) utility on my system disk. It was written for a different terminal, a Wyse that uses ANSI terminal commands, so of course CLS printed garbage on my screen and didn't clear it. This old TRS uses a simple ^Z to clear the screen. That is, the ASCII character decimal 26 or Hexadecimal 1A. Well, why not write a new CLS utility for the system since I am going to use it this way for the foreseeable future?

FLEX has a couple dozen "system calls" that are very useful particularly when writing system utility programs. If you have one of those Advanced Programmer's guides, you can follow along. If not, just keep this series of columns available, since we'll describe every system call that is used. To clear the screen on my terminal, the computer has to respond to my CLS command by loading and running a little program that outputs a ^Z to my terminal and then returning to FLEX.

We will need two FLEX calls. The first is called PUTCHR, and it simply outputs the contents of the A accumulator of the processor to the serial port to which the terminal is connected. PUTCHR is "called" by using an absolute JSR (Jump to SubRoutine) instruction to the subroutine in FLEX after loading the proper code into the A accumulator. We return to FLEX by doing an absolute JMP to the FLEX Warm start address called WARMS by FLEX. I use the word "absolute" here because there are also "relative" jump instructions on the 6809. Those are BRA for BRAnch, LBRA for Long BRAnch, BSR for Branch to SubRoutine, and LBSR for Long Branch to SubRoutine. If you are running SK*DOS you will find that these routines have different but very similar names.

If you have the book you will see that WARMS is defined as $CD03. In 6809 assembler, a dollar sign $ is the prefix for hexadecimal code. The FLEX calls all are up in this area of memory which is of course occupied by FLEX when it is loaded. The address for PUTCHR is $CD18.

First a comment line:

```
* Program to clear the screen of a TRS DT-1 terminal (^Z)
```

Comment lines start with a star (asterisk) in the very left most column of the screen. Everything after the star and on the same line is a comment and is ignored by the assembler. Now we define the FLEX call addresses:

```
WARMS EQU $CD03
PUTCHR EQU $CD18
```

FLEX comes complete with a file called FLEXEQU.LIB which contains the equates for all of the system calls, but the library file is kind of big and we only need these two, so let's do our own. In this assembler all "labels" or names start in the first column of the screen. We are defining names to be replaced by hexadecimal values with these two lines. The EQU is an assembler "directive" that tells the assembler to assign the hexadecimal value to the name or Label preceding it. Statements that equate a numeric value to a label are frequently called "equates". Having defined these, when the assembler sees the word WARMS, (meaningful to a programmer) it will substitute the value $CD03 (meaningful to the computer).

Next we need an origin statement. This tells the assembler where to put the program in memory. All Flex utilities that are small enough load into a "Utility Command area" that is several hundred bytes long, and starts at $C100, extending to (I'm pretty sure) $C7FF.

```
        ORG $C100
```

Note specifically that ORG is not a label or a name, but is an assembler directive. It starts anywhere AFTER the first column. ORG must be followed by an address, usually done in hexadecimal though the assembler could handle the decimal value just as well.

Now we start the code. A program to be run under FLEX needs a label at the point where the code is to start execution when it is loaded by the FLEX binary loader.

```
START LDA #$1A       THIS TEXT IS A COMMENT
```

START is a label. It again must start in the first column. LDA is the operation code meaning LoaD accumulator A. What follows the LDA opcode is the operand. In this case the # means "immediate". The $1A is the ASCII code for ^Z. The immediate sign means essentially "the code immediately following this symbol". An opcode may or may not require an operand. For example COMA is the instruction to complement the contents of the A accumulator. In other words "invert" the contents. Change the 1's to zeros and the zeros to 1's. It doesn't need an operand. LDA, requires an operand. I would read this line as: Load A immediate hex 1A.

```
      JSR PUTCHR
```

Again note that JSR is an opcode and it is not in the first column. PUTCHR is the routine that puts the contents of the accumulator out to the terminal serial port. A subroutine is a section of code that has the operator RTS at the end, meaning ReTurn from Subroutine. When that code is finished, the RTS causes execution to begin at the line after the one containing the JSR. We've cleared the screen. Now we have to return to FLEX so that familiar +++ prompt appears and we are ready to execute another command or program.

```
      JMP WARMS
```

After we output the clear command to the terminal we return to flex with this unconditional jump. You might think we should be done, but not quite. The Assembler requires us to tell it where to start execution of our program.

```
      END START
```

Again note that JMP and END are not in the first column of the line. END START is an assembler directive that causes the program's transfer address to be set to the label START. Seems sort of dumb for this small program, but some programs don't start execution at the first line of the code, and many have a large number of "labels". By the way, the label START here can be whatever. Some programmers like BEGIN. As long as the label that follows the END directive is the label at which you want the program to start execution, everything is fine.

Here is the whole program:

```
* Program to clear the screen of a TRS DT-1 terminal (^Z)

WARMS EQU $CD03
PUTCHR EQU $CD18

      ORG $C100

START LDA #$1A       THIS TEXT IS A COMMENT
      JSR PUTCHR
      JMP WARMS
      END START
```

Blank lines are permitted in a program. If there is any text on the line that is a comment the star must appear in the first column. There are some rules that apply to program listings. Spaces separate "fields" in the program source code. The first field starting in the first column is the label field. Next is the operator field, then the operand field, and finally the comment field. Comments after the operand are valid up to the end of the line.

It is permissible to add more spaces between fields. Some assembler programmers format their source code with extra spaces so it would look something like this:

```
* Program to clear the screen of a TRS DT-1 terminal (^Z)

WARMS  EQU $CD03
PUTCHR  EQU $CD18

      ORG $C100

START LDA #$1A     THIS TEXT IS A COMMENT
      JSR PUTCHR
      JMP WARMS      AND ANOTHER
      END START
```

While this format is a lot easier to read, it takes up more space as a text file, and the assembler will format it's output listing if you want to read that. I've run the assembler on this code and saved the output listing to a file. I've created a label for the CLS instruction also. Here is the result.

```
* program to clear screen for the Tandy DT-100

001A            CLS    EQU $1A

CD18            PUTCHR EQU $CD18
CD03            WARMS  EQU $CD03

C100                   ORG $C100 FLEX UTILITY AREA
C100 861A       START LDA #CLS
C102 BD CD18          JSR PUTCHR
C105 7E CD03          JMP WARMS
                      END START

0 ERROR(S) DETECTED
```

The interesting thing about this output, which can go to the screen or to a printer, is first that it is formatted, and that it lists the hexadecimal machine code instructions that it has generated. The program, first of all, is just eight bytes long. That, of course is because it is little more than a couple of system calls. The first column of this listing shows the memory address at the start of each line of code. We set ORG to be $C100 so the program starts at that address. The code for LDA # is $86. We defined CLS as $1A and that is what appears as the operand. BD is the JSR code and the CD18 is the PUTCHR. 7E is the JMP absolute code and it is followed by the address to jump to, $CD03. The assembler listing doesn't show how the output file looks, but how the program will load into memory. The file contains some housekeeping bytes, the first memory load address, the number of bytes, and the transfer address at the end.

As "homework" if you have the Advanced Programmer's guide you might look through the system calls such as GETCHR and PSTRNG and see what they do. Don't bother getting into the file handling routines, since we won't get that far for a while yet. We'll practice for a while writing programs to do things in memory and report them to the terminal. After that becomes fairly routine we'll write one that opens a disk file. That is scary at first because it is possible to clobber your system disk and the like. We'll talk about precautions to use before testing a first program that handles disk files.

The assembler will assemble this program when given the command: ASMB CLS. Sometimes we want to modify the action of the assembler. To eliminate the listing out to the terminal, eliminate a symbol table output and erase an old output file if it exists, the command would be ASMB CLS +LSY. If you make an error, the assembler will flag it for you. The largest single error that I make is to forget to indent a line that starts with an opcode and not a label.

By the way, if your terminal takes multiple "characters" to clear it, you can repeat the LDA # and the PUTCHR call as many times as you like. For example maybe your terminal needs the sequence ESC * to clear the screen. You can define ESC as $1B and use LDA ESC. You can use a character and the assembler will use the proper ASCII code, by using an apostrophe before the character. LDA #'* will get you the star or asterisk ASCII code in accumulator A. The changed part of the program would look like this:

```
ESC    EQU $1B

START  LDA #ESC
       JSR PUTCHR
       LDA #'*
       JSR PUTCHR
       JMP WARMS
       END START
```

The program has gotten "big". We've gone from eight bytes to twelve if I count correctly. FLEX will use a whole 256 byte sector for this, so even if you have to send your terminal a dozen characters, it won't occupy any more disk space.

Let's talk a little about vocabulary. The code shown just above the last paragraph is called a "Source Listing". That is, it is the file that the programmer generates in words. When it is assembled the result is what we call an "Object file" or "executable file".

I am reminded that I once had a difference of opinion with a reader of one of my early columns. I had said something about liking a high level language for large programs. This reader disagreed and thought I ought to use assembler for everything. Further discussion showed that he thought 100 bytes was a large program! I was thinking of anything over a couple of K of object code. I deal regularly with 32K 6809 object files programmed in PL/9. I would not want to have to maintain these in Assembler, though at this point, I could. Assembler has it's place in the scheme of things, however where speed is an absolute necessity, or where you want something small to do a simple job.

Before I go on to other topics, let's summarize what we've covered in this simple program. I haven't used the term addressing modes, but we've used two.

LDA #$1A Immediate addressing — $1A is placed in ACCA (ACCA is a standard abbreviation for accumulator A) Immediate addressing is used for constants hard coded into the program.

JMP $CD03 Extended addressing. The operand is a 16 bit value, the address to which to jump.

LDA $1234 Extended addressing — the contents of address $1234 are placed in ACCA. This mode is used to access variables. You would probably use a label that was "equated" to $1234.

LABELS are assigned values either by means of "equates" or by their position in the program. In 6809 Assembler a label can represent a 16 bit "word" value such as an address, or it can represent an 8 bit "byte" value as assigned by an equate statement.

CLS EQU $1A — assigns the value $1A to the label CLS

START LDA #CLS — the START label immediately follows the ORG $C100 directive. Labels in the program label field have the value of the program counter at that point. Since we set the program counter to $C100 with the ORG directive, START has that value (see output listing). Labels can be assigned values by one other means that we haven't used yet, the RMB (reserve memory byte(s)) assembler directive.

In case you are wondering, the opcode for LDA # is different from the opcode for LDA extended. The assembler can tell from the operand which is meant, and it generates the appropriate machine code.

Miscellaneous

I've just been writing some drivers for an LCD display onto which we are going to put numbers and text in graphic mode. My first shot was on a PC in C, and it is acceptably fast. I am presently working on the 6809 version in PL/9. My first try at a translation from C to PL/9 was a bit slow in execution. It took about four seconds to fill the screen with text. After some optimization of the PL/9 code it took just under 1 second. I may well have to write some of the procedures as "asmprocs" in PL/9 to try to increase the screen writing speed. PL/9 nicely allows embedded assembler code and we have written a utility called ASMGEN that converts an assembler output listing (like the one above with the opcodes) to the format required by PL/9 for an assembler procedure.

One thing that becomes obvious is that I am going to run out of program space in the 6809 system. One thought I had today was to try to compress the bitmap character images in my font table. Of course then I am slowing the system down as a penalty for saving memory. I wrote a quick program to read a bitmap for an 8 times normal size number 5. The bitmap is 280 bytes. I thought I would count strings of 0's and 1's starting with zeros and alternating, outputting the count to an output file. As long as there are fewer reversals than bytes of bitmap, I would come out ahead. The 280 byte bitmap for the 5 was decreased to 137 bytes, just a bit better than a 50% compression. The compression procedure is simple, and the inverse expansion procedure would be about the same. I would expand the bitmap and then write it to the display.

Another thought was an algorithm to magnify a smaller font bitmap by a factor of 2, making each pixel in the original four in the larger map and then apply an algorithm of some sort to smooth the outline by turning off pixels on outside corners etc. I decided that the smoothing procedures would probably take up more space than I would save, and again would severely slow down the process. A few tests with bitmaps filled in on square ruled paper convinced me that the rules would be complex and that there would have to be a number of exceptions in order to come up with anything nearly as good as a well designed character at the higher resolution.

I finally hit on the simple scheme. The LCD controller has some extra memory. I can store my bitmaps in an EPROM on the controller and read them from there when I write the screen. All my maps occupy slightly more than 8K, and I have 32K available on the controller board. it will be a little slower to read the data but not much. This immediately doubles my available program memory. If I had to, I could put some of the driver routines in the controller memory too. At any rate, I had fun contemplating and testing compression ideas and character outline smoothing algorithms.

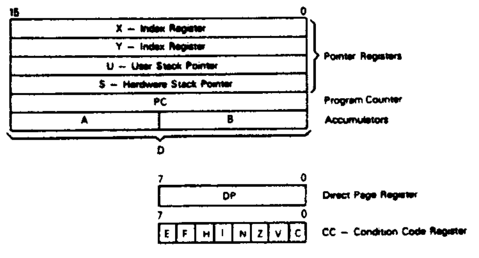
The Numbers Game

Since I have a few K of text space left here, I thought I'd mention something that gripes me a bit. Traditionally, 1K is 2 to the 10th power or 1024 in the field of computing. 1 Meg is 1K times 1K or 1,048,576. RAM and ROM memory is always specified this way. 1 Megabyte of RAM is 1,048,576 bytes. I would therefore expect a 100 Megabyte drive to hold 104,857,600 bytes. I find that all the hard drive manufacturers would call this 105 Megabytes, or at least 104. My new 420 Megabyte drive actually is a 406 Megabyte drive in terms of 1,048,576 byte Megabytes. Of course when you format this drive it reports something on the order of 420,000,000 bytes free before you copy any files onto it. The 170 megabyte drive that I use has an actual capacity of 162 megabytes. I resent the almost 5% inflation of the actual drive capacity calculated the way computer folks have always calculated K and Meg. I would not be unhappy if the drive manufacturers would switch terms and call their 170 megabyte drive a "170 million byte drive", but it ought to be 162 megabytes.

MC6809

From Motorola Microcontroller Manual Vol. II.

FIGURE 4 — PROGRAMMING MODEL OF THE MICROPROCESSING UNIT



PROGRAMMING MODEL

As shown in Figure 4, the MC6809 adds three registers to the set available in the MC6800. The added registers include a direct page register, the user stack pointer, and a second index register.

ACCUMULATORS (A, B, D)

The A and B registers are general purpose accumulators which are used for arithmetic calculations and manipulation of data.

Certain instructions concatenate the A and B registers to form a single 16-bit accumulator. This is referred to as the D register, and is formed with the A register as the most significant byte.

DIRECT PAGE REGISTER (DP)

The direct page register of the MC6809 serves to enhance the direct addressing mode. The content of this register appears at the higher address outputs (A8-A15) during direct addressing instruction execution. This allows the direct mode to be used at any place in memory, under program control. To ensure M6800 compatibility, all bits of this register are cleared during processor reset.

INDEX REGISTERS (X, Y)

The index registers are used in indexed mode of addressing. The 16-bit address in this register takes part in the calculation of effective addresses. This address may be used to point to data directly or may be modified by an optional constant or register offset. During some indexed modes, the contents of the index register are incremented or decremented to point to the next item of tabular type data. All four pointer registers (X, Y, U, S) may be used as index registers.

STACK POINTER (U,S)

The hardware stack pointer (S) is used automatically by the processor during subroutine calls and interrupts. The stack pointers of the MC6809 point to the top of the stack, in contrast to the MC6800 stack pointer, which pointed to the next free location on the stack. The user stack pointer (U) is controlled exclusively by the programmer. This allows arguments to be passed to and from subroutines with ease. Both stack pointers have the same indexed mode addressing capabilities as the X and Y registers, but also support Push and Pull instructions. This allows the MC6809 to be used efficiently as a stack processor, greatly enhancing its ability to support higher level languages and modular programming.

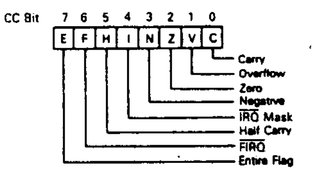
PROGRAM COUNTER

The program counter is used by the processor to point to the address of the next instruction to be executed by the processor. Relative addressing is provided allowing the program counter to be used like an index register in some situations.

CONDITION CODE REGISTER

The condition code register defines the state of the processor at any given time. See Figure 5.

FIGURE 5 — CONDITION CODE REGISTER FORMAT



CARRY FLAG (C)

Bit 0 is the carry flag, and is usually the carry from the binary ALU. C is also used to represent a 'borrow' from subtract-like instructions (CMP, NEG, SUB, SBC) and is the complement of the carry from the binary ALU.

OVERFLOW FLAG (V)

Bit 1 is the overflow flag, and is set to a one by an operation which causes a signed twos complement arithmetic overflow. This overflow is detected in an operation in which the carry from the MSB in the ALU does not match the carry from the MSB-1.

ZERO FLAG (Z)

Bit 2 is the zero flag, and is set to a one if the result of the previous operation was identically zero.

NEGATIVE FLAG (N)

Bit 3 is the negative flag, which contains exactly the value of the MSB of the result of the preceding operation. Thus, a negative twos-complement result will leave N set to a one.

$\overline{\text{IRQ}}$ MASK (I)

Bit 4 is the $\overline{\text{IRQ}}$ mask bit. The processor will not recognize interrupts from the $\overline{\text{IRQ}}$ line if this bit is set to a one. $\overline{\text{NMI}}$, $\overline{\text{FIRQ}}$, $\overline{\text{IRQ}}$, $\overline{\text{RESET}}$, and SWI all set I to a one. SWI2 and SWI3 do not affect I.

HALF CARRY (H)

Bit 5 is the half-carry bit, and is used to indicate a carry from bit 3 in the ALU as a result of an 8-bit addition only (ADC or ADD). This bit is used by the DAA instruction to perform a BCD decimal add adjust operation. The state of this flag is undefined in all subtract-like instructions.

$\overline{\text{FIRQ}}$ MASK (F)

Bit 6 is the $\overline{\text{FIRQ}}$ mask bit. The processor will not recognize interrupts from the $\overline{\text{FIRQ}}$ line if this bit is a one. $\overline{\text{NMI}}$, $\overline{\text{FIRQ}}$, SWI, and $\overline{\text{RESET}}$ all set F to a one. $\overline{\text{IRQ}}$, SWI2, and SWI3 do not affect F.

ENTIRE FLAG (E)

Bit 7 is the entire flag, and when set to a one indicates that the complete machine state (all the registers) was stacked, as opposed to the subset state (PC and CC). The E bit of the stacked CC is used on a return from interrupt (RTI) to determine the extent of the unstacking. Therefore, the current E left in the condition code register represents past action.

Regular Feature

Intermediate Users

Stepper Motors

PC/XT Corner

by Frank Sergeant

Oh, it was a joy using Forth to fool with a stepper motor. The project came together quickly as I tested a little piece at a time until I figured out how it worked.

Mental Model of a Stepper Motor

I start with a fairly simple mental model of a stepper motor. As I see it, it is basically a collection of electromagnets that pull the motor shaft to a slightly new position. The idea is to energize only one or a few of the electromagnets at a time, and in the right sequence, rather like a donkey chasing a carrot and being beaten with a whip, to move it around and around. The motor has just two electromagnets — really two sets, where all in a given set are energized the same way at the same time. But, to keep the explanation simpler, I may talk as if there were just the two electromagnets, rather than the two sets. With each electromagnet, you can either not energize it, or energize it so the North pole is on the "left" and the South pole is on the "right", or energize it so the South pole is on the "left" and the North pole is on the "right." To energize it, you pass current through it in one direction or in the other direction.

How Many Wires

Stepper motors come in several flavors. Really, you effectively have only two coils of wire in the motor, one for each (set of) electromagnet(s). Thus, you can get by with 4 wires. Each coil has two ends and there are two coils. With this arrangement you need external circuitry to allow you to switch the flow of current through a given coil from one direction to the other direction. An "H-bridge" is commonly used for this.

Much more convenient is to have a motor with either 6 or 8 wires. Although we still really have only the two (sets of) electromagnets, we now have two coil for each electromagnet. This does not give us 4 sets of electromagnets. In a given set, you either use one winding or the other winding, but never both at the same time. The whole point of this is to avoid the need for the H-bridge. With four windings, each with two ends, you have 8 wires. The six-winding motor still has 8 wires internally, but two ends of each electromagnet windings are joined together for you. So, either 6 or 8-wire motors are equally easy to use. You could also have a 5-wire motor, where either the "ground" or the "positive" end of all four windings are connected together inside the motor. The motor I used has 8 wires. I connected one end of each winding together to form the common positive end, and selectively grounded one of each of the other 4 wires to energize a particular winding.

My Motor

My motor says ASTROSYN STEPPER, AST P/N 23LM-K005-P4, 7.0 V/PHASE, NO. T32865, Minebea Co., Ltd., Made in Thailand. I don't remember where I got it — possibly from Tanner Electronics. I got several surplus, probably several years ago for about $5 each or so. With an ohm meter I found out which wires belonged to the same winding. Obviously, with 8 wires there were 4 sets, one for each of the 4 windings. However, I could not figure out how to tell which pairs of windings were for the same set of electromagnets. My plan was to ground one end of each of the 4 windings and connect the other end of each winding, one at a time, to +5 volts, or so. I used a resistor to limit the current while I was fooling with it. My worry was that if I picked the wrong ends of a pair of wires for one electromagnet to ground, that whichever of its two free ends that I connected to 5 volts, would result in the same polarity. I still haven't resolved this worry. Any suggestions? Anyway, the motor was advertised as having 200 steps per revolution, and when I give it 200 steps it goes around exactly once, so I believe I must have it connected correctly.

The Motor Connections

So, I've got 4 free ends and 4 ends connected to 5 volts. How do I turn on one winding at a time? For experimenting you can just touch the lucky winding to ground, but for real use you want a rather more convenient method. I choose the DS2003 "high current/voltage Darlington driver" chip from National Semiconductor. I think this is the same chip as the Sprague ULN 2003. It comes in a 16-pin DIP. It has 7 Darlington transistor pairs, which act as 7 switches, with a reverse-biased diode across each switch. There are 7 control inputs, one for each switch, which accept TTL or CMOS logic levels. A high closes the switch and a low opens the switch. The reverse-biased diode prevents the kick from the coil when the switch is turned off from damaging the transistors. In case I don't get around to drawing the schematic, pins 1 through 7 are the control inputs controlling corresponding "outputs" on pins 16 through 10. Pin 8 is ground. The "outputs" output a ground, so to speak, when their corresponding control lines are activated. That is, if pin 1 is at a logic low, then pin 16 is not connected to ground, but if pin 1 is at a logic high, then pin 16 is connected to ground. So, the four windings receive 5 volts on one end all the time, but we select one winding at a time to be connected to ground and thus energize the winding. I used the first 4 switches to control the 4 windings. This gives me 3 switches left over, for other purposes, but not quite enough for another stepper motor.

Sequence is Everything

The trick is to energize the windings in the proper order to tease the shaft around and around. Let's call the 4 windings A, B, C, D. At first I thought there were 6 possible sequences

ABCDABCD...,
ABDCABDC...,
ACBDACBD...,
ACDBACDB...,
ADBCADBC...,
ADCBADCB...

Since I didn't know which was the correct sequence, I figured I would try each of them and pick the best. Later, I realized there were only 3 possible sequences with the other 3 just being the reverse of the first 3. So, one of the three would make the motor go in one direction. Reversing that sequence would make the motor go in the other direction. I didn't care which way we started, so I just had 3 sequences to test.

Parallel Port

I decided to let 4 of the PC's parallel port output lines control the 4 switches. Ordinarily the parallel port is connected to a printer. It communicates the data byte to the printer over the 8 data lines. A separate strobe line tells the printer when the data is valid. The parallel port also has other handshaking lines and ground lines. Ordinarily a "standard printer cable" is connected to the PC's parallel port. This has a male DB25 connector on the end that connects to the PC and a male Centronics 36-pin connector on the end that connects to the printer. I decided not to use that type of cable because of the difficulty of getting a matching female 36-pin connector for the stepper motor circuit. Instead, I used a straight-through 25-pin male to 25-pin female "serial" cable. PC serial ports have either a 9-pin male or 25-pin male connector, and so take a cable with a female end.

PC parallel ports have a 25-pin female connector, and so take a cable with a male end. Don't connect your cable backwards and thus to a PC serial port! Anyway, using the serial cable, pin 2 through 9 are the data bits 0 through 7. We'll just use the 1st four (pins 2, 3, 4, & 5). Pin 1 is the strobe, which we won't need. Pins 18 through 25 are grounds, but who knows if all of them go through the cable (check your cable). Pick one of them for your ground. I think I picked #21. So, those 5 lines are all we need. The "serial" cable ought to be fairly cheap ($3 or $4). Connect pin 2 of cable to pin 1 of the DS2003 as follows

	"serial" cable pin	DS2003 pin
Data line 0	2	1
Data line 1	3	2
Data line 2	4	3
Data line 3	5	4
ground	21 etc	8

then connect a 5 volt supply with the ground to DS2003 pin 8 and the 5 volt end to the common ends of the 4 stepper motor windings. Connect DS2003 pins 16, 15, 14, & 13 to the four loose ends of the stepper motor windings. You could put a current-limiting resistor in series between the power supply and the common end of the motor windings, especially if you use a higher voltage supply. Your motor may vary, so it is safer to measure the resistance of your motor windings and figure out what amount of current-limiting you might want to provide.

Software

I used Pygmy Forth version 1.4 for the following. The general ideas should work with any Forth for the PC. Pygmy can be downloaded at no charge from various ftp sites, including oak.oakland.edu, or from various bulletin boards, or I sell a bonus disk for $15 (in the U.S.) which includes the latest version plus some extras.

First you need to know how to address the PC's parallel port. You commonly refer to the parallel ports as LPT1, LPT2, etc. The I/O addresses that correspond to each of these are stored in a BIOS data area at the 8 bytes starting at address $0040:0008. LPT1 is commonly either $0378 or $03BC. Following is a Forth word to print the addresses of the 4 possible parallel ports.

```
: .PORTS ( -)
        BASE @  HEX
        $40 8  4 FOR 2DUP L@ U.  2 + NEXT 2DROP
        BASE !  ;
```

BASE @ merely fetches the current value of BASE so we can change to base 16 with HEX. Then $40 8 puts the segment and offset address for the start of the BIOS data area onto the stack. The 4 FOR ... NEXT marks a loop that is done 4 times. Inside the loop, the segment:offset pair is duplicated with 2DUP, L@ is a long fetch that reads the 16-bit value stored at the segment:offset address. U. prints this value as an unsigned number. 2 + increments the segment:offset address on the stack to point to the next location. Finally, when the loop ends, 2DROP throws away the segment:offset address that we no longer need and BASE ! restores the previously saved value of BASE. You can type this word to find what the actual parallel port addresses are in your machine. As I mentioned, LPT1 will probably be $0378 or $03BC.

Actually, you don't need the word .PORTS. It is just for your general information. You don't need to know what the port address is, you just need to know where to find it. In the following, we will define a word LPT which looks up the port address for you and stores it into a variable named 'PAR.

```
VARIABLE 'PAR  ( holds base I/O address for parallel port)
```

'PAR is a variable which will hold the I/O address of the port we decide to use.

```
: LPT ( # -) 1- 2* 8 + $40 SWAP L@ 'PAR ! ;
```

LPT is a word that finds the address of the port you want to use and puts it into the variable 'PAR. For example, if you say 1 LPT the word LPT will look up the address for LPT1: and store it in 'PAR.

Next, we need a way to write a value to the parallel port to set the lower four data lines which are connected to the DS2003 chip.

```
: LPT! ( c -) 'PAR @ PC! ;
```

LPT! does the trick. Given a byte value on the stack, that value is written to the I/O port whose address is stored in the variable 'PAR. PC! is the word that does the actual writing to the port. The P stands for port, the C indicates the value to be written is "character" sized (i.e. a byte), and the ! stands for "store" and means we are writing _to_ the port, rather than trying to read from it. So, if we wanted to turn on the least significant bit of the parallel port, the bit that is connected to pin 1 of the DS2003 chip, we could say

```
1 LPT!
```

if we wanted to turn on each of the 4 bits one at a time and then turn all the lines off, we could say

```
1 LPT!
2 LPT!
4 LPT!
8 LPT!
0 LPT!
```

We could even define a word to turn the motor off with

```
: MOTOR-OFF  0 LPT! ;
```

Now we are nearly ready to do some serious playing. We will build a table holding the sequence we want to use to turn on the motor windings in the proper order. Wait, we don't know what that sequence is yet! Ok, we will DEFER a word that represents the table we will use, once we figure out what the table should be:

```
DEFER TABLE
```

As I mentioned, there are only 3 possible sequences. Let's build a separate table for each of the three:

```
CREATE  TABLE1  1 C, 8 C, 2 C, 4 C,    ( sequence)
CREATE  TABLE2  1 C, 8 C, 4 C, 2 C,    ( sequence)
CREATE  TABLE3  1 C, 4 C, 8 C, 2 C,    ( sequence)
```

The first table says we will write a 1, then an 8, then a 2, then a 4 to the parallel port in that order. Then, one at a time, we point the word TABLE to one of the three possibilities and try it. For example,

```
' TABLE1  IS  TABLE
```

is how we would set up the word TABLE so that it would really execute the word TABLE1.

Then, we might want to allow a variable delay to determine how fast we write values to the motor. We will declare a variable ON-DELAY to hold the number of milliseconds we want to wait between writes to the port, and we will initialize it to 100 milliseconds.

```
VARIABLE ON-DELAY
100 ( milliseconds) ON-DELAY !
```

The key word we need is one to make the stepper motor take a single step. We will use the word STEP. Since there are 4 different values we will write to the port to energize the motor windings, we will use the numbers 0, 1, 2, 3 as indexes into the TABLE to represent each of the 4 values. Essentially, given an index on the stack between 0 and 3 which represents the _last_ index used. Then we update the index to the one to use for the current step and fetch the corresponding value from TABLE and write it to the port. We will leave the index used this time on the stack so the next time we take a step that information will be available. Finally, we need to delay for a while.

There is a slight complication. We might want the motor to turn in one direction or in the other direction. We will let the variable DIRECTION hold the value to add to the index we last used in order to give the proper index to use for the step we are about to take. The word FWD sets direction to the value 1. This seems to make since, right? If the last index used was zero, and we add the value in DIRECTION to it, we get the value 1, then next time when we add DIRECTION to it we get 2. This works fine until the last index is a 3 and we add DIRECTION to it and get a 4. Utoh. What we really want to do is make the index cycle through the numbers 0,1,2,3,0,1,2,3,0,1,2,3,... So we want to divide the number by 4 and use the remainder. We use the cheap trick of 3 AND to accomplish this. Thus 0 3 AND gives 0, 1 3 AND gives 1, 2 3 AND gives 2, 3 3 AND gives 3, and 4 3 AND gives 0, which is just what we want!

There is one more slight complication. What if we want to back up, to move through the table in the other direction. Well, we might start subtracting 1 from the previous index, but another way of doing it is to add 3. Once we do the 3 AND, we will find the index backing up just like we want.

```
VARIABLE DIRECTION
: FWD ( last - last') 1 DIRECTION !  ;        FWD
: BACK ( last - last') 3 DIRECTION ! ;
: STEP  ( last - this)
        DIRECTION @ +       ( find the next index to use)
        3 AND DUP           ( equivalent to 4 MOD)
        TABLE + C@  LPT!  ( fetch value & write to port)
        ON-DELAY @ MS   ;     ( kill time)
```

Notice how simply and pleasantly the above can be done in Forth. Of course, we might get tired typing the word STEP over and over, so we make a word to repeat it for us. Given a last index number and a count, STEPS is the answer.

```
: STEPS  ( last # - this)
        FOR ?SCROLL  STEP   NEXT  ;
```

Saying 0 30 STEPS will take 30 steps and leave the last index number on the stack, so we could then say 200 STEPS to move a full revolution. Of course, we aren't yet sure which table to use, so trying

```
' TABLE1 IS TABLE   0 200 STEPS    DROP
' TABLE2 IS TABLE   0 200 STEPS    DROP
' TABLE3 IS TABLE   0 200 STEPS    DROP
```

should let us find the one that works best.

We could add a little error detection code to the word STEP as shown below,

```
: STEP  ( last - this)
        DUP 0 4 BETWEEN NOT ABORT" BAD LAST
        STEP"  DIRECTION @ +
        3 AND DUP ST + C@ LPT!  ON-DELAY @ MS   ;
```

but it is hardly needed. The common mistake would be to forget to put the last index on the stack before calling STEP or STEPS. In that case, some trash on the stack would be used instead, but the 3 AND would clip it to between 0 to 4, so not much harm would be done.

We might find we say 20000 STEPS and decide we don't really want to wait that long! So, the word STEPS contains the word ?SCROLL which checks the keyboard and aborts the current word if you press the Esc key. Or, if you press any other key, ?SCROLL stops and waits until you again press any key.

Once you find the right table to use, you can throw away the other two tables, or save them in case you get a different stepper motor. You can change the value placed into ON-DELAY to vary the speed of the motor. Here is an example of how to turn the motor forward a full turn, back a half turn, forward a quarter turn, back and eighth turn and then stop:

```
0       FWD   200 STEPS
        BACK  100 STEPS
        FWD    50 STEPS
        BACK   25 STEPS
        MOTOR-OFF
```

Let's add one more word to set the speed of the motor

```
: SPEED  ( # -)   ON-DELAY !  ;
```

This is great fun for demonstrating the motor. Note that we have built a motor control language with the following commands: SPEED, FWD, BACK, STEP, STEPS, MOTOR-OFF. Hell, even someone who didn't like Forth might enjoy using such a pleasant and interactive stepper motor control language.

Eventually, though, you might want to encode the complex motions into words of their own, to cut down on your typing. The above demonstration could be put in a word named DEMO as follows:

```
: DEMO  ( -)
     0  FWD   200 STEPS
        BACK  100 STEPS
         FWD    50 STEPS
        BACK   25 STEPS
         MOTOR-OFF        ;
```

Remember, STEPS already has the word ?SCROLL built into it, so just pressing a key will stop and restart it. Or, you could define an endless loop (until you press Esc) to repeat the demo faster and faster and then slower and slower, or produce certain motions based on certain key presses or input switches, etc.

Going Forth

You may have guessed that I have plans for this stepper motor controller circuit in the XYZ table I hope to have one day. So far we've only talked about controlling a single motor. The DS2003 chip only has 3 left over switches; not enough for another motor, but two DS2003 chips would have 14 switches which would be enough for 3 motors with two switches left over. The next question is whether we can get twelve output lines from a single parallel port or whether we would either need to use 2 ports or some sort of multiplexing scheme. Well, we have 8 data lines; enough to control two motors. Then we have 4 output handshake lines; enough for the 3rd motor. (One motor for each of X, Y, & Z requires three motors.) So, it looks like a single parallel port will be sufficient.

Even if a stepper motor is not what you want to control, I hope the above ideas will help you use the parallel port for whatever you might want to control.

Left Over

I still have many things to do and to write about. I will be delighted to hear your questions and suggestions. I received some information from DynaArt about their iron-on methods for making printed circuit boards. I had been disappointed in my first attempts with this sort of thing, but with their suggestions I might find it works well after all. Eventually, I hope to pass on the suggestions and comments about my results. I also have some of the Press-N-Peel material I want to try. I have heard glowing reports about it, but also that it may be inconsistent from batch to batch. I would like to try it as well and give you a comparison of the two. I feel that I am so clumsy with this sort of thing, that if any of it ever works well for me then it will work great for you.

I'm also interested in software digital logic breadboarding, especially for the intro to computer architecture labs I teach. The idea being that a student might learn more of the principles if some of the tedious mechanics of wiring up the circuits were eliminated. I have two packages to look at as soon as I get the time.

Prices I mentioned for old PCs in previous articles seemed way too high as soon as they were in print. I just got a catalog from Sun Remarketing, Inc. (1-800-821-3221) which specializes in used Macintosh computers. They offer an IBM PC with 640K RAM, 2 360K floppy drives, keyboard, serial port, parallel port, and monochrome monitor for $99 (plus $45 shipping & handling!). For $50 more they will replace one of the 360K drives with a 10Mbyte hard drive. I've seen '386SX motherboards for around $64. Some places, I think, don't even _sell_ '386 motherboard any more because they are now outdated. Gee, not only can old PCs join the classic computer ranks, but my '386 seems to be a classic too.

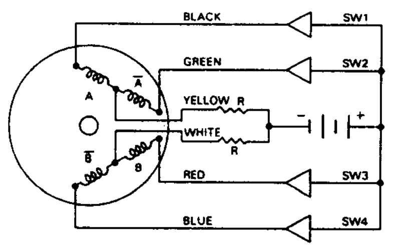
The PLD programmer is still on my list to work on as soon as I can get this conversion project done. I want to discuss the PIC processor, which is getting so much press in connection with The BASIC Stamp, and compare it to the Motorola 68HC11. My first feeling is that the PIC is a step backward and that, dollar for dollar, the 'HC11 is a far better buy and easier to work with. More on this later.

Remember my GEnie email address is F.SERGEANT or, through the internet, f.sergeant@GEnie.geis.com in case I say to hell with school and lose my fs07675@academia.swt.edu account.

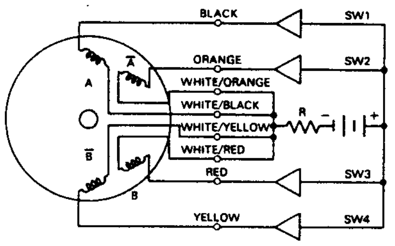
END

Unipolor Drive Circuit

① 6 LEADS



② 8 LEADS



Switching Sequence

FOUR-STEP INPUT SEQUENCE* (Full-step mode)

STEP	SW1	SW2	SW3	SW4
1	ON	OFF	ON	OFF
2	ON	OFF	OFF	ON
3	OFF	ON	OFF	ON
4	OFF	ON	ON	OFF
1	ON	OFF	ON	OFF

*Provides CW rotation as viewed from nameplate end of motor.
To reverse direction of motor rotation energize steps in the following order: 1, 4, 3, 2, 1.

EIGHT-STEP INPUT SEQUENCE* (Half-step mode)

STEP	SW1	SW2	SW3	SW4
1	ON	OFF	ON	OFF
2	ON	OFF	OFF	OFF
3	ON	OFF	OFF	ON
4	OFF	OFF	OFF	ON
5	OFF	ON	OFF	ON
6	OFF	ON	OFF	OFF
7	OFF	ON	ON	OFF
8	OFF	OFF	ON	OFF
1	ON	OFF	ON	OFF

*Provides CW rotation as viewed from nameplate end of motor.
To reverse direction of motor rotation energize steps in the following order: 1, 8, 7, 6, 5, 4, 3, 2, 1.

Courtesy of:

ORIENTAL MOTOR®

ORIENTAL MOTOR U.S.A., CORP.
Head and Los Angeles Office:
2701 Plaza Del Amo, Suite 702
Torrance, CA 90503-7395

SERIES ULN-2000A
HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS

THESE HIGH-VOLTAGE, HIGH-CURRENT Darlington arrays are comprised of seven silicon NPN Darlington pairs on a common monolithic substrate. All units have open-collector outputs and integral diodes for inductive load transient suppression.

Peak inrush currents to 600 mA (Series ULN-2000A and ULN-2020A) or 750 mA (Series ULN-2010A) are permissible, making them ideal for driving tungsten filament lamps.

Series ULN-2001A devices are general purpose arrays that may be used with standard bipolar digital logic using external current limiting, or with most PMOS or CMOS directly. All are pinned with outputs opposite inputs to facilitate printed wiring board layout and are priced to compete directly with discrete transistor alternatives.

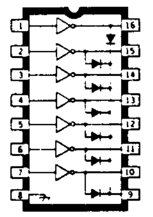
Series ULN-2002A is designed for use with 14 to 25 V PMOS devices. Each input has a Zener diode and resistor in series to limit the input current to a safe value in that application. The Zener diode also gives these devices excellent noise immunity.

Series ULN-2003A has a 2.7 kΩ series base resistor for each Darlington pair, allowing operation directly with TTL or CMOS operating at a supply voltage of 5 V. These devices will handle numerous interface needs — particularly those beyond the capabilities of standard logic buffers.

Series ULN-2004A has a 10.5 kΩ series input resistor that permits operation directly from CMOS or PMOS outputs utilizing supply voltages of 6 to 15 V. The required input current is below that of Series ULN-2003A, while the required input voltage is less than that required by Series ULN-2002A.

Series ULN-2005A is designed for use with standard TTL and Schottky TTL, with which higher output currents are required and loading of the logic output is not a concern. These devices will sink a minimum of 350 mA when driven from a "totem pole" logic output.

Series ULN-2000A is the original high-voltage, high-current Darlington array. The output transistors are capable of sinking 500 mA and will sustain at

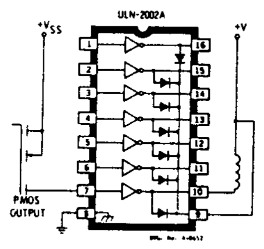


Device Number Designation

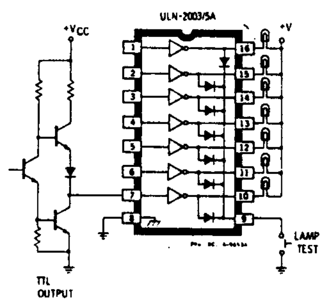
$V_{CE\,MAX}$	50 V	50 V	95 V
$I_{C\,MAX}$	500 mA	600 mA	500 mA
Logic	Type Number		
General Purpose PMOS, CMOS	ULN-2001A	ULN-2011A	ULN-2021A
14-25 V PMOS	ULN-2002A	ULN-2012A	ULN-2022A
5 V TTL, CMOS	ULN-2003A	ULN-2013A	ULN-2023A
6-15 V CMOS, PMOS	ULN-2004A	ULN-2014A	ULN-2024A
High-Output TTL	ULN-2005A	ULN-2015A	ULN-2025A

TYPICAL APPLICATIONS

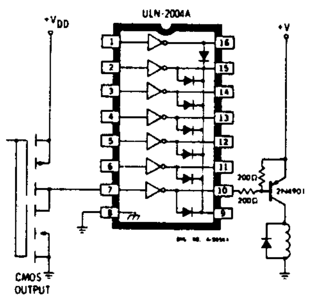
PMOS TO LOAD



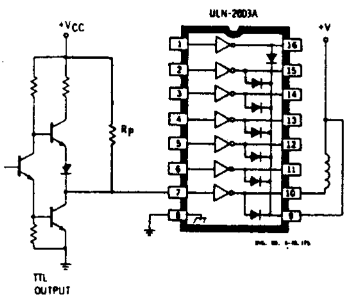
TTL TO LOAD

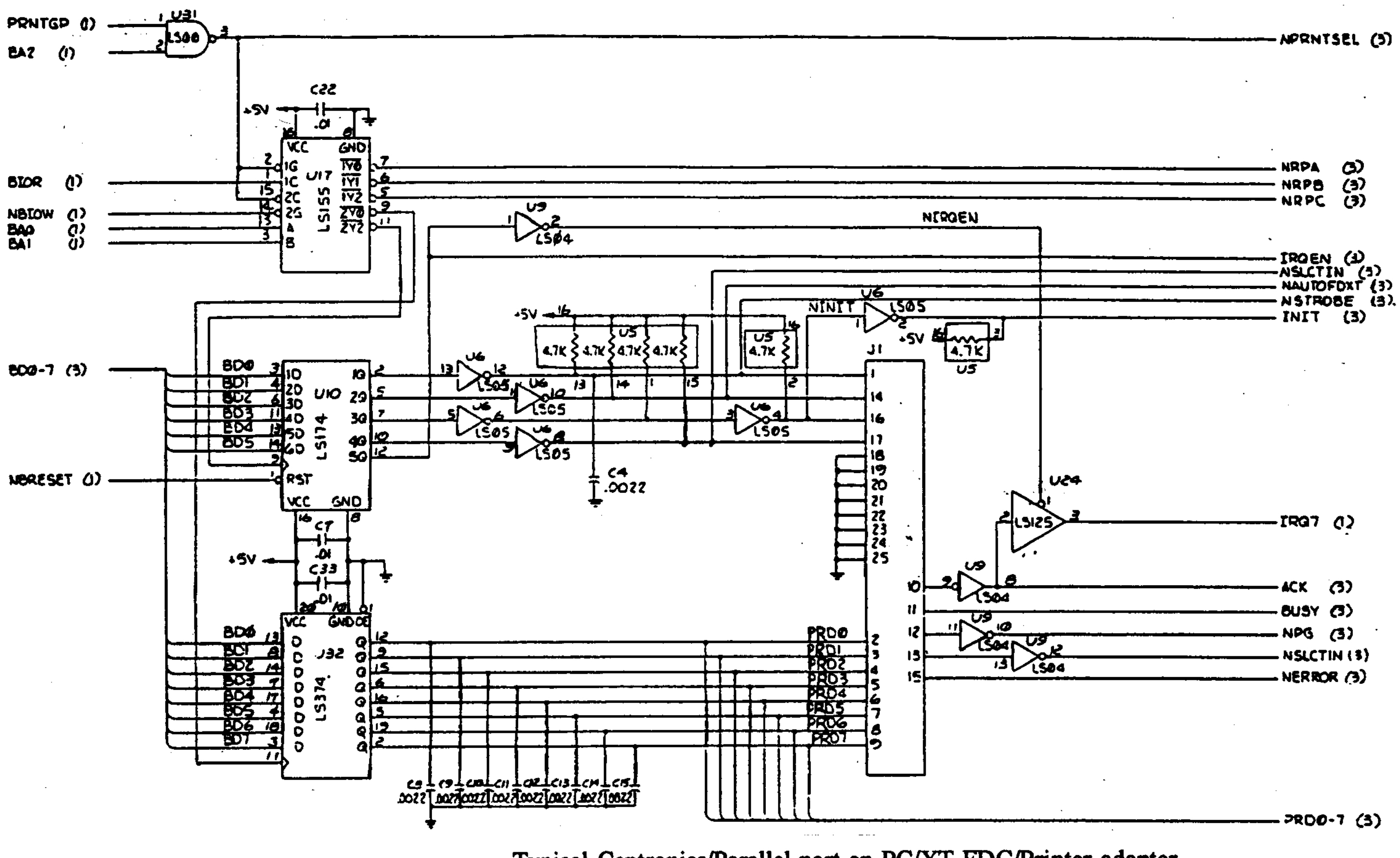


BUFFER FOR HIGH-CURRENT LOAD



USE OF PULL-UP RESISTORS TO INCREASE DRIVE CURRENT





Typical Centronics/Parallel port on PC/XT FDC/Printer adapter.

Regular Feature

ZCPR Support

CP/M part 2

The Z-System Corner II

By Ron Mitchell

There was a programmer of wisdom
Who wrote a small patch for his system
The aim of the code was to make bugs explode,
But his T-CAP fell off and he missed 'em.

(Cue: groaning, laughter and applause)

Nothing worse than suffering from an inadequate T-CAP, or TCAP as it's more properly known, especially when you're trying to write second grade limericks. More about that later. (Not the limericks, the TCAP)

I need to know who you are. There are numerous ways of approaching this series of articles on Z-System. The best one will find its way onto these pages only if I hear your comments and questions. If you are like the various groups of computer users I've had the pleasure of knowing over the years, you have diverse experience and various levels of knowledge. That in itself makes a series like this somewhat difficult to write. If I make it too basic, I'll lose the old hands. If I get too technical (assuming that I could) the new people trying Z-System for the first time will get up and leave. Can't win for losing.

I am on internet through the facilities of the Ottawa Freenet:
ac087@freenet.carleton.ca
Or just plain old Snail Mail:
Apt. 1107,
210 Gloucester St.,
Ottawa Ontario
Canada, K2P 2K4

Finally if you'd like to check out one of the few PBBS 5.0 Bulletin Boards operating on a Coleco ADAM, you can call (613) 230-9511. The "Byteman" operates limited hours between 11pm and 5pm daily, 11pm to 8am Saturday, Sunday and holidays. Parameters are 300/1200/2400 8N1.

One way or another, get in touch. I'm open to suggestion on the content, level of difficulty, and format of these articles. I'd also like to hear some of your experiences with Z-System and how you're using it.

Unfinished Business

There are items of business left from the last installment; we'll pick them up here. We'll begin by completing an overview of the structure of the CP/M operating system and then follow with a description of the role played by the various component parts. With this understanding under our belts, we'll be in a better position to understand in future installments what Z-System does differently. While doing all of this we'll want to cover some general ground rules having to do with syntax and how it is normally expressed.

All of this is going to prepare us to actually load Z-system at the beginning of the next session. Before we do, we have some groundwork to complete that we began in the first installment and that will put us in good shape to really get down to the job at hand in the next installment.

Off to a flying Start

At the outset, there are a couple of assumptions to be made.

1) You know where your CP/M disk is and you've booted it up once or twice.

2) You have purchased Z-System (NZCOM or Z3PLUS) and you're wondering what to do next.

3) You have an open mind and would like to know more about Z-System.

Also, for the record let's get our products straight:

ZCPR2 and **ZCPR3**: Written by Richard Conn 1982 - 1984 (give or take).

NZ-COM: written by Bridger Mitchell and Jay Sage; the docs are dated 1988. This program is the Z-System for CP/M 2.2 machines.

Z3PLUS: written by Bridger Mitchell and Jay Sage. This is Z-System for CP/M 3.0 or CP/M Plus systems.

The history has it that there was another individual deeply involved in the transition between ZCPR and Z-System. That was Joe Wright. More about him in a moment.

Somewhere in the mid eighties, when I had bought my first implementation of CP/M 2.2, I was after all the books on CP/M that I could get my hands on. The manual that came with CP/M for my system had one or two chapters right at the beginning that I could understand. Then it took off into a netherworld of technical explanation which really left me struggling. I came back to it later and understood it, but only with a little help from my friends. There was a period of time when most of the CP/M 'newbies' on my block were reacting much the same way. Those who had the nerve to download one or two of the public domain offerings on local BBS's ran up against the implementation problems of installing something that was originally written for a Kaypro or an Osborne. The terminal most of us used required Zenith/Heath emulation but we didn't even know that, nor the meaning of the word 'patch'.

The early documentation seemed to be written for people who already knew what they were doing, and certainly I was not alone in making this assessment. It took some time spent in reading, re-reading, thinking about, and asking just to arrive at the level of understanding I have now. And that is by no means complete. There are always new things to learn.

Some Help from my Friends

A friend of mine one day, spying on my desk a copy of "The Soul of CP/M" (Mitchell Waite, Robert Lafore - Waite Group - Howard W. Samms, 3rd printing, 1986 - ISBN 0-672-22030X) asked me if I'd installed ZCPR yet. When I said no, he more or less intimated that I would not be a true CP/M user unless and until I did. He added that if I could get ZCPR up and running on my Coleco ADAM I could credit myself with the equivalent of a degree in computer science.

Knowing what I know now, I suspect that he may have been quite right. It seems to me that installation of ZCPR would have required a knowledge of Z80 assembly language even though most of the code files I've since seen are more than well commented. It certainly would have required knowledge of how to use an assembler, linker, loader, and libraries of standard routines. And it would have necessitated full knowledge of the size of one's own system and how to generate replacement versions of CP/M. Then there would have been the joys of adapting the system to different combinations of hardware. It was all possible, but you had to know what you were doing.

Enter Joe Wright. The preface to the Z3PLUS describes it this way:

"Joe Wright brought automatic operation. It was he who first conceived of and demonstrated with Z-COM what many deemed impossible - a version of Z-System that would install itself automatically on almost any CP/M 2.2 computer. Yet, even after Z-COM's success, it still appeared that Z-System could never run on a CP/M-Plus computer with its radically different command processor and banked memory operating system. Z3Plus proves otherwise."

The preface goes on to state the contributions of Bridger Mitchell and Jay Sage in the development of Z-System. Bridger made the systems universal. He developed the ZRL file format and loader which would allow a single file to adapt to any Z-System. Jay Sage added what the preface describes as "dynamics", the ability of the operating system to change its size and characteristics to suit the need at hand, right in the middle of a command line if necessary.

As this series goes on, we will come to appreciate these things more and more.

The Basics

For the moment, let's continue with the basics. In your computer there is memory, probably 65,536 places, pigeon holes, empty boxes, call them what you will. In the last installment we broke down briefly what these memory locations might be used for. You might even have more than 64K. You might have 128K, and perhaps your processor, the Z80, is capable of looking only at 64K at a time, so you're into bank switching. But let's not get too complicated.

Consider the work that has to be done when you sit down in front of your system to make an immensely significant contribution to the world of computing. No matter what you're trying to do, it's quite likely that you will want the following capabilities:

- to output a character to some sort of display device.

- to receive information (characters) from the keyboard or some other type of input device.

- input and output of data from and to a storage media; disks, tapes, hard drives, etc.

The list goes on. But no matter what type of application you're using or what program you might be writing, your job can usually be broken down into some very simple tasks involving the movement, management and processing of data. And the plain fact is, the contents of your computer's memory is one of two things: data or instructions.

The ZCPR3 Manual makes the picture a little more detailed. It talks about 1) Memory; 2) Processors and Processes (a process is a running program); 3) Devices; and 4) Information.

Author Richard Conn talks about memory management, process management and device management. CP/M doesn't do much about the first two. It doesn't allocate memory, and since there is only one processor there is really not much in this realm to manage. Where CP/M excels is in the area of device management. No matter what you've got hooked up, CP/M creates a standard way of looking at the peripherals. The programmer doesn't have to know how the individual device drivers work. All he has to know is that they are all accessed through the standard BIOS or BASIC INPUT/OUTPUT SYSTEM. There is a jump table of instructions located at the beginning of the BIOS. Each entry in the table is 3 bytes long, and no matter what CP/M system is being used, the functions are always in the same order. The programmer needs only to know what the functions do, what information or parameters are needed as input, and what information or parameters are returned by the function when it has done its job. It's sometimes referred to as a 'black box' mode of operation.

Since this is not a course in CP/M we'll stop right there. Essentially the process is to use assembly language to load certain registers in the Z80 processor with the values required by the particular routine being called. You then call the routine and check certain other registers for the data which the routine has provided. Perhaps this particular routine will be called several times so as to get, for example all the characters, one at a time, that a user types before typing a return.

The BIOS is one major portion of CP/M. There is also a 'System Parameter' area, the BDOS and the CCP. The more you rub shoulders with CP/M folk, the more you'll hear these terms being bandied about.

The System Parameter area is often referred to as Page Zero or the first 256 memory locations between 0000H and 0100H. This area is reserved by CP/M for certain vital information about where various elements of the system are located. That's probably oversimplifying it somewhat, but it'll do for the moment. Needless to say we don't write anything into page zero unless we're firmly convinced that we know what we're doing.

The BASIC DISK OPERATING SYSTEM or BDOS looks after information management. It allows you to select a disk, create a file, open a file, close a file, rename a file, delete a file, set the memory address in the computer to which data is written, and read or write a block from a file.

Now when I first read all of this, it seemed to me that there wasn't anything very sexy about any of it. It's very boring stuff, but when you think about it, it's absolutely essential to anything you might want to do with your computer. The better you know the structure of CP/M, the more you'll appreciate the Z-System enhancements which we'll start talking about in a moment or two (honest).

The BDOS provides 37 standard functions. Any good book on CP/M lists them, and describes the parameters they need and the registers they use for input and output. I can spend more time with this in future articles if there proves to be a need. Most beginner's courses on assembly language programming that use the CP/M operating system start you out by having you write a Z-80 program that simply puts 1 character on the screen. Not much you say, but crawling is best mastered before walking and running.

One of the concepts I had a great deal of difficulty with at the beginning was that most of these functions do their thing either 1 byte at a time or some very small group of bytes at a time. It was difficult to see how anything of substance was going to be achieved with such small steps. What you have to remember is that the Z-80 processor is performing these small steps at a very fast pace. You won't have time to eat your lunch while you're watching. So it dawned on me that each job that a computer does is made up of very deliberate single steps. Go get an underline character 80 times and presto, you have a line across the screen. Repeat that process 4 or 5 times and you have a thicker line.

Moving along, we've talked briefly about Page Zero and the BIOS and the BDOS. I would really like to recommend that you find and read a good introductory text on CP/M in conjunction with this series. I've got two here that are constant companions: "The Soul of CP/M" referred to earlier, and another called "Mastering CP/M" by Alan R. Miller (Sybex 1983 ISBN 0-89588-068-7)

You'll find that when you've got CP/M booted, the BIOS and BDOS don't jump right up and introduce themselves right away. And Page Zero doesn't really put up a sign saying, "Please Stay Out!" In fact all you get is:

```
A>
```

Perhaps if you're a Canadian we should rewrite that:

```
EH>
```

And there you and your computer sit, staring at one another. There are two things required for anything further to happen. Your command and another part of CP/M known as the CONSOLE COMMAND PROCESSOR or CCP. The CCP, as the cliche goes, is where the rubber hits the road. Richard Conn calls it the "human interface."

In fact the CCP under vanilla CP/M is not very smart. It only knows six words. These are the six that we talked about in installment 1:

```
DIR ERA REN USER SAVE TYPE
```

That's it, that's all.

Type something else:

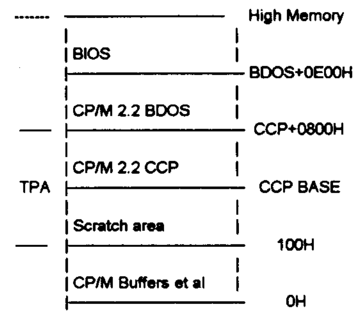
```
EH
```

EH is not one of the six built in commands, so the CCP assumes that you're looking for a program name on the disk and goes to look there for a directory entry that matches what you typed. If it finds one it attempts to run a program by that name. If it does not it comes back to the screen and types your command followed by a question mark.

```
A> EH?
```

Filling up the Memory

So let's recap. The diagram you'll no doubt see in your favorite CP/M text might look like this:



Here's To the Relatives

There are two things to get used to here. The first is the expression of various points in memory in relative terms. In other words, the beginning of the BDOS is 800 Hex bytes above the beginning of the CCP. Why not express it as a fixed memory address? Well, you might have 64K of memory available, or 48K or something else. Addresses may vary from one machine to the next, but the relative positioning of the start of each segment is the same. The area between 0100H and the beginning of the BDOS is known as the Transient Program Area or TPA.

The larger your TPA, the larger the application you can run.

The second thing to get used to is the expression of memory locations in hexadecimal rather than decimal. You can convert the values if you wish, but you'll find that most people don't in this world. Hexadecimal arithmetic more closely matches the 8 bit setup within the computer.

There is another aspect of the basics that caused me difficulty. That was the notation that CP/M writers used to convey generalized forms of specific instructions. This type of notation is not confined to CP/M nor solely to the field of computer science. To illustrate, you might see something like this:

```
PIP destination=source#1,source#2,...,source #n
```

or

```
MLOAD [<outnam=>][ <file1.type>, ] <file2>[,
<file3...>] [bias]
```

or something as simple as:

```
COPY [du:] afn [du:]
```

Again, we're going to use this notation because it's fairly standard among CP/M users, I certainly can't think of anything better. There are the following general rules:

1) AFN and UFN stand respectively for ambiguous filename and unambiguous filename. You have no idea how long it took me to figure this out. It may well be printed in several places at the beginning of each and every CP/M text I own, but the moment I see either one I'm infuriated. Question: why the heck don't they just say what they mean? A filename that contains wild card characters, or a filename that does not contain wild card characters.

2) Information contained within angle braces <> is information that must be typed exactly? Well no, not exactly. As far as I can tell it means something more along the lines of .... fill in the blanks with your own information.

3) Most always, information contained in square brackets [] means that the information specified is optional. That is, you can leave it out without affecting the operation of the command or program.

4) DU: or du: stands for drive/user. So if you see A0:FIDLEFUD.COM that would be a .COM file named FIDLEFUD located on drive A: user area 0.

5) outfil or outfile and infil or infile usually require you to substitute the names of your input and output files in the order specified.

6) and of course if you remember high school math you know that:

file#1,file#2,file#3,......,file#n

means keep adding filenames to fill your boots.... as many as you have need to process, or until you fill up the command line buffer.The value #n is meant to refer to the last one in the list of files to be processed, however many there might be.

You old hands may laugh, but you'd be surprised how easily we of the novice category become confused. Sometimes several strings of this type of notation written and intended in good faith are enough to send us away forever. We might never be back.

Exit, Stage Left

This seems like a good place to stop. During this installment and the last one, we've dealt as much with CP/M 2.2 in its 'vanilla' state as we have with Z-System. From here on in that will change. In the next article we'll begin with a discussion of some of the advantages of Z-System, and we'll re-draw the memory map that appeared here to show you how Z-System adds to the capabilities available while retaining compatibility with CP/M.

We'll also have some information for you on the various sources of assistance available for Z-System users. There's a lot of help out there even yet. And finally, we'll get back to and take care of the problem of the inadequate TCAP.

Do you need
***Micro Cornucopia* Disks?**
Echelon Publications?
Boot Disks?
Disk Copying?

Lambda Software Publishing

can now supply reprints of *Micro Cornucopia* Magazine, Kaypro Disks, Boot disks, CP/M 2.2, ZCPR and CP/M programs.

Kaypro disks	$5.00
all 49 disks	$200.00
Big Board disks	$5.00
all 30 disks	$100.00
Catalog of disks	$5.00
Disk Copying	$10.00
MicroC reprints	$8.00
Z-Letter back issues	$3.00
CP/M 2.2	$25.00
CP/M Plus	$25.00
Spellbinder v5.3H	$60.00
Echelon Publications	$15.00
Four or more	$10.00

User Guides:
ZCPR 3.3, Z-System, ZAS/ZLINK, ZDM/ZDMZ/ZDMH, JetFind, and many other Manuals.

Sound Potentials Public-domain software - catalog $10.00
Any disk in collection $10.00/disk

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

Regular Feature

Intermediate

Letters In The Mail Bag

Dr. S-100

By Herb R. Johnson

"Dr. S-100's Fall column" by Herb Johnson (c) Aug 1994
Internet: **hjohnson@pluto.njcc.com**

Introduction

Given the rising costs of the IDE S-100 project, I've decided to strip it down to just the "chip" and wirewrap. That is, you'll have to hand wire the chip to a S-100 prototype card. Details are contained in a separate article this issue, written by Claude Palm of Palmtech who developed the chip, and myself. I know of at least **one** reader who is interested: check my mail below.

I got some nice deals on a few systems, and I discuss the problems of shipping and purchasing. And as usual I'll share my correspondence. Notably, this includes some further discussion of **Digital Research GSX** by Emmanuel Roche.

Networking

I've joined the other techno-lemmings and got an **Internet** account a few months ago. Rather than pay the connect-time charges of one of the big networks like Compuserve or Genie, I decided to support a new guy in the neighboring town of Lawrence NJ. The New Jersey Computer Connection (NJCC) is like many local BBS's that have sprung up in the last several years, except for two things. It runs under Linux, a "freeware" Unix compatible OS; and it has a connection to Internet instead of one of the many dial-up BBS networks such as Fidonet. For a flat fee each month, I get unlimited access (except when the lines are busy) and a modest amount of disk space. See my "references" section for my 'net address. I'd recommend the news group "comp.os.cpm" for most of our CP/M-based readers.

My biggest challenge, after dealing with the odd syntax of Unix, was to find an "off-line reader" that could accept a downloaded file of mail and news, allow me to read it and respond, and would package my responses for uploading back to the Linux/Unix system. This is the only way to cruise the Information Highway: staying on-line for hours ties up the phone! The reader of choice for my MS-DOS machine (sorry) is **yarn**, which is available as shareware. Check your favorite Unix site or large MS-DOS depository, or ask your sysop.

For those of you who can't afford this or who do not have a network provider in your area code, you can still find a lot of activity on the many BBS-based networks. **FidoNet** has been around for several years, and carries the "CPMTECH" conference area. Most major metropolitan areas have a Fidonet node somewhere. There are other regional, national, and international networks of this kind: ask your local BBS operator for details.

Correspondence

If possible, please include your **network address** or **BBS location** if you have one. This will allow myself and others to respond to your requests. And, if you use an address of any sort from my column, please note the source! And, if you want our correspondence **private**, please tell me so: otherwise I presume you really want to hear from other S-100 owners and welcome a place in my column. I will use some discretion in my quotations and comments.

I may not answer your letter immediately: let me know if it is urgent. It would also help if you include a **phone number**, or even a reply postcard! If you want a system or cards, it would help me a lot if you tell me what you have or want in detail (if you can). And, tell me where you heard about me.

Emmanuel on GSX for CP/M, Amstrad, NCR

Emmanuel Roche of Troyes, France continues to inform me on his progress on interpreting **Digital Research's GSX** graphic standard. In June he sent me a copy of a Byte Magazine article (printed from a disk file) from February 1983 (p. 256) "Realizing Graphics Standards for Microcomputers" by Fred Langhorst (Digital Research) and Thomas Clarkson (GSS). Also, Emmanuel has been working on disassembling the **Amstrad PCW8256 BIOS ROM** and sent some results to me: anyone interested should contact myself or him. He is also working on the **NCR DecisionMate V** and needs info on its I/O port assignments. He asks <with my comments in brackets> "The <NCR> driver uses four I/O ports: 10H, 11H, 0A0H, and 0A1H. From the code, Emmanuel thinks 0A0H and 0A1H give access to the NEC uPD7220 GDC <graphics chip?> and that ports 10H and 11H give information about the hardware Could someone confirm and give the requested technical information. Emmanuel does not have an NCR available: he merely disassembled the NCR driver to see how DRI did it...and maybe he'll port it to the Epson QX-10 which uses the same NEC chip." He enclosed an ad of an Illuminated Technologies S-100 card which also uses this graphics chip. He says "a GSX driver for this board would have boosted the S-100 bus with superior graphics...and probably launched a whole industry. But DRI didn't write it..."

Regarding GSX, he writes: "TO contribute to rolling the ball of GSX a little further, enclosed you will find an article explaining GSX and an example of its use: a <plotter> graph with your name <and a color bar chart!>. GSX is a CP/M 2.2 program, so it will run under Z-system (<although> I am strictly DRI-compatible). The only (and big) problem is the CRT driver: very few were written for CP/M-80, as DRI was switching to the 16-bit wonderland <8086> at that time.

He later reports in July <again with my comments>:"Yippee!!! Success at last! The Epson FX-80 GSX printer driver is finally available! <He has been working on this for some time, based on interpretations of other DR's drivers.> Now that this powerful GSX driver is available, the road is open for more physical devices, like laser printers...

Please report to your readers that "Emmanuel Roche is looking for people still using CP/M, but using a laser printer such as the HP LaserJet 4 or Canon LBP-8 IV, in order to beta test GSX printer drivers. You will be requested to provide some technical infos to write five hardware specific routines. GSX screen drivers are available for the Amstrad PCW8256 <a popular European CP/M Z-80 system>, Epson QX-10, NCR Decision Mate V, and DEC VT-100 terminal <a close cousin of ANSI, I believe>."

"Personally, as told in my last letter, I am working on the NCR CRT driver. The only thing I don't understand so far is the use of I/O ports 10H and 11H. If one of your readers of TCJ could write to tell me the truth or to confirm my guesses, I would be most grateful (and he would get a copy of the source code!).

"I also plan later to work on the Amstrad screen driver, because it is the most successful GSX microcomputer ever, having sold more than 1,250,000 units <!wow!>."

Although Emmanuel tells me "he's not looking for personal publicity" in writing to me, I nonetheless encourage him to put together an introductory article on GSX, once he feels he has a good enough understanding to develop a "complete" description for distribution. I'll send some of this info to **David McGlone** of **Lambda Software**, who has a license agreement with the former Digital Research. I'll refer his Amstrad work to **Elliam Associates** who supports the Amstrad. I'm impressed by the sheer number of machines produced!

News from England

As usual, Emmanuel encloses the **Disc Library News** newsletter from the Windsor (England) Bulletin Board User's Group. He suggests that "their morale is low <and> they need some boost". Indeed, the newsletter says their harddisk crashed and their library of disks has only received a few dozen requests. Their library is up to volume 139; they report 17 volumes of various utilities such as assemblers, text tools, editors, unARCers, and so on. Some of them I recognize as pretty familiar. A catalog disk is available.

The Editor notes that our own **TCJ** reports a drop in Z-systems sales; Jay Sage's discussions of 4DOS over Z-system; and "TCJ itself has decided to allot 10% of its space to DOS" as suggestions that "the writing is on the wall....?".

Fair Trade

A. H. Smith of Alexandria VA thanks me for the **Compupro** information he requested. He's working with a Compupro 8/16 system (later copies as the Heath Z-100 design). He tells me "I have recently horsetraded some PC equipment (the stuff was junk to me) for several late-model Compupro boards, as follows;

Disk 1B <a 5"/8" floppy disk controller>
M-Drive H (512k) <a RAM disk card>
CPU 286 <and 80286 CPU card — yes, on the S-100!>
SPIO <an I/O card, you guessed it>

These have the voltage regulator jumpers for the IEEE 696 "B" standard, where the supply voltages are +5V and +/-12V. If you are interested in trading hardware, I could easily be convinced to trade the a System Support II card for an Interfacer 4 <both I/O cards of various sorts>, trade the '286 for a Disk 1A, or whatever. Since I am not particularly fond of RAM disks, I might also be tempted to part with one or more M-Drive cards.

Have you had any requests for "late model" Compupro equipment? There is a guy in California who retails Compupro/Viasyn <the latter was a brief name change for Compupro> hardware and software at exorbitant prices, such as $400 for a System Support II." Again, thanks very much for your assistance. If you have any other "requirements" for S-100 equipment, including Northstar, please let me know."

"The Dr." (that's me) often enjoys these kinds of trades and mutual assistance. It's likely some else has the board or manual you need, or needs the stuff you have, so I enjoy being the "matchmaker".

The Oldest S-100 system in continuous use?

How long has your system been up? **Ramon F Gandia of Nome, Alaska** tells me:

" The S-100 IDE/floppy adaptor you described sounds great! Let me know where to send the money! <My friend!> My Altair 8800 has been running since 1979, and I switched to CP/M about 5 years ago <!> when the old MITS <the company that built the Altair> drives became flaky. I am using a Fulcrum <the company that "succeeded" IMSAI> Omindisk controller, but I had to do most of the BIOS and all of the boot <code> because Fulcrum's software was written for the Z-80 and wouldn't run on my 8080 MITS cpu card. Thus far I am running on 5" floppies since I haven't located a WD-1002-05 controller card."

<The Dr. believes this is an old SASI card that converts MFM hard drives to

the SASI standard, an older version of the SCSI interface. Anyone who has this card may want to contact Ramon.>

"In any case, an IDE drive would be better. Some IDE drives have as much as a 256K buffer which would make an old Altair really fly! <Imagine the possibilities....!> In my system I have a Memory MErchants 64K RAM card, with the last chip a 2716 <PROM> where I put my boot code, Alan Miller's monitor, console and printer routines (BIOS callable) and a routine to receive on a serial port and to put the data into memory starting at 0100H <CP/M TPA>. On my ibm pc I have an EPROM burner, and over the years I have had a lot of fun with this setup; now on version 57 of the EPROM chip!"

Thanks for such an enthusiastic letter, Ramon! This is indeed a useful combination! You should write it up in detail for TCJ; it would be adaptable to any Z-80/8080 development environment! Ramon also referred me to a source of IMSAI stuff, a guy that is "sitting on a pile" but "he never gets around to it on these little sales". Sigh. Just as for the $400 Compupro board, there are enough industrial users of these systems that three-digit prices (and presumably service) are supported. After all, that WAS their original price!

A long "weight" for a system.

Billy Munroe of Spring Hill, FL "needs an S-100 system to help me with a correspondence course. (It makes lots of references to the S-100 and a paper punch machine.) Could you please write me with a price (or price list) for my **realistic** want. Thank you." If anyone has a system for him for an apparently modest price, contact Billy.

The problem from my end, Billy, is the **shipping costs**. For example, I just got a 60-lb box from California: by UPS ground shipment, it cost about $25. Even by the cheapest rate, U.S. Post Office 4th class book rate, it cost $11 to ship 35 lbs of books! When you consider a "system" of an S-100 chassis with power supply and cards is 25 to 50 lbs or more; books and disks maybe another 25 lbs; and 8-inch disk drives of around 50 lbs; you'll end up spending about $50 more-or-less in shipping costs.

What I can do, other than post the request, is connect people like Billy with people near him who have notified me a system is available. Unfortunately, by the time I'm told a system is "available", it's just before it hits the dumpster! I still encourage people with systems about to be tossed to contact me - I may pay the shipping and more anyway, or I can make some referrals.

Quick notes

I want to acknowledge mail from **Gary Cooke of Washington IL**, who is valiantly struggling with a Morrow S-100 machine and who discusses this on FidoNet; I'll write more on this next issue. **Kent Walters of Ogden UT** says "I offer him a glimmer of hope" in getting info on his new Vector Graphic S-100 "Vector 4" computers (I think I can help!).

References

Emmanuel Roche, 8 rue Herluison, 10000 Troyes, FRANCE.

Windsor BBUG, 11 Haslemere Road, Windsor, Berkshire ENGLAND SL4 5ET.

Art J Smith, 1808 Jamestown Rd., Alexandria VA 22308.

Ramon F Gandia, P. O. Box 970, Nome, Alaska 99762.

Billy Munroe, 20829 Moreland Dr., Spring Hill FL 34610.

Gary Cooke, 200 W Monroe St., Washington IL 61571-1466.

Kent Walters, 3159 Ogden Ave, Ogden UT 84401-3905.

WANTED

TCJ needs your embedded story or project!

Our readers are waiting to hear from you about how you developed that embedded project using 8051 or 6805. Used Forth, "C", BASIC, or assembler any language is fine, just tell us what happened, how you did it,and how it ended up. No project too small!

Send those Ideas to:

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

CLASSIFIED RATES!
$5.00 PER LISTING!

TCJ Classified ads are on a prepaid basis only. The cost is $5.00 per ad entry. 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.

Commercial Advertising Rates:

Size	Once	4+
Full	$120	$90
1/2 Page	$75	$60
1/3 Page	$60	$45
1/4 Page	$50	$40
Market Place	$25	$100/yr

Send your items to:

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

When calling TCJ advertisers, Say you saw it in TCJ!

TCJ Center Fold

Special Feature

All Users

S-100 IDE Project

S-100 IDE Project Progress Report
(c) Herb Johnson 1994. Portions (c) Palmtech 1994

Early this year, I was contacted by Claude Palm of Palmtech of Queensland, Australia. He is a developer of Z-80 and Z-180 (pronounced zed-one-eighty, incidently) systems, and a user of S-100 systems for prototyping. While developing a compact Z-180 system, he came up a cute little single-chip solution for IDE drive interfacing, and offered the chip to me for consideration. We spent some time in correspondence, and I investigated the costs of an S-100 IDE card. I reluctantly decided that the startup costs and small quantity costs of such a product would be too much for my meager budget of time and money, and at $150 or more each as a selling price would also be too "pricy" for many of my S-100 colleagues and readers.

Bill Kibler begs to disagree. He believes that anyone with a soldering iron and an wirewrap tool - and an S-100 fan - would love to get one of these chips. After conferring with Claude, we decided to offer the chip, a circuit, and some software to my Loyal Readers to see where the winds will blow.

This issue, I'll present Claude's early notes, some schematics, and parts costs. Next issue, if I survive the flames of criticism and IF I RECEIVE SUFFICIENT SUPPORTING CORRESPONDENCE, I'll continue with more software details and encouraging news. I *may* get Claude to discuss the process of developing such a chip: he recently suggested "that would fill a book", but I encouraged him to think further. He will at least show us a "state diagram" of the chip's operation.

The following was written by Claude Palm of Palmtech (with a little editing from me.)

Introduction

Inspired by the recent interest in both S100 systems and IDE drives in TCJ, Palmtech has come up with a single chip IDE interface for the S-100 bus: the PT IDE100. The chip comes in a 68-pin PLCC package and performs all buffering, decoding and 8/16 bit conversion required to operate one or two IDE drives on an S-100 bus. There is additional on-chip circuitry to attach a second I/O device, a floppy controller.

A home constructor could use a PT IDE100 chip to make a "hard card", i. e. an S-100 card with on-board hard disk drive. A low profile (LP) IDE drive can be mounted on the S-100 card, so that no external cabling is necessary. LP size IDE drives are quite common, with low power consumption, and modest cost.

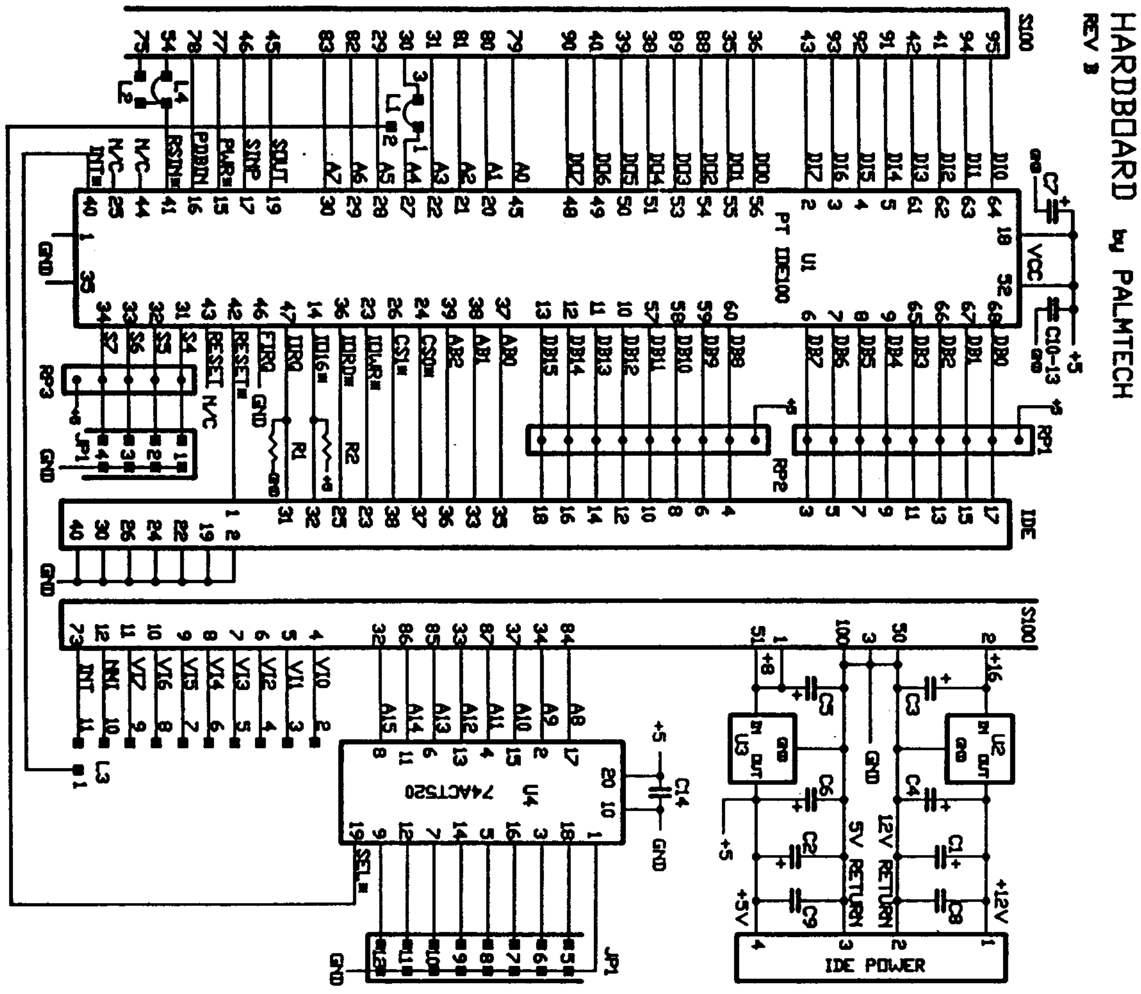
Hardware

The "HARDBOARD" requires few components: the drive, a PT IDE100 chip, regulators, resistor packs, decoupling caps, an address decoder (for 16-bit I/O addresses if needed). Signal (non-power) connections can be wirewrapped or, preferably, point-to-point soldered. Wire wrapping will result in a triple-thick card (with on-board drive and long wirewrap sockets). Wirewrap PLCC and IDE sockets are more expensive than the solderable equivalents.

Power and ground connections must be made with copper strip or hookup wire and MUST be soldered. Good decoupling (with bypass capacitors) is essential with hand wired connections, even with heavy hookup wire. Minimum decoupling is with 2 tantalum caps and 1 ceramic cap at each regulator, 1 tantalum and 4 ceramics at the PT IDE100 chip, and 2 electrolytic caps and 2 ceramics at the IDE power connector. Skimping on these caps or on wiring can generate voltage spikes during operation which will cause unpredictable errors. See the HARDCARD suggested layout. The drive cable is a short length of 40-wire flatcable with IDC plugs at either end.

As there is space to spare on the HARDBOARD the PT IDE100 has two pins (FIRQ and RESET*) to interface with a DP8473 Floppy disk controller chip (FDC) from National Semiconductor. The floppy disk registers map "below" the IDE controller registers in the I/O map. (More information on this extension will be available later. Only the chip, some resistors and caps, and a crystal is required.) The DP8473 comes in a 52 pin PLCC package and supports four drives; the 48-pin DIP version supports 2 drives.

Register addresses will be called "X3" or "X7": the "X" is set by the user to be 0 through F to complete the 8-bit address. If a sixteen bit address is needed, the 74HCT520 and the 8 jumpers will select the upper 8 bits of the address. If an 74HCT520 is not available, use an '521, '688 or an '689 but include 8 2.2k pullup resistors (from the switch or jumper pins to +5 volts).



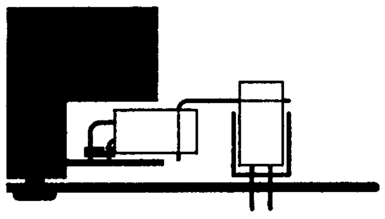
Comp list:

U1 PT IDE100
U2 7812
U3 7805
(U4 74ACT520)
R1 330ohm
R2 10k
RP1-2 2.2kx8
RP3 2.2kx4
C1-2 47uF tant/electro
C3-7 10uF tant
C8-14 0,1uF
JP1 4x2 (11x2 for 16-bit IO)

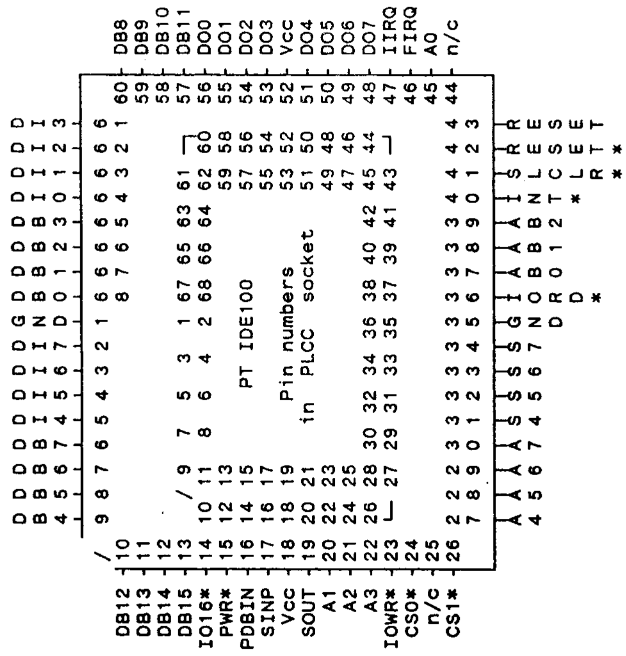
A0-3	READ	WRITE
x0	FDC Status (4)	-----
x1	FDC Read Data (5)	FDC Write Data (5)
x2	-----	FDC Drive Control
x3/7	Disk Changed Bit (7)	FDC Date Rate (7)
x4	Interrupt source	-----
x5	-----	-----
x6	IDE Status 2	IDE Digital output
x7/3	IDE Drive Address	FDC Date Rate
x8	IDE Read Data	IDE Write Date
x9	IDE Error	IDE Features
xA	IDE Sector Count (Read/Write)	
xB	IDE Sector Number (Read/Write)	
xC	IDE LSB Cylinder (Read/Write)	
xD	IDE MSB Cylinder (Read/Write)	
xE	IDE Drive, Head (Read/Write)	
xF	IDE Status 1	IDE Command

x3 and x7 are the same register

Table of I/O registers.

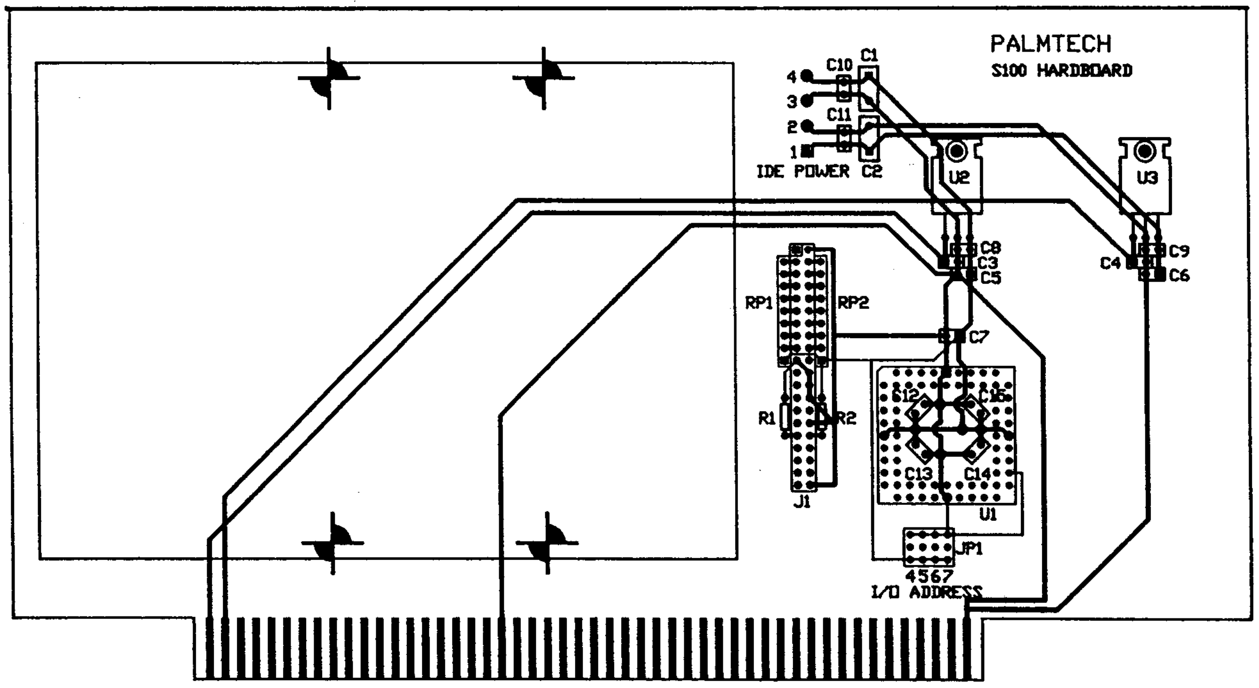


Mechanical detail of the IDE connector


PT IDE100
Pin numbers in PLCC socket
DB12
DB13
DB14
DB15
IO16*
PWR*
PDBIN
SINP
Vcc
SOUT
A1
A2
A3
IOWR*
CS0*
n/c
CS1*
DB8
DB9
DB10
DB11
DO0
DO1
DO2
DO3
Vcc
DO4
DO5
DO6
DO7
IIRQ
FIRQ
A0
n/c
DB4 DB5 DB6 DB7 DI4 DI5 DI6 DI7 GND DB0 DB1 DB2 DB3 DI0 DI1 DI2 DI3
A4 A5 A6 A7 S4 S5 S6 S7 GND IORD* AB0 AB1 AB2 INT* SLCLR* RESET* RESET

DI0-7	O	S100	Data in
DO0-7	I	S100	Data out
DB0-7	IO	IDE,FDC	Data bus LSB
DB8-15	IO	IDE	Data bus MSB
A0-7	I	S100	Address bus
AB0-2	O	IDE,FDC	Address bus
S4-7	I	gnd/vcc	I/O Address space select jumpers
CS0*	O	IDE	Address select x8-xF
CS1*	O	IDE,FDC	Address select x0-x7
SINP	I	S100	Status bus
SOUT	I	S100	Status bus
PDBIN	I	S100	Control bus
PWR*	I	S100	Control bus
IORD*	O	IDE,FDC	Read strobe
IOWR*	O	IDE,FDC	Write strobe
IO16*	I	IDE	Data I/O is 16-bit
IIRQ	I	IDE	Interrupt request
FIRQ	I	FDC	Interrupt request
INT*	O	S100	open collector to INT*, NMI*, or VIO-7
SLCLR*	I	S100	Slave Clear (or RES*, POC*)
RESET*	O	IDE	Drive Reset
RESET	O	FDC	FDC Reset
n/c	O	---	Test, DO NOT CONNECT

* denotes active low, I = Input, O = Output, IO = bidirectional



Suggested parts layout for HARDBOARD.

Square pads denotes: pin 1 in a connector / positive in a polarized cap / common in a resistor SIP
Wide lines denotes copper strip or heavy gauge wire with soldered connections.
Signal lines are not shown.

Parts list:

U1	PT S100IDE		
U2	7805 or LM340T5 3-pin 5V 1A regulator		
U3	7812 or LM340T12 3-pin 12V 1A regulator (or 1.5A)		
RP1-2	8x1k SIP resistor pack, or 8 discrete 1k resistors each		
R1	10k	R2	470ohm
C1-2	22-47uF electrolyte	C3-4	10uF tantalum
C5-7	4.7-6.8uF tantalum	C8-15	0.1uF ceramic monolithic

Programming information

Board address pins S4-S7 set the board's 16 byte I/O space. The register table shows all registers valid with an IDE drive and the optional FDC chip attached. There is some internal address remapping for the FDC that takes place on the IDE100 chip AB2 line. Numbers in brackets are the physical addresses sent to the FDC. Register address X7 and X3 addresses the same register - when read, bit 7 comes from the FDC and reflects the READY or DISK CHANGED line, while bits 0-6 comes from the IDE and reflects the complement of the currently selected drive, head number and write gate. Writing to either address sets the FDC data rate (250/300/500/1000 kbps).

The PT IDE100 has an on-chip interrupt register at X4 so that a single S100 interrupt can service both FDC and IDE.

Data is processed in the same fashion as the old Western Digital floppy controller chips. Writing to address XC through XF clears an internal data register and counter. The IDE drive requires that you write a command to address XF to start data transfer, and that is sufficient for the PT IDE100 chip. There is only one restriction when writing 16-bit data to sector buffer XF: You must write an even number of data bytes. The last odd byte will be lost if not followed by a second "even" byte. But, you can read an odd number of bytes from the buffer, for a partial sector read.

Data is written and read in the order MSB (most significant byte) first, then LSB (least significant byte). For example, reading the second word after an IDENTIFY command for the number of cylinders in the drive, you will read, say, 03, then D4. The correct sequence of these is 03D4 hex, or 980 decimal. However, the IBM PC stores data LSB first, then MSB. If you read a drive written on an IBM PC, the DATA contained in the sectors will be "flipped", byte for byte.

The PT IDE100 does not support the IDE "IOCHRDY" line, a wait signal to stop the CPU. CAM ATA set aside IDE pin 27 for it and declared it optional. Seagate specifies this pin as RESERVED. Some old drives, such as Conners, use pin 21 for IOCHRDY, some use 27, some both, some neither.

Performance and timing

The PT IDE100 adds a 50ns delay from PDBIN valid to data valid. Add to that the drive's access time for total read access. Measured time for a 10MHZ HD64180 (Z180 like processor) and a Seagate ST351 40 MByte drive was 175 ns from address valid at the S-100 bus to data available at the bus. Write delays are similar. Measured from address valid on the bus to data and IOWR* active at the drive was 140ns. As Seagate specifies 40ns data hold time, total write time was 180 ns.

The one timing constraint is that IO16* from the drive must be valid before PDBIN or PWR* is asserted by the S-100 CPU card. Normally there is ample time for this, and any problem is unlikely unless the CPU is very fast and the drive is very slow. If in doubt, add 28 ns to the drive's max address valid to IO16* delay.

Costs (written by Herb Johnson). (You are shortly going to see why the costs of this project discouraged me.)

Cost of the PT IDE100 chip will be $40.00 US, plus $5 for shipping from Australia. This price may shift due to fluctuations in both the Australian dollar and the U.S. dollar. A typical PLCC solder-tail socket is manufactured by Aries Electronics as part 68-535-10: Digi-Key carries this as A419-ND for $2.43 each. You'll need a PLCC extraction tool to remove the PLCC chip: a screwdriver will break the socket. Digikey carries a tool from OK as a "universal PLCC extraction tool", number K293-ND, for $19.75. The 74HCT520 is not in the DigiKey catalog, but the 74HCT688 is for $1.07. Of course, you'll need resistors, caps, IC sockets, etc.

I did not see S-100 prototype cards in the DigiKey catalog. These are S-100 boards that are all drilled out with tenth-inch holes, and a gold-plated edge connector with pads to each pin for soldering. Vector still makes them as part number 8804 (with ground/power busses) or 8801-6 (blank). My Newark catalog shows these priced at $43.21 (8804) and $57.57 (8801-6)! I have several new 8804 cards that I'd sell for $30 each. You may want to shop around: old electronics parts stores probably have these hanging around. Ask DigiKey if they will special-order it.

I had hoped to offer a pre-etched and drilled printed circuit board for this project. However, the one disadvantage of S-100 cards - their large size, originally a plus for 1980-class technology - stood as the biggest obstacle. For a run of several cards, just based on the SIZE of the card, I have been quoted about $45 to $60 each, depending on methods used. And, some of those quotes do NOT include plating the "fingers", or edge connector!

One way or another, you can see this project will be over $100 in parts and tools. Nonetheless, I will act as Palmtech's "distributor" for chips and information. Palmtech is using a version of this chip in one of their products, so they will remain interested in its S-100 use. Claude regularly uses S-100 for prototyping! For our overseas readers, I'll also include Claude's address and phone.

Herbert R Johnson
CN 5256 #105
Princeton NJ 08543
(609) 771-1503
Internet hjohnson@pluto.njcc.com

Claude Palm, Palmtech
cnr Moonah & Willis Sts.
Boulia, QLD. 4829 AUSTRALIA
voice 077-463-109
(no net address)
fax 077-463-198

Mr. Kaypro

By Charles B. Stafford

Regular Feature

Kaypro Support

Advent Decoder Part 2

Personality decoder board Part 2, for All DSDD CP/M Kaypros except K-4x, and Robie

WHEREIN we shall complete the construction project started two issues ago, and install it, thereby completing yet one more step in the transmogrification of Darth Vader's lunch box.

For those among you who have just joined us, (and those who, like me, only have a 64k memory) we started this project two issues back, by etching a printed circuit board using copier toner for photo-resist, and collecting the rest of the necessary parts. We also discussed a second option of using a commercially available prototype circuit board. The schematic diagram from which we are working was also published. We did not however get into the theory of operation. We shall remedy that oversight now.

THEORY of OPERATION

The circuitry on the Personality Decoder board is divided into three distinct functions; The Personality part dealing with telling the BIOS how many and what variety of drives are installed, the actual Decoder part that separates the "drive select" signals and routes them to the proper drive, and a timing circuit that allows a "fast step" option for those drives that can handle it.

PERSONALITY

The 8 position dip switch, 4.7k ohm resister pack, and the 74LS151, IC2, comprise the Personality section. When one of the switch sections is closed it pulls the corresponding pin of IC2 low. Various combinations of pins "pulled low" result in a discrete number being transmitted back to the BIOS via the "read" line from pin 6. When a drive is selected for reading or writing, the TurboBios interprets the number returned on the ready line to determine whether or not the drive is installed, and what type it is.

DRIVE SELECT

The decoding is handled by IC3, the 7445. It takes the inputs from the drive A and drive B lines, combined with "motor on" signal and transmits a signal on pins 5,6,7,or 9 which are routed to the four select lines on floppy drive cable via connector J2. The TurboBios designates four drive select signals on two lines by using the "both on" and "both off" states as well as "A only on" and "B only on" and IC3 figures out what it really means.

NOTE: If this project is intended for a K-10 You should not install the DRV B select line between J 1 pin 12 and IC3. The connection and pull-up resistor between IC2 and IC3 should be installed however. The reason is that the BDOS for the K-10 uses the DRV B select line is used to access the hard-drive.

After installation the floppy drives will the first and third logical drives and the dip switch should reflect this.

A FASTSEEK option is included for those floppy disk drives that are capable of stepping faster than 6ms per track. Use of the FAST SEEK option (FSO) can lead to significant speed improvements and noise reduction when used with some drives. For those who are curious, it's the series capacitor, reducing the time constant that does the trick. THIS OPTION ONLY WORKS WITH THE TURBOROM. It is NOT compatible with any other ROM including the original Kaypro ROM, NOR is it compatible with any software which directly accesses the floppy disk controller, such as Uniform, Media Master, or Fastback. To use the FSO and these programs on the same computer, you must use a single pole double throw toggle switch as described in note 6, Section E.6. page E-7, of the TurboRom Manual.

SIDE SELECT

The schematic also shows a "side select" line for K-II/83 only. This was included in the design for those who had not yet done the II to 4 upgrade yet, or who were doing it in conjunction with the installation of the Personality Decoder board. The K-II/83 mother-board did not include a trace to pin 32 of the floppy connector, and this is a relatively painless way to do it. It can be included or not at your discretion.

ON TO CONSTRUCTION: LAYOUT

For those of you who etched the custom circuit board, layout is easy, just put the proper component in the holes as labeled on the circuit board, and rejoin us a few paragraphs later.

This stage can make assembly very easy or more intricate, depending on your own proclivities. There are several choices, ranging from "let's just get this thing together as painlessly as possible" to "WOW, how did you do that?" Observe Figure 1. This is a representation of the "top" (non-solder) side of the Syntax PC-462905 Uni-Board mentioned in the first half of this project.

Each group of three holes in each row is in one solder pad, so three leads can be connected by putting one in each hole and soldering it on the back side. This will make life a great deal easier (as I learned from an HCW [HIGHLY CERTIFIED WIZARD]). There are also two "busses", each consisting of half the perimeter holes and lines of holes between the three hole pads, also making life a little easier. The holes on the board outside the boundaries of the drawing were omitted, because we don't need them. Now, the choices, the bus on the top and right sides will be the "ground" bus, and the bus on the left and bottom sides will be the "power" bus. For ease of construction, the "output" header, J2, will be placed so that one row of pins is in the ground bus, and the other row is in the first (left-most) column of "three"s. This is convenient because, checking the schematic diagram, all the even-numbered pins must be grounded.

J1 and the "dip" switch will be placed using the same consideration. The other sockets are 4-holes wide, so if we place them over the center line of the three hole columns we lose one hole each pad, but, if we place them so as to straddle the ground and power busses, we won't use up any more holes than necessary, making wiring easier. Figure 2 shows the result as seen from the top. J1 is the final area where choices are important. Perhaps the most elegant solution is to put J1 on the bottom side of the circuit board, so that the finished board will just plug onto the motherboard, and the existing cable will plug onto J2. The difficulty encountered here is "side-soldering" a connector not intended for side-soldering. It can be done, by pulling the pins far enough out of the connector shell so they can be soldered, but so that the plastic shell will still maintain the alignment of the pins, and by using a block under the board to keep the pins from falling all the way into the holes until they are soldered, side-soldering the pins, and then pushing the plastic shell back down on the pins.

A second option is to just put J1 on top of the board, solder it on the bottom and plug the finished product onto the mother board upside down. The disadvantage here is that one must remove the personality-decoder board completely to reconfigure it. The third option and perhaps most reasonable compromise would be to put another header at the J1 location and use a cable to connect it to the mother-board. The caveat here is to make sure that J1 and J2 are clearly labeled, to eliminate confusion when doing the final installation.

Once you've decided on the layout, put the components in place, invert the board (you might want to use a folded towel to put it on, to compensate for component thickness) and solder the pins at the diagonal corners of each component. Soldering only the diagonal pins now will allow for inspection of the result and adjustment if necessary (adjusting one pin is a lot easier than de-soldering all 14 or 16. When you install the header at J2 (also at J1 if that's your choice), if you use a right angle header as I did, just let the pins barely protrude from the bottom of the circuit board, so that there will be enough room to put the plug on when you're finished.

With the position of the dip switch shifted to take advantage of the ground bus, one column of holes is covered up. Installation of the strip resistor on the bottom of the circuit board using these holes (with the "dot" end toward the bottom [power bus end] of the board) will free up the last holes in the pads for wiring, so that there will only be one lead in each hole.

So endeth the layout portion of the project.

WIRING

Before we really get too involved, you might want to make at least two copies of the schematic diagram, and two of Figure 3. We'll use one of each during the wiring phase, and another during the checking phase. Some colored pencils (not all red, but at least three different colors) and a highlighter will also come in handy.

What might at first have seemed to be a real chore, has been greatly simplified by the choices we made during layout. For instance, all the ground connections to J1, and J2, have already been made, as have the connections from the resistors to the dip switch.

So here we go! First things first, the power and ground connections to the three ICs. Following the table on the schematic and remembering that IC pins are numbered counter-clockwise from the notch when viewed from the TOP, make, install and solder jumpers from the power and ground busses to the appropriate pins of the IC sockets. All wiring can be done on the top of the circuit board, but a more elegant solution here is to use bare wire jumpers on the bottom of the board from the appropriate pin to the bus, which is right next door, so to speak. While we are in the neighborhood, a similar jumper from the powerbus to the resistor would be appropriate. See Figure 3.

SOLDERING

Perhaps a short refresher on soldering is appropriate here. (All you HCWs can take a short nap now) For the rest of us, as a general rule, the smaller the soldering iron the better, and Radio Shack varieties work as well as the higher priced spread. Thermostatically controlled, variable temperature irons are very nice for commercial applications, but not necessary for us AWs (Apprentice Wizards). My favorite weapon is a 15 watt refugee from Radio Shack, which has a very fine tip, and is light enough to be very handy. It will probably not last more than 5 years, but I'll be ready for a change by then anyway, and it was very inexpensive.

Eutectic solder (63/37) works the best, but 60/40 is an acceptable substitute. Rosin flux is the only way to go and a small can will last almost forever. Heat up the iron, tin (or re-tin) the tip (dip the tip in the flux, cover it with solder and wipe it off with an old piece of damp towel) and we're ready to press-on.

Hold the tip against the lead or pin protruding from the solder pad, count three and apply the solder. Not much, just enough to fill the hole, remove the iron and proceed to the next one.

The numbering of the pins of J1 depends on whether you used a header, or a socket. Headers are numbered from left to right with odd pins on the top and even on the bottom. Sockets on the other hand are numbered from right to left, again with the odd pins on the top, and the even pins on the bottom.

Starting with pin 10 of J1, run a wire to pin 13 of IC3. Run another wire from the remaining hole at pin 13 of IC3 to pin 11 of IC2. Run a 4.7k ohm resistor from the remaining hole at pin 11 of IC2 to the power bus. See Figure 4. Grab your highlighter, and one of the duplicate schematics and highlight the circuit you just wired. It should be labeled DRV A on the schematic. Now start at pin 12 of J1 and using the same procedure run the DRV B select line, and all the rest of the circuits as shown on the schematic, highlighting each circuit as you put it in.

A SMALL HINT

Before you start running wires, you might want to use one of your duplicate copies of Figure 3 and your colored pencils to decide how to route the wires, and where to put the resistors. The way we ran the DRV A line is not the way that's shown on the schematic, but electrically they're the same. We could have put the resistors from the extra hole at pins 10 & 12 at J1 to the nearest power buss (See Figure 4) and it would still be electrically the same. The reason we can get away with it is that we're working at relatively low frequencies and with relatively short conductor paths.

TO CONTINUE

When you get to the SIDE SEL, READY, VCC, and FAST SEEK lines, Radio Shack sells a set of "Micro-clip test leads" which are very inexpensive and which do the job ideally. They come as a pair, and are about 12 inches long. Cut them in half and you'll have the four leads you need, except that two will be black, and two will be red. Pieces of masking tape or spots of plastic model paint on the clips will handle the labeling.

When the schematic diagram is completely highlighted, you are finished with the wiring phase and it's time for INSPECTION.

INSPECTION

This is our last line of defense against catastrophes. Actually, it isn't all that bad, We're only working with 5 volts, and even inverted polarity won't smoke anything. That isn't always true, but with these components, we're lucky.

Using the second duplicate schematic as a reference, check each conductor's endpoints on the circuit board and as you find them correct, highlight the conductor on the schematic diagram. This may seem redundant, but it's the only way to be sure.

TESTING

The easiest and scariest part, will it, or won't it? Nothing left to do but take the bull by the horns. Install the board as you planned either by plugging it onto the motherboard, or using an auxiliary cable (don't forget to insulate the bottom) and plug the 34 conductor ribbon drive cable onto it. Set the switches appropriately, and now it's time to deal with the clip leads.

The READY line

Find and carefully remove the floppy controller. It will be a 40 pin flat pack IC, designated as U-82 on '83 models, U-44 on '84 models, and U-74 on K-10 '83 models. Carefully bend out pin 32 to about a 45 degree angle and reinsert the floppy controller into its socket matching the notch in the IC with the notch in the socket. Connect the READY microclip to pin 32, the one you just bent out.

VCC

The Personality/Decoder board must be supplied with +5 volt DC power. This is done by connecting the VCC microclip to pin 14 of U-86 on '83 models, pin 14 of U-72 on '84 models, or pin 16 of U-32 on K-10 '83 models.

If you have installed 96tpi drives, you can install the FAST SEEK option. It is a two step process, first connect the FAST SEEK microclip to pin 22 of the floppy controller by removing the IC, bending out the pin (22) and reinserting the IC into its socket (see "The READY Line" above), and second, use TURBOCFG to specify the appropriate step rate. See your TurboRom manual Chapter 4 for particulars.

SIDE SELECT

If this installation is part of a K-II to K-IV upgrade project, AND the K-II has not been modified to use double-sided drives, cut the conductor identified as J3 on the schematic and connect the SIDE SELECT microclip to pin 13 of U-72.

If Your Computer is already using double-sided drives, remove this microclip lead to prevent inadvertent shorts.

NOW

Power up the computer, the power LED will light, the disk drive should run, and the Drive A LED should light.

WHAT TO DO IF IT DOESN'T

First turn off the power and disconnect the power cord, then do a physical inspection, making sure that the right ICs are in the proper sockets, and none of the pins got folded under. Second, reinspect the wiring, using the schematic, paying close attention to any solder that might have sneaked over onto a forbidden solder pad creating an unplanned short-circuit. Third, unplug the ribbon cable to the drives and check that the select line for FD1 is high (5.0 V) initially but goes low (0) when the computer tries to boot. If it does, but the computer still won't boot, recheck the dip switch to make sure that sw2 is set for the drive you have as DRV A. The next suspect is the ribbon cable itself.

As a last resort you can reach me weekends and evenings at (916)483-0312, or CompuServe 73664,2470, or Internet 73664.2470@cis. GOOD LUCK AND HAPPY SOLDERING :)

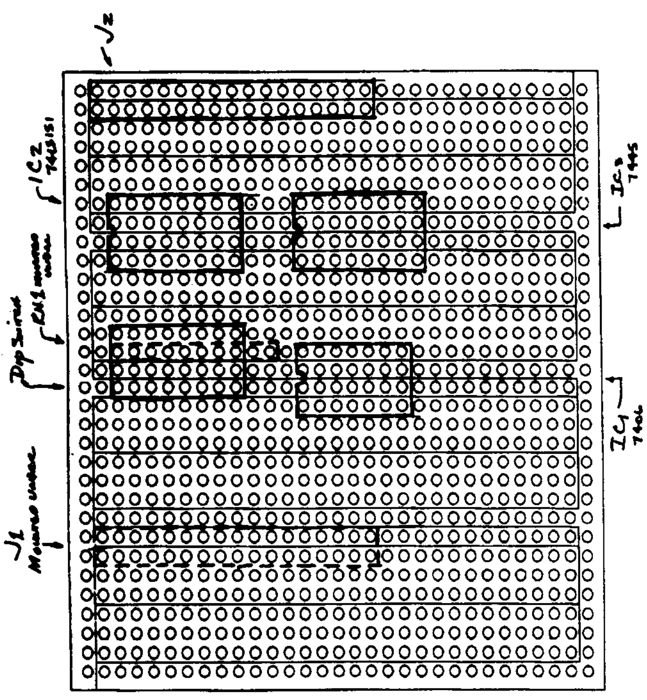


Figure 2

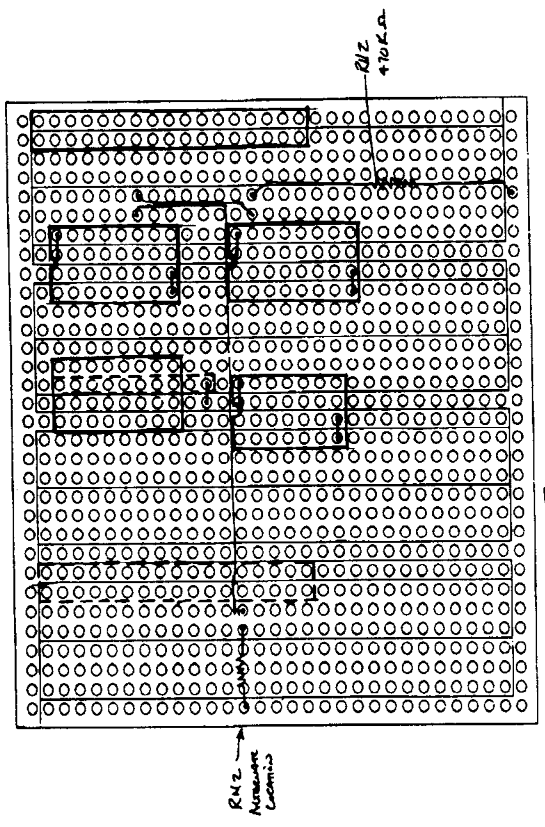


Figure 4

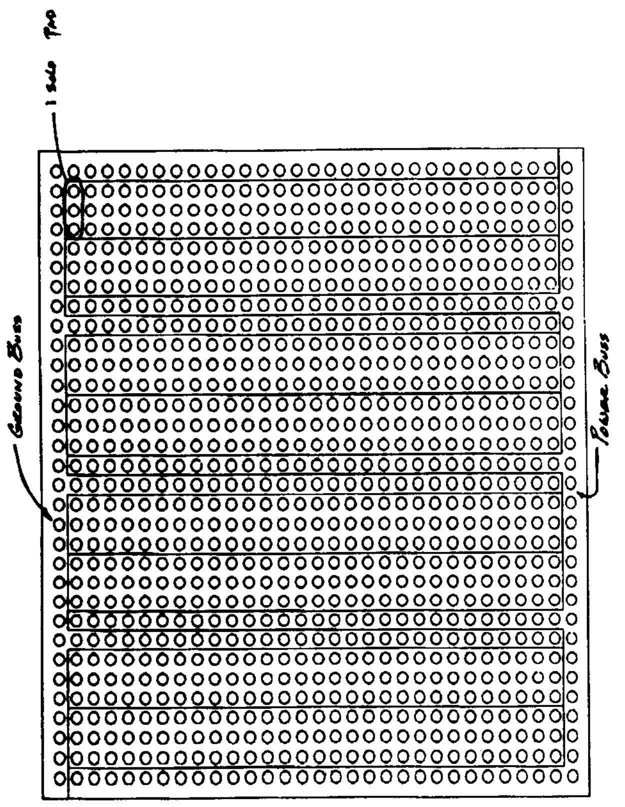


Figure 1

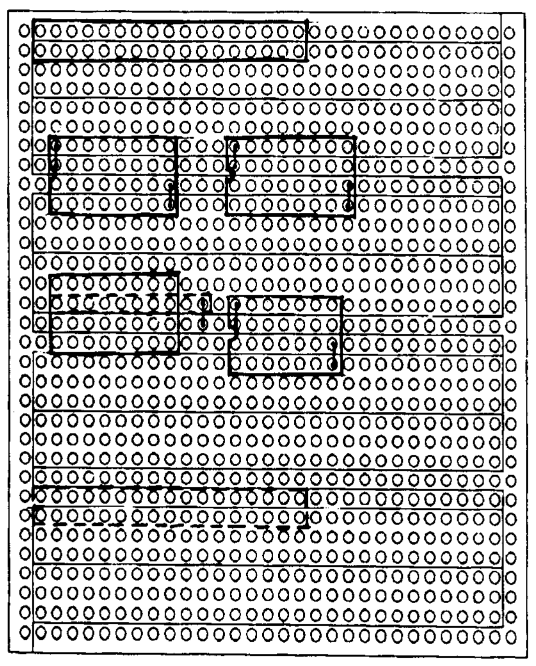


Figure 3

Real Computing

By Rick Rodman

32-Bit Systems

All Readers

TCP/IP & 32 Bit News

Tiny-TCP, the RS-485 network, Linux 1.1, and FreeBSD

Tiny-TCP

I'm still hard at work porting TinyTCP to the Xerox 820, using the Eco-C compiler for the Z-80, as I mentioned last time. One problem I ran into was the 6-character significance limit on the CP/M side, with the assembler and the compiler both. I hadn't worked with this limitation in a long time. This has meant that many names have had to be changed throughout the source, so the source files are changing rapidly. I had hoped to be able to substitute only one driver, XSIO, for the CASYNCMS serial driver. This driver uses interrupts both for sending and receiving.

I do intend to port this package to my NS32 system, the CompuPro CPU-32016-based machine running Bare Metal, once it is working on the Xerox. Since that's an S-100 system, I can add lots of serial ports to it with no difficulty. As you may recall, the NS32 CPUs, like Morotola-family CPUs, have no separate I/O space; the S-100 I/O space is mapped into the memory region FE0000 to FEFFFF hex. For most S-100 boards, only the low byte of the address is checked.

One reader suggested that we consider parallel-port Ethernet gizmos rather than using SLIP links. Linux includes drivers for some of these gizmos, which we could easily modify and use, and the Clarkson University folks have drivers for some of them as well. I'd have no problem lashing one of them to a PIO, either in the Xerox or an S-100 machine, but the problem is that these gizmos are quite expensive.

I'd like to add that one of our rules regarding this package is that it must not be allowed to become big and complicated. If we wanted something big and complicated, we would have started with KA9Q or something like that. Nevertheless, at some point, we could investigate UDP (User Datagram Protocol), a connectionless protocol which sits atop IP. There is a Trivial FTP (TFTP) which rides on top of UDP rather than TCP, and NFS rides on top of UDP also.

RS-485 network

Tilmann Reh sent me a TIFF file of his proposed RS-485 bus interface, which should be displayed alongside. Please consider this schematic.

On the RS-232 end, two optoisolators (K1 and K2) are used to isolate the computer from noise spikes induced into the network cable or various power circuits. Data out from the computer (TXD) drives the LED of the lower optoisolator (K2) for data going out onto the network; since the TXD line rests at logic 1 (-12V), the LED is normally off. Data coming back in drives the upper isolator (K1). The RXD line is pulled down most of the time, swinging between ground and +5. This *should* work in most cases; in other cases we may need to swing negative.

The collector of the optoisolator K1 is powered by DTR or RTS, which must be programmed to logic 0, causing it to swing positive. Is this logic true or false? I think it's logic true, but there are so many inversions I'm not sure.

When data is sent, a monostable is triggered, which is labelled IC1 and should be a 74HCT123 (not -132). The unused inputs mentioned are pins 5, 6, 7, 9, 10, and 11. This device is triggered by the start bits of the data. As mentioned previously, all we need to do from the software standpoint is to send an FF hex byte before the data. The software has to listen before sending data to make sure the bus is idle, and examine received data as it is sent to make sure there was no collision.

Sending and receiving on the RS-485 bus is accomplished by the LTC-485, IC2. Tilmann comments that the 75176 is cheaper.

The actual bus is a 4-pin connector. I would suggest using modular telephone cable, using RJ-11 (6 pin with 4 pins wired). However, because of the propensity of these cables to be sometimes twisted, sometimes not, we might want to think about this a little first. (Are modular telephone connectors used in other countries?)

At one end of the bus would be a wall transformer supplying +5 volt power to the devices on the bus side. I think a terminating resistor is required as well.

Digi-Key lists the 6N139 at $1.63, the 74HCT123 at $0.80, and the LTC485 at $2.25; throwing in a couple of dollars for resistors, caps and diodes, the total cost per node would be about $8. At levels such as these, labor and packaging costs greatly outweigh components costs. Anyone interested in making a small PCB?

Linux 1.1

Plug and Play Linux, from Yggdrasil and available less expensively from other dealers, is the best Unix for PCs available, and one of the best operating systems available overall. When you consider the price, there's no comparison at all.

The first thing you notice in this package is the excellent manual. This manual goes through the installation step-by-step, up to a certain point. All the bugs in the version 0.99 have been cleaned up, and the on-screen instructions greatly improved. There is now a "custom" installation option which requires around 35 megabytes of hard drive, a nice fit between the 2 megabyte CD-dependent and the 600 megabyte source-type installation.

I have a Dell machine with a PS/2 bus mouse, an ATI 8514-clone video board, a 500 MB IDE hard drive which I want *left alone,* and a 128-megabyte removable optical (MO erasable) drive. I had no problem at all installing Linux on the erasable optical (I had no problem with the earlier version, either). Everything went extemely smoothly.

There were two gotchas. As in version 0.99, I could not get the bus mouse to work at all. While the old manual had several pages of mostly unhelpful discussion about different mice, the new manual says nothing about mice at all. Plug a Microsoft-compatible serial mouse into one of your COM ports, and that'll work.

The other gotcha was the network card. As in version 0.99, again, the NE-2000 driver will only support interrupt 12. My board can't be set to interrupt 12; I can choose from 3, 4, 5, or 2 (9). Of these, the only usable one is 5. Unfortunately, the Linux driver doesn't allow me to select the interrupt.

Obviously, having source to everything, I could modify the drivers and fix both of these problems (assuming I can find out where the bus mouse is located, which may not be easy given the typically sparse documentation on PC hardware).

You're supposed to be able to use an SMC (Western Digital) Ethernet card; I have a couple of these, but they're in use in other machines at the moment. So, I wasn't able to test the networking. Linux supports 2 SLIP channels, so it could be used as a router for our TinyTCP network.

The X Window package provided is not just the basic X stuff, but includes a window manager called "fvwm" which looks very nice, X-window-compatible editors and lots of utilities, and some games - such as the beautiful and addictive Pool, which accurately simulates a pool table and has consumed more of my time than I care to mention.

Most outstanding is a "control-panel" program which allows you to interactively configure network addresses, install additional software to the hard drive, and make various changes to the system.

All the Gnu software is here, including Gnu EMACS and GCC. Also, there is a multimedia package called "Andrew" which looks very interesting; it's supposed to be some kind of hypertext system with photos, movies, sound, and so on. By the way, several sound cards are supported; the speaker is used if there's no sound card.

There's a tremendous amount to explore here: an Image Magic viewer, the MPEG player, lots of sample X window code, PBM, Ghostscript... and source to everything.

FreeBSD on the PC

I've also tried another PC Unix supplied on CD-ROM, FreeBSD from Walnut Creek CD-ROM. While similar in concept to Plug-and-Play Linux, this is a real BSD Unix kernel which comes with many of the same tools and X Window, and again, source for everything. There is no printed manual; all you get is online documentation.

The CD has what are called "Rock Ridge extensions". By this technique, if the CD is loaded on a Unix machine, the machine sees Unix-style mixed-case longer filenames; on a DOS machine, it sees DOS-style uppercase-only 8.3 filenames. This makes this CD useful, if not for anything else, as a nice source archive of Unix tools which can be recompiled on the Sun.

FreeBSD had no problem with my NE2000 on interrupt 5. However, it refused to recognize the removable optical drive and, therefore, tried to install itself on the CD-ROM, which, of course, didn't work. There's little you can control during the installation process to say, 'Hey, you're making a mistake here', so that's as far as I could get.

Magneto-optical drives have been around for quite some time, so it's rather hard for me to believe that any operating system written since 1985 or so would have any trouble with them. Someday, I suppose, I'll get around to connecting up a magnetic hard drive to try out FreeBSD. At this point, I can't imagine it having much to recommend it over Linux.

Those of you who don't really want to get involved with modifying the operating system may want to consider Mark Williams' new version of Coherent, which is available with X Window. It doesn't come with any source, but that may be alright for you. You don't need a CD-ROM - in these days of 20-diskette-plus installations, it installs from just 5 diskettes. I've mentioned before that Coherent is a mature, stable system which can easily be used in place of Unix in many applications.

For those of you who want to use X but don't want to have to learn Unix as well, the package for you may be Quarterdeck's Desqview/X. I've been told it now comes with a TCP/IP stack and a GCC-based developer's kit, so there's nothing else you need to buy.

Next time

Don't you hate it when columnists whine about having too much to say, so you'll have to wait for the next issue for that super-interesting topic you really wanted to read about? Er - well, owing to space considerations, the matter of CD-Recordable will have to wait til next issue.

Where to call or write

Real Computing BBS or Fax: +1 703 330 9049
E-mail: rickr@aib.com
Mail: 8329 Ivy Glen Court, Manassas VA 22110

Digi-Key 1-800-DIGI-KEY (1-800-344-4539)
701 Brooks Ave. South, POB 677, Thief River Falls MN 56701-0677

Plug-and-Play Linux 1.1 - $39.95
Yggdrasil Computing, Inc. +1 510 526 7531
P.O. Box 8418, Berkeley CA 94707-8418

Also available from:
Just Computers! +1 707 769 1648
P.O. Box 751414, Petaluma CA 94975-1414

FreeBSD $39.95
Walnut Creek CD-ROM +1 510 674 0783
1547 Palos Verdes Mall Suite 260, Walnut Creek CA 94596

Coherent $99.95, with X Window[s] $149.95
Mark Williams Company +1 800 636 6700 or +1 708 291 6700
60 Revere Drive, Northbrook IL 60062

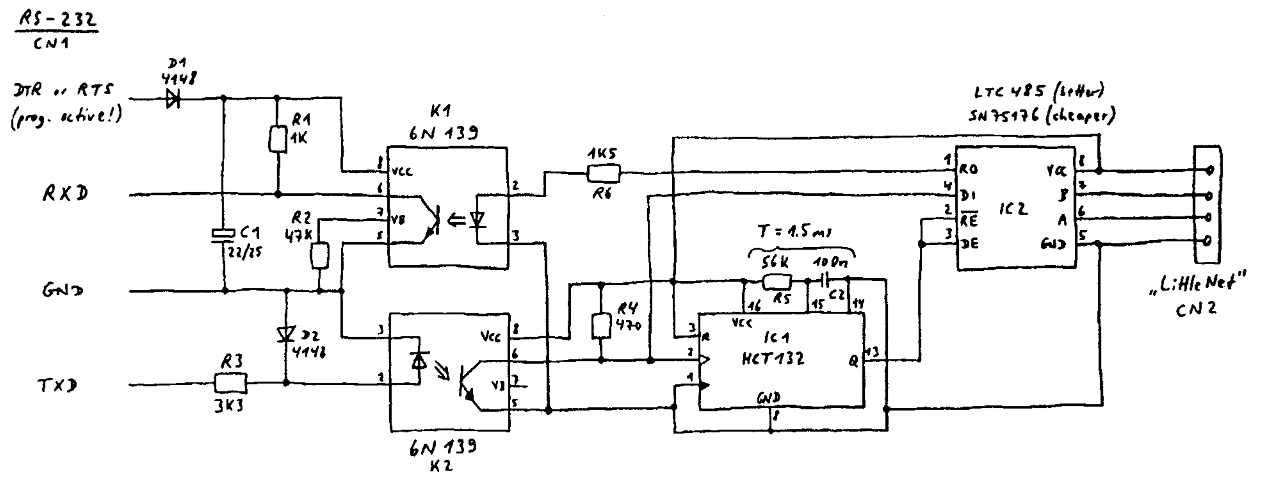
Desqview/X
Quarterdeck Office Systems
150 Pico Boulevard, Santa Monica CA 90405-9852

LINUX $29^{95}
Yggdrasil CDROM
Plug-and-Play * New Release
X Window System
TCP/IP, NFS and other protocols
GNU C, C++, Other Languages
Taylor UUCP, kermit, ZModem
Binaries and Source Code
Supports Most CDROM Drives

Just Computers!
(707) 769-1648
FAX: (707) 765-2447
P.O. Box 751414
Petaluma, CA 94975-1414
linux@justcomp.com
Visa/MasterCard
For auto-information, send e-mail to info@justcomp.com
Include word "help" in message

ISOLATED INTERFACE RS-232 TO RS-485

T. Reh 940713



add deblocking cap's 10µ + 100n at RS-485 Transceiver!
connect unused inputs of HCT132 with Ground!

Line Termination at both ends of LittleNet Bus cable : A—$\boxed{120}$—B

All Transceiver circuits powered with central supply via LittleNet VCC line (5V).

Special Feature

Intermediate Users

Part 6: High Level Words

MOVING FORTH

by Brad Rodriguez

Part 6: the Z80 high-level kernel

ERRATA

There are two goofs in the CAMEL80.AZM file I presented in *TCJ*#67. The minor goof is that the name length specified in the HEAD macro for the Forth word > was incorrectly typed as 2 instead of 1.

The major goof results from a subtlety of CP/M console I/O. KEY must not echo the typed character, and so used BDOS function 6. KEY? used BDOS function 11 to test non-destructively for the presence of a keypress. Unfortunately, BDOS function 6 does not "clear" the keypress detected by function 11! I have now rewritten KEY? to use BDOS function 6 (see Listing 1). Since this is a "destructive" test, I had to add logic to save the "consumed" keypress and return it when KEY is next used. This new logic can be used whenever your hardware (or operating system) provides only a destructive test-for-keypress.

HIGH LEVEL DEFINITIONS

In the last installment I did not expound greatly on the source code. Each Forth "primitive" performs a miniscule, sharply-defined function. It was almost all Z80 assembler code, and if it wasn't obvious *why* a particular word was included, I hope it was clear *what* each word did.

In this installment I have no such luxury: I will present the high level definitions which embody the elegant (and tortuous) logic of the Forth language. Entire books have been written [1,2,3] describing Forth kernels, and if you want complete mastery I highly recommend you buy one of them. For *TCJ* I'll limit myself to some of the key words of the compiler and interpreter, given in Listing 2.

TEXT INTERPRETER OPERATION

The text or "outer" interpreter is the Forth code which accepts input from the keyboard and performs the desired Forth operations. (This is distinct from the address or "inner" interpreter, NEXT, which executes compiled threaded code.) The best way to understand it is to work through the startup of the Forth system.

1. The CP/M entry point (see listing in previous installment) determines the top of available memory, set the stack pointers (PSP,RSP) and user pointer (UP), establishing the memory map shown in Figure 1. It then sets the "inner" interpreter pointer (IP) to execute the Forth word **COLD**.

2. **COLD** initializes the user variables from a startup table, and then does **ABORT**. (**COLD** will also attempt to execute a Forth command from the CP/M command line.)

3. **ABORT** resets the parameter stack pointer and does **QUIT**.

4. **QUIT** resets the return stack pointer, loop stack pointer, and interpret state, and then begins to interpret Forth commands. (The name is apt because **QUIT** can be used to abort an application and get back to the "top level" of Forth. Unlike **ABORT**, **QUIT** will leave the parameter stack contents alone.) **QUIT** is an infinite loop which will **ACCEPT** a line from the keyboard, and then **INTERPRET** it as Forth commands. When not compiling, **QUIT** will prompt "ok" after each line.

5. **INTERPRET** is an almost verbatim translation of the algorithm given in section 3.4 of the ANS Forth document. It parses one space-delimited string from the input, and tries to **FIND** the Forth word of that name. If the word is found, it will be either executed (if it is an IMMEDIATE word, or if in the "interpret" state, STATE=0) or compiled into the dictionary (if in the "compile" state, STATE<>0). If not found, Forth attempts to convert the string as a number. If successful, **LITERAL** will either place it on the parameter stack (if in "interpret" state) or compile it as an in-line literal value (if in "compile" state). If not a Forth word and not a valid number, the string is typed, an error message is displayed, and the interpreter **ABORT**s. This process is repeated, string by string, until the end of the input line is reached.

THE FORTH DICTIONARY

Whoa! How does the interpreter "find" a Forth word by name? Answer: Forth keeps a "dictionary" of the names of all Forth words. Each name is connected in some fashion with the executable code for the corresponding word.

There are many ways to store a set of strings for searching: a simple array, a linked list, a multiple linked list, hash table, etc. Almost all are valid here — all Forth asks is that, if you reuse a name, the *latest* definition is found when you search the dictionary.

It's also possible to have several sets of names ("vocabularies", or "wordlists" in the new ANSI jargon). This lets you reuse a name *without* losing its previous meaning. For example, you could have an integer +, a floating-point +, even a + for strings...one way to achieve the "operator overloading" so beloved by the object-oriented community.

Each string may be connected with its executable code by being physically adjacent in memory — i.e., the name appears in memory just before the executable code, thus being called the "head" or "header" of the Forth word. Or the strings may be located in a totally different part of memory, and connected with pointers to executable code ("separate heads").

You can even have unnamed ("headless") fragments of Forth code, if you *know* you'll never need to compile or interpret them. ANSI only requires that the ANS Forth words be findable.

The design decisions could fill another article. Suffice it to say that CamelForth uses the simplest scheme: a single linked list, with the header located just before the executable code. No vocabularies... although I may add them in a future issue of *TCJ*.

HEADER STRUCTURE (FIGURE 2)

Still more design decisions: what data should be present in the header, and how should it be stored?

The minimum data is the name, precedence bit, and pointer (explicit or implicit) to executable code. For simplicity, CamelForth stores the name as a "counted string" (one byte of length, followed by N characters). Early Forth Inc. products stored a length but only the first three characters, for faster comparisons (the actual improvement gained is another hot debate). Fig-Forth compromised, flagging the last character with MSB high in order to allow either full-length or truncated names. Other Forths have used packed strings [4], and I suspect even C-style null-terminated strings have been used.

The "precedence bit" is a flag which indicates if this word has IMMEDIATE status. IMMEDIATE words are executed *even during compilation*, which is how Forth implements compiler directives and control structres. There are other ways to distinguish compiler directives — Pygmy Forth [5], for example, puts them in a separate vocabulary. But ANS Forth essentially mandates the use of a precedence bit [6]. Many Forths store this bit in the "length" byte. I have chosen to put it in a separate byte, in order to use the "normal" string operators on word names (e.g. S= within **FIND**, and **TYPE** within **WORDS**).

If the names are kept in a linked list, there must be a link. Usually the latest word is at the head of the linked list, and the link points to a previous word. This enforces the ANSI (and traditional) requirement for redefined words. Charles Curley [7] has studied the placement of the link field, and found that the compiler can be made significantly faster if the link field comes *before* the name (rather than after, as was done in Fig-Forth).

Figure 2 shows the structure of the CamelForth word header, and the Fig-Forth, F83, and Pygmy Forth headers for comparison. The "view" vield of F83 and Pygmy is an example of other useful information which can be stored in the Forth word header.

Remember: it's important to distinguish the header from the "body" (executable part) of the word. They need not be stored together. The header is only used during compilation and interpretation, and a "purely executable" Forth application could dispense with headers entirely. However, headers must be present — at least for the ANSI word set — for it to be a legal ANS Forth System.

When "compiling" a Forth system from assembler source code, you can define macros to build this header (see HEAD and IMMED in CAMEL80.AZM). In the Forth environment the header, *and the Code Field*, is constructed by the word **CREATE**.

COMPILER OPERATION

We now know enough to understand the Forth compiler. The word : starts a new high-level definition, by creating a header for the word (**CREATE**), changing its Code Field to "docolon" (**!COLON**), and switching to compile state (**]**). Recall that, in compile state, every word encountered by the text interpreter is compiled into the dictionary instead of being executed. This will continue until the word ; is encountered. Being an IMMEDIATE word, ; will execute, compiling an **EXIT** to end the definition, and then switching back to interpret state (**[**).

Also, : will **HIDE** the new word, and ; will **REVEAL** it (by setting and clearing the "smudge" bit in the name). This is to allow a Forth word to be redefined in terms of its "prior self". To force a recursive call to the word being defined, use **RECURSE**.

Thus we see that there is no distinct Forth "compiler", in the same sense that we would speak of a C or Pascal compiler. The Forth compiler is embodied in the actions of various Forth words. This makes it easy for you to change or extend the compiler, but makes it difficult to create a Forth application *without* a built-in compiler!

THE DEPENDENCY WORD SET

Most of the remaining high-level words are either a) necessary to implement the compiler and interpreter, or b) provided solely for your programming pleasure. But there is one set which deserves special mention: the words I have separated into the file CAMEL80D.AZM (Listing 3).

One of the goals of the ANSI Forth Standard was to hide CPU and model dependencies (Direct or Indirect Threaded? 16 or 32 bit?) from the application programmer. Several words were added to the Standard for this purpose. I have taken this one step further, attempting to encapsulate these dependencies *even within the kernel*. Ideally, the high-level Forth code in the file CAMEL80H.AZM should be the same for all CamelForth targets (although different assemblers will have different syntax).

Differences in cell size and word alignment are managed by the ANS Forth words **ALIGN ALIGNED CELL+ CELLS CHAR+ CHARS** and my own addition, **CELL** (equivalent to **1 CELLS**, but smaller when compiled).

The words **COMPILE, !CF ,CF !COLON** and **,EXIT** hide peculiarities of the threading model, such as a) how are the threads represented, and b) how is the Code Field implemented? The value of these words becomes evident when you look at the differences between the direct-threaded Z80 and the subroutine-threaded 8051:

word	compiles on Z80	compiles on 8051
COMPILE,	address	LCALL address
!CF	CALL address	LCALL address
,CF	!CF & allot 3 bytes	!CF & allot 3 bytes
!COLON	CALL docolon	nothing!
,EXIT	address of EXIT	RET

(**!CF** and **,CF** are different for indirect-threaded Forths.)

In similar fashion, the words **,BRANCH ,DEST** and **!DEST** hide the implementation of high-level branch and loop operators. I have tried to invent — without borrowing from existing Forths! — the minimal set of operators which can factor out all the implementation differences. Only time, expert criticism, and many CamelForths will tell how successful I've been.

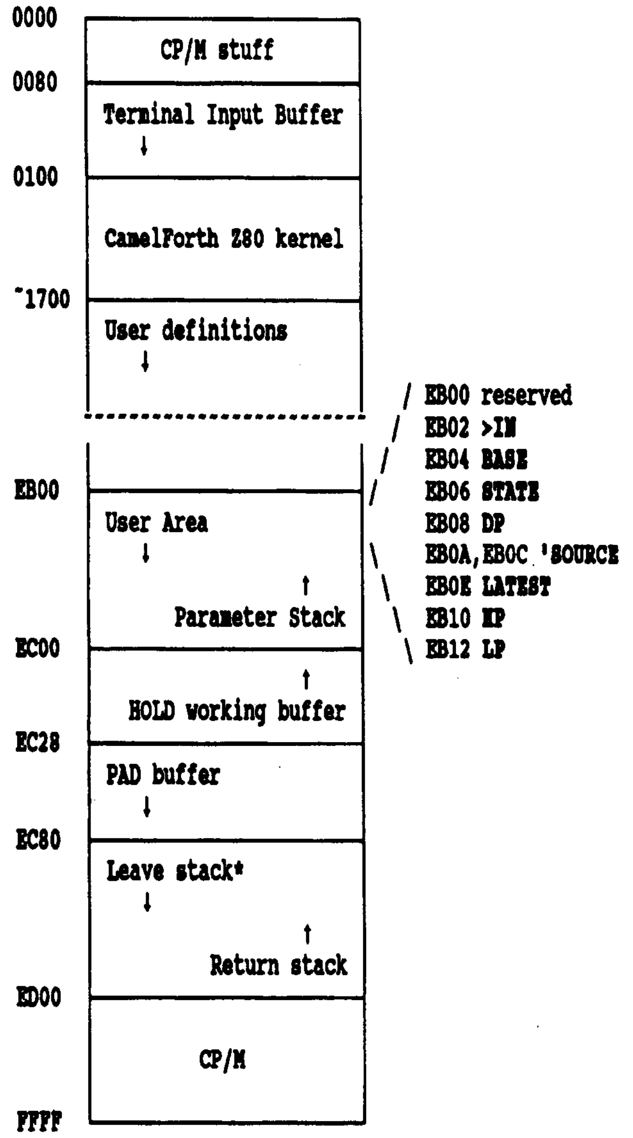
So far I have *not* been successful factoring the differences in header structure into a similar set of words. The words **FIND** and **CREATE** are so intimately involved with the header contents that I haven't yet found suitable subfactors. I have made a start, with the words **NFA>LFA NFA>CFA IMMED? HIDE REVEAL** and the ANS Forth words **>BODY IMMEDIATE**. I'll continue to work on this. Fortunately, it is practical for the time being to use the identical header structure on all CamelForth implementations (since they're all byte-addressed 16-bit Forths).

NEXT TIME...

I will probably present the 8051 kernel, and talk about how the Forth compiler and interpreter are modified for Harvard architectures (computers that have logically distinct memories for Code and Data, like the 8051). For the 8051 I will print the files CAMEL51 and CAMEL51D, but probably only excerpts from CAMEL51H, since (except for formatting of the assembler file) the high-level code shouldn't be different from what I've presented this issue...and Bill needs the space for other articles! Don't worry, the full code will be uploaded to GEnie.

FIGURE 1. Z80 CP/M CAMELFORTH MEMORY MAP

assuming CP/M BDOS starts at ED00 hex.



*** used during compilation of DO..LOOPs.**

However, I may succumb to demands of Scroungemaster II builders, and publish the 6809 CamelForth configured for the Scroungemaster II board. Whichever I do next, I'll do the other just one installment later.

REFERENCES

1. Derick, Mitch and Baker, Linda, Forth Encyclopedia, Mountain View Press, Route 2 Box 429, La Honda, CA 94020 USA (1982). Word-by-word description of Fig-Forth.

2. Ting, C. H., Systems Guide to fig-Forth, Offete Enterprises, 1306 South B Street, San Mateo, CA 94402 USA (1981).

3. Ting, C. H., Inside F83, Offete Enterprises (1986).

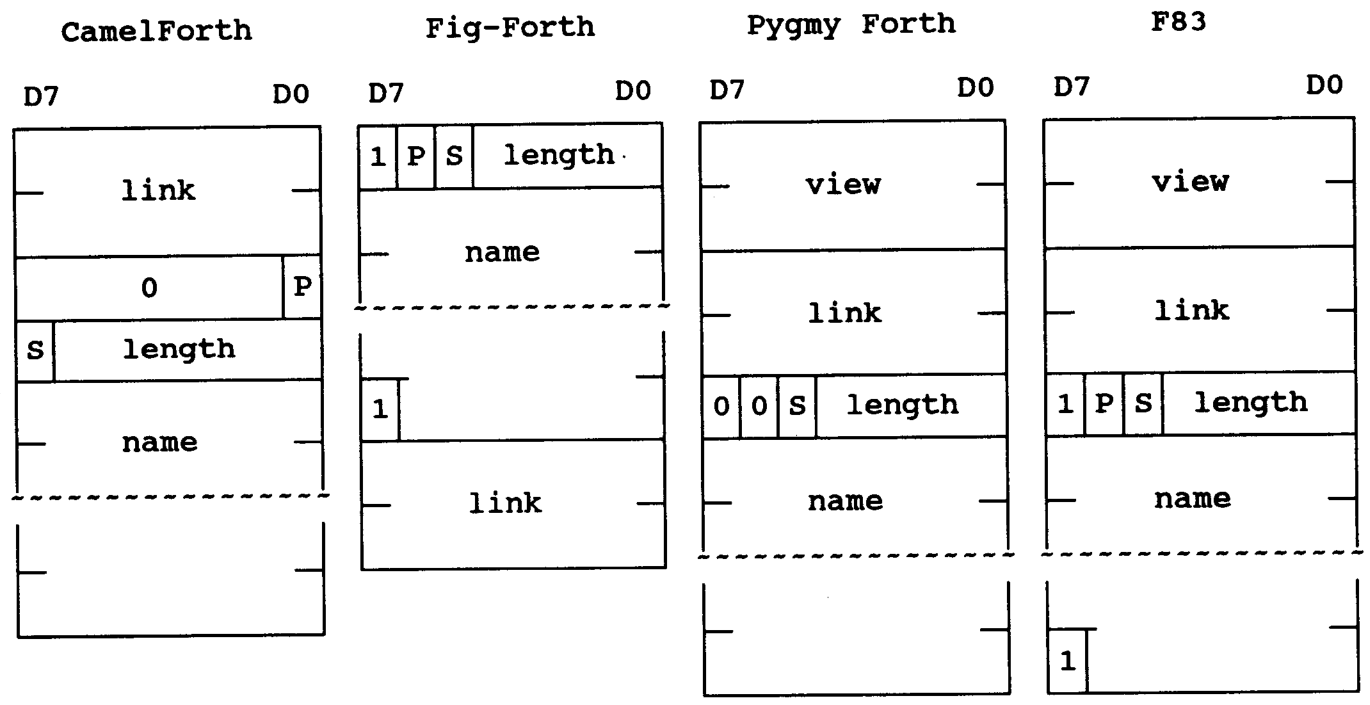
5. Sergeant, Frank, Pygmy Forth for the IBM PC, version 1.4 (1992). Distributed by the author, available from the Forth Interest Group (P.O. Box 2154, Oakland CA 94621 USA) or on GEnie.

6. J. E. Thomas examined this issue thoroughly when converting Pygmy Forth to an ANSI Forth. No matter what tricks you play with relinking words, strict ANSI compliance is violated. A regrettable decision on the part of the ANS Forth team.

7. In private communication.

The source code for Z80 CamelForth is *now* available on GEnie as CAMEL80.ARC in the CP/M and Forth Roundtables. Really. I just uploaded it. (Apologies to those who have been waiting.)

FIGURE 2. HEADER STRUCTURES



Link - in CamelForth and Fig-Forth, points to the previous word's Length byte. In Pygmy Forth and F83, points to the previous word's Link.
P - Precedence bit, equals 1 for an IMMEDIATE word (not used in Pygmy).
S - Smudge bit, used to prevent FIND from finding this word.
1 - in Fig-Forth and F83, the length byte and the last character of the name are flagged with a 1 in the most significant bit (bit 7).
View - in Pygmy Forth and F83, contains the block number of the source code for this word.

TABLE 1. GLOSSARY OF "HIGH LEVEL" WORDS
(files CAMEL80D.AZM and CAMEL80H.AZM)

NAME stack in — stack out description

**Guide to stack diagrams: R: = return stack,
c = 8-bit character, flag = boolean (0 or -1),
n = signed 16-bit, u = unsigned 16-bit,
d = signed 32-bit, ud = unsigned 32-bit,
+n = unsigned 15-bit, x = any cell value,
i*x j*x = any number of cell values,
a-addr = aligned adrs, c-addr = character adrs
p-addr = I/O port adrs, sys = system-specific.
Refer to ANS Forth document for more details.**

ANS Forth Core words
These are required words whose definitions are specified by the ANS Forth document.

#	ud1 — ud2	convert 1 digit of output
#S	ud1 — ud2	convert remaining digits
#>	ud1 — c-addr u	end conv., get string
'	— xt	find word in dictionary
(	—	skip input until)
*	n1 n2 — n3	signed multiply
*/	n1 n2 n3 — n4	n1*n2/n3
*/MOD	n1 n2 n3 — n4 n5	n1*n2/n3, rem & quot
+LOOP	adrs — L: 0 a1 a2 .. aN —	
,	x —	append cell to dict
/	n1 n2 — n3	signed divide
/MOD	n1 n2 — n3 n4	signed divide, rem & quot
:	—	begin a colon definition
;		end a colon definition
<#	—	begin numeric conversion
>BODY	xt — a-addr	adrs of param field
>IN	— a-addr	holds offset into TIB
>NUMBER	ud adr u — ud' adr' u'	convert string to number
2DROP	x1 x2 —	drop 2 cells
2DUP	x1 x2 — x1 x2 x1 x2	dup top 2 cells
2OVER	x1 x2 x3 x4 — x1 x2 x3 x4 x1 x2	per diag
2SWAP	x1 x2 x3 x4 — x3 x4 x1 x2	per diagram
2!	x1 x2 a-addr —	store 2 cells
2@	a-addr — x1 x2	fetch 2 cells
ABORT	i*x — R: j*x —	clear stack & QUIT
ABORT"	i*x 0 — i*x R: j*x — j*x	print msg &
	i*x x1 — R: j*x —	abort,x1<>0
ABS	n1 — +n2	absolute value
ACCEPT	c-addr +n — +n'	get line from terminal
ALIGN	—	align HERE
ALIGNED	addr — a-addr	align given addr
ALLOT	n —	allocate n bytes in dict
BASE	— a-addr	holds conversion radix
BEGIN	— adrs	target for backward branch
BL	— char	an ASCII space
C,	char —	append char to dict
CELLS	n1 — n2	cells->adrs units
CELL+	a-addr1 — a-addr2	add cell size to adrs
CHAR	— char	parse ASCII character
CHARS	n1 — n2	chars->adrs units
CHAR+	c-addr1 — c-addr2	add char size to adrs
COUNT	c-addr1 — c-addr2 u	counted->adr/len
CR	—	output newline
CREATE	—	create an empty definition
DECIMAL	—	set number base to decimal
DEPTH	— +n	number of items on stack
DO	— adrs L: — 0	start of DO..LOOP
DOES>	—	change action of latest def'n
ELSE	adrs1 — adrs2	branch for IF..ELSE
ENVIRONMENT?	c-addr u — false	system query
EVALUATE	i*x c-addr u — j*x	interpret string
FIND	c-addr — c-addr 0	..if name not found
	xt 1	..if immediate
	xt	..if "normal"
FM/MOD	d1 n1 — n2 n3	floored signed division
HERE	— addr	returns dictionary pointer
HOLD	char —	add char to output string
IF	— adrs	conditional forward branch
IMMEDIATE	—	make last def'n immediate
LEAVE	— L: — adrs	exit DO..LOOP
LITERAL	x —	append numeric literal to dict.
LOOP	adrs — L: 0 a1 a2 .. aN —	
MAX	n1 n2 — n3	signed maximum
MIN	n1 n2 — n3	signed minimum
MOD	n1 n2 — n3	signed remainder
MOVE	addr1 addr2 u —	smart move
M*	n1 n2 — d	signed 16*16->32 multiply
POSTPONE	—	postpone compile action of word
QUIT	— R: i*x —	interpret from keyboard
RECURSE	—	recurse current definition
REPEAT	adrs1 adrs2 —	resolve WHILE loop
SIGN	n —	add minus sign if n<0
SM/REM	d1 n1 — n2 n3	symmetric signed division
SOURCE	— adr n	current input buffer
SPACE	—	output a space
SPACES	n —	output n spaces
STATE	— a-addr	holds compiler state
S"	—	compile in-line string
."	—	compile string to print
S>D	n — d	single -> double precision
THEN	adrs —	resolve forward branch
TYPE	c-addr +n —	type line to terminal
UNTIL	adrs —	conditional backward branch
U.	u —	display u unsigned
.	n —	display n signed
WHILE	— adrs	branch for WHILE loop
WORD	char — c-addr n	parse word delim by char
[	—	enter interpretive state
[CHAR]	—	compile character literal
[']	—	find word & compile as literal
]	—	enter compiling state

ANS Forth Extensions
These are optional words whose definitions are specified by the ANS Forth document.

.S	—	print stack contents
/STRING	a u n — a+n u-n	trim string
AGAIN	adrs —	uncond'l backward branch
COMPILE,	xt —	append execution token
DABS	d1 — +d2	absolute value, dbl.prec.
DNEGATE	d1 — d2	negate, double precision
HEX	—	set number base to hex
PAD	— a-addr	user PAD buffer
TIB	— a-addr	Terminal Input Buffer
WITHIN	n1|u1 n2|u2 n3|u3 — f	test n2<=n1<n3?
WORDS	—	list all words in dict.

Private Extensions
These are words which are unique to CamelForth. Many of these are necessary to implement ANS Forth words, but are not specified by the ANS document. Others are functions I find useful.

!CF	adrs cfa —	set code action of a word
!COLON	—	change code field to docolon
!DEST	dest adrs —	change a branch dest'n
#INIT	— n	#bytes of user area init data
'SOURCE	— a-addr	two cells: len, adrs
(DOES>)	—	run-time action of DOES>
(S")	— c-addr u	run-time code for S"
,BRANCH	xt —	append a branch instruction
,CF	adrs —	append a code field
,DEST	dest —	append a branch address
,EXIT	—	append hi-level EXIT action
>COUNTED	src n dst —	copy to counted str
>DIGIT	n — c	convert to 0..9A..Z
>L	x — L: — x	move to Leave stack
?ABORT	f c-addr u —	abort & print msg
?DNEGATE	d1 n — d2	negate d1 if n negative
?NEGATE	n1 n2 — n3	negate n1 if n2 negative
?NUMBER	c-addr — n -1	convert string->number
	— c-addr 0	if convert error
?SIGN	adr n — adr' n' f	get optional sign
		advance adr/n if sign; return NZ if negative
CELL	— n	size of one cell
COLD	—	cold start Forth system
COMPILE	—	append inline execution token
DIGIT?	c — n -1	..if c is a valid digit
	— x 0	..otherwise
DP	— a-addr	holds dictionary ptr
ENDLOOP	adrs xt — L: 0 a1 a2 .. aN —	
HIDE	—	"hide" latest definition
HP	— a-addr	HOLD pointer
IMMED?	nfa — f	fetch immediate flag
INTERPRET	i*x c-addr u — j*x	interpret given buffer
L0	— a-addr	bottom of Leave stack
LATEST	— a-addr	last word in dictionary
LP	— a-addr	Leave-stack pointer
L>	— x L: x —	move from Leave stack
NFA>CFA	nfa — cfa	name adr -> code field
NFA>LFA	nfa — lfa	name adr -> link field
R0	— a-addr	end of return stack
REVEAL	—	"reveal" latest definition
S0	— a-addr	end of parameter stack
TIBSIZE	— n	size of TIB
U0	— a-addr	current user area adrs
UD*	ud1 d2 — ud3	32*16->32 multiply
UD/MOD	ud1 u2 — u3 ud4	32/16->32 divide
UINIT	— addr	initial values for user area
UMAX	u1 u2 — u	unsigned maximum
UMIN	u1 u2 — u	unsigned minimum

LISTING 1. Errata in CAMEL80.AZM from TCJ #67.

```
; Can't mix BDOS function 6 and BDOS function 0B.
; =============================================================

;Z SAVEKEY  -- addr  temporary storage for KEY?
    head savekey,7,SAVEKEY,dovar
                DW 0

;X KEY?     — f    return true if char waiting
;   0FF 6 BDOS DUP SAVEKEY C! ;   rtns 0 or key
; must use BDOS function 6 to work with KEY
    head querykey,4,KEY?,docolon
                DW LIT,0FFH,LIT,06H,BDOS
                DW DUP,SAVEKEY,CSTORE,EXIT

;C KEY      — c    get character from keyboard
;   BEGIN SAVEKEY C@ 0= WHILE KEY? DROP REPEAT
;   SAVEKEY C@  0 SAVEKEY C! ;
; must use CP/M direct console I/O to avoid echo
; (BDOS function 6, contained within KEY?)
    head KEY,3,KEY,docolon
KEY1:       DW SAVEKEY,CFETCH,ZEROEQUAL
            DW qbranch,KEY2
            DW QUERYKEY,DROP,branch,KEY1
KEY2:       DW SAVEKEY,CFETCH,LIT,0
            DW SAVEKEY,CSTORE
            DW EXIT

; Length in '>' head is incorrect.
; ================================

;C >     n1 n2 — flag       test n1>n2, signed
    head GREATER,1,>,docolon
            DW SWOP,LESS,EXIT
```

; LISTING 2.;

```
; ===============================================================
; CamelForth for the Zilog Z80
; (c) 1994 Bradford J. Rodriguez
; Permission is granted to freely copy, modify,
; and distribute this program for personal or
; educational use.  Commercial inquiries should
; be directed to the author at 221 King St. E.,
; #32, Hamilton, Ontario L8N 1B5 Canada
;
; CAMEL80H.AZM: High Level Words
;   Source code is for the Z80MR macro assembler.
;   Forth words are documented as follows:
;*   NAME     stack — stack    description
;   Word names in upper case are from the ANS
;   Forth Core word set.  Names in lower case are
;   "internal" implementation words & extensions.
;
; ===============================================================
; SYSTEM VARIABLES & CONSTANTS
; ==================

;C BL        — char      an ASCII space
    head BL,2,BL,docon
            dw 20h

;Z tibsize  — n          size of TIB
    head TIBSIZE,7,TIBSIZE,docon
            dw 126      ; 2 chars safety zone

;X tib       — a-addr   Terminal Input Buffer
;  HEX 80 CONSTANT TIB   CP/M systems: 128 bytes
;  HEX -80 USER TIB   others: below user area
    head TIB,3,TIB,docon
            dw 80h

;Z u0        — a-addr   current user area adrs
;  0 USER U0
    head U0,2,U0,douser
            dw 0

;C >IN       — a-addr   holds offset into TIB
;  2 USER >IN
    head TOIN,3,>IN,douser
            dw 2

;C BASE    — a-addr    holds conversion radix
;  4 USER BASE
    head BASE,4,BASE,douser
            dw 4

;C STATE   — a-addr   holds compiler state
;  6 USER STATE
    head STATE,5,STATE,douser
            dw 6

;Z dp        — a-addr    holds dictionary ptr
;  8 USER DP
    head DP,2,DP,douser
            dw 8

;Z 'source  — a-addr     two cells: len, adrs
; 10 USER 'SOURCE
;    head TICKSOURCE,7,'SOURCE,douser
            DW link       ; must expand
            DB 0          ; manually
link    DEFL $            ; because of
            DB 7,27h,'SOURCE'     ; tick character
TICKSOURCE: call douser           ; in name!
            dw 10
```

```
;Z latest   — a-addr     last word in dict.
;  14 USER LATEST
   head LATEST,6,LATEST,douser
            dw 14

;Z hp       — a-addr    HOLD pointer
;  16 USER HP
   head HP,2,HP,douser
            dw 16

;Z LP       — a-addr    Leave-stack pointer
;  18 USER LP
   head LP,2,LP,douser
            dw 18

;Z s0       — a-addr    end of parameter stack
   head S0,2,S0,douser
            dw 100h

;X PAD      — a-addr    user PAD buffer
;           = end of hold area!
   head PAD,3,PAD,douser
            dw 128h

;Z l0       — a-addr    bottom of Leave stack
   head L0,2,L0,douser
            dw 180h

;Z r0       — a-addr    end of return stack
   head R0,2,R0,douser
            dw 200h

;Z uinit   — addr  initial values for user area
   head UINIT,5,UINIT,docreate
            DW 0,0,10,0           ;
reserved,>IN,BASE,STATE
            DW enddict; DP
            DW 0,0     ; SOURCE init'd elsewhere
            DW lastword          ; LATEST
            DW 0       ; HP init'd elsewhere

;Z #init   — n    #bytes of user area init data
   head NINIT,5,#INIT,docon
            DW 18

; ARITHMETIC OPERATORS
==========================

;C S>D   n — d          single -> double prec.
;  DUP 0< ;
   head STOD,3,S>D,docolon
            dw DUP,ZEROLESS,EXIT

;Z ?NEGATE  n1 n2 — n3  negate n1 if n2 negative
;  0< IF NEGATE THEN ;          ...a common factor
   head QNEGATE,7,?NEGATE,docolon
            DW ZEROLESS,qbranch,QNEG1,NEGATE
QNEG1:      DW EXIT

;C ABS      n1 — +n2   absolute value
;  DUP ?NEGATE ;
   head ABS,3,ABS,docolon
            DW DUP,QNEGATE,EXIT

;X DNEGATE   d1 — d2              negate double
precision
;  SWAP INVERT SWAP INVERT 1 M+ ;
   head DNEGATE,7,DNEGATE,docolon
            DW SWOP,INVERT,SWOP
            DW INVERT,LIT,1,MPLUS
            DW EXIT

;Z ?DNEGATE  d1 n — d2  negate d1 if n negative
;  0< IF DNEGATE THEN ;         ...a common factor
   head QDNEGATE,8,?DNEGATE,docolon
            DW ZEROLESS,qbranch,DNEG1,DNEGATE
DNEG1:      DW EXIT

;X DABS    d1 — +d2   absolute value dbl.prec.
;  DUP ?DNEGATE ;
   head DABS,4,DABS,docolon
            DW DUP,QDNEGATE,EXIT

;C M*       n1 n2 — d   signed 16*16->32 multiply
;  2DUP XOR >R        carries sign of the result
;  SWAP ABS SWAP ABS UM*
;  R> ?DNEGATE ;
   head MSTAR,2,M*,docolon
            DW TWODUP,XOR,TOR
            DW SWOP,ABS,SWOP,ABS,UMSTAR
            DW RFROM,QDNEGATE,EXIT

;C SM/REM  d1 n1 — n2 n3  symmetric signed div
;  2DUP XOR >R         sign of quotient
;  OVER >R             sign of remainder
;  ABS >R DABS R> UM/MOD
;  SWAP R> ?NEGATE
;  SWAP R> ?NEGATE ;
; Ref. dpANS-6 section 3.2.2.1.
   head SMSLASHREM,6,SM/REM,docolon
            DW TWODUP,XOR,TOR,OVER,TOR
            DW ABS,TOR,DABS,RFROM,UMSLASHMOD
            DW SWOP,RFROM,QNEGATE
            DW SWOP,RFROM,QNEGATE
            DW EXIT

;C FM/MOD  d1 n1 — n2 n3  floored signed div'n
;  DUP >R  save divisor
;  SM/REM
;  DUP 0< IF              if quotient negative,
;           SWAP R> +              add divisor to rem'dr
;           SWAP 1-   decrement quotient
;  ELSE R> DROP THEN ;
; Ref. dpANS-6 section 3.2.2.1.
   head FMSLASHMOD,6,FM/MOD,docolon
            DW DUP,TOR,SMSLASHREM
            DW DUP,ZEROLESS,qbranch,FMMOD1
            DW SWOP,RFROM,PLUS,SWOP,ONEMINUS
            DW branch,FMMOD2
FMMOD1:  DW RFROM,DROP
FMMOD2:  DW EXIT

;C *        n1 n2 — n3 signed multiply
;  M* DROP ;
   head STAR,1,*,docolon
            dw MSTAR,DROP,EXIT

;C /MOD   n1 n2 — n3 n4   signed divide/rem'dr
;  >R S>D R> FM/MOD ;
   head SLASHMOD,4,/MOD,docolon
            dw TOR,STOD,RFROM,FMSLASHMOD,EXIT

;C /        n1 n2 — n3 signed divide
;  /MOD nip ;
   head SLASH,1,/,docolon
            dw SLASHMOD,NIP,EXIT

;C MOD   n1 n2 — n3   signed remainder
;  /MOD DROP ;
   head MOD,3,MOD,docolon
            dw SLASHMOD,DROP,EXIT

;C */MOD  n1 n2 n3 — n4 n5   n1*n2/n3, rem&quot
;  >R M* R> FM/MOD ;
   head SSMOD,5,*/MOD,docolon
            dw TOR,MSTAR,RFROM,FMSLASHMOD,EXIT

;C */       n1 n2 n3 — n4          n1*n2/n3
;  */MOD nip ;
   head STARSLASH,2,*/,docolon
            dw SSMOD,NIP,EXIT

;C MAX   n1 n2 — n3   signed maximum
;  2DUP < IF SWAP THEN DROP ;
   head MAX,3,MAX,docolon
            dw TWODUP,LESS,qbranch,MAX1,SWOP
MAX1:       dw DROP,EXIT

;C MIN   n1 n2 — n3   signed minimum
;  2DUP > IF SWAP THEN DROP ;
   head MIN,3,MIN,docolon
            dw TWODUP,GREATER,qbranch,MIN1,SWOP
MIN1:       dw DROP,EXIT

; DOUBLE OPERATORS
==============================

;C 2@   a-addr — x1 x2   fetch 2 cells
;  DUP CELL+ @ SWAP @ ;
;  the lower address will appear on top of stack
   head TWOFETCH,2,2@,docolon
            dw DUP,CELLPLUS,FETCH
            dw SWOP,FETCH,EXIT

;C 2!   x1 x2 a-addr —    store 2 cells
;  SWAP OVER ! CELL+ ! ;
;  the top of stack is stored at the lower adrs
   head TWOSTORE,2,2!,docolon
            dw SWOP,OVER,STORE
            dw CELLPLUS,STORE,EXIT

;C 2DROP  x1 x2 —      drop 2 cells
;  DROP DROP ;
   head TWODROP,5,2DROP,docolon
            dw DROP,DROP,EXIT

;C 2DUP   x1 x2 — x1 x2 x1 x2   dup top 2 cells
;  OVER OVER ;
   head TWODUP,4,2DUP,docolon
            dw OVER,OVER,EXIT

;C 2SWAP  x1 x2 x3 x4 — x3 x4 x1 x2  per diagram
;  ROT >R ROT R> ;
   head TWOSWAP,5,2SWAP,docolon
            dw ROT,TOR,ROT,RFROM,EXIT

;C 2OVER  x1 x2 x3 x4 — x1 x2 x3 x4 x1 x2
;  >R >R 2DUP R> R> 2SWAP ;
   head TWOOVER,5,2OVER,docolon
            dw TOR,TOR,TWODUP,RFROM,RFROM
            dw TWOSWAP,EXIT

; INPUT/OUTPUT
==================================

;C COUNT   c-addr1 — c-addr2 u  counted->adr/len
;  DUP CHAR+ SWAP C@ ;
   head COUNT,5,COUNT,docolon
            dw DUP,CHARPLUS,SWOP,CFETCH,EXIT

;C CR       —            output newline
;  0D EMIT 0A EMIT ;
   head CR,2,CR,docolon
            dw lit,0dh,EMIT,lit,0ah,EMIT,EXIT

;C SPACE  —              output a space
;  BL EMIT ;
   head SPACE,5,SPACE,docolon
            dw BL,EMIT,EXIT

;C SPACES   n —         output n spaces
;  BEGIN DUP WHILE SPACE 1- REPEAT DROP ;
   head SPACES,6,SPACES,docolon
SPCS1:      DW DUP,qbranch,SPCS2
            DW SPACE,ONEMINUS,branch,SPCS1
SPCS2:      DW DROP,EXIT

;Z umin      u1 u2 — u   unsigned minimum
;  2DUP U> IF SWAP THEN DROP ;
   head UMIN,4,UMIN,docolon
            DW TWODUP,UGREATER
            DW QBRANCH,UMIN1,SWOP
UMIN1:      DW DROP,EXIT

;Z umax    u1 u2 — u    unsigned maximum
;  2DUP U< IF SWAP THEN DROP ;
   head UMAX,4,UMAX,docolon
            DW TWODUP,ULESS
            DW QBRANCH,UMAX1,SWOP
UMAX1:      DW DROP,EXIT

;C ACCEPT  c-addr +n — +n'  get line from term'l
;  OVER + 1- OVER     — sa ea a
;  BEGIN KEY          — sa ea a c
;  DUP 0D <> WHILE
;           DUP EMIT  — sa ea a c
;           DUP 8 = IF  DROP 1-   >R OVER R> UMAX
;           ELSE  OVER C! 1+ OVER UMIN
;           THEN        — sa ea a
;  REPEAT — sa ea a c
;  DROP NIP SWAP - ;
   head ACCEPT,6,ACCEPT,docolon
            DW OVER,PLUS,ONEMINUS,OVER
ACC1:       DW KEY,DUP,LIT,0DH,NOTEQUAL
            DW QBRANCH,ACC5
            DW DUP,EMIT,DUP,LIT,8
            DW EQUAL,QBRANCH,ACC3
            DW DROP,ONEMINUS,TOR
            DW OVER,RFROM,UMAX
            DW BRANCH,ACC4
ACC3:       DW OVER,CSTORE,ONEPLUS,OVER,UMIN
ACC4:       DW BRANCH,ACC1
ACC5:       DW DROP,NIP,SWOP,MINUS,EXIT

;C TYPE    c-addr +n — type line to term'l
;  ?DUP IF
;           OVER + SWAP DO I C@ EMIT LOOP
;  ELSE DROP THEN ;
   head TYPE,4,TYPE,docolon
            DW QDUP,QBRANCH,TYP4
            DW OVER,PLUS,SWOP,XDO
TYP3:       DW II,CFETCH,EMIT,XLOOP,TYP3
            DW BRANCH,TYP5
TYP4:       DW DROP
TYP5:       DW EXIT

;Z (S")      — c-addr u   run-time code for S"
;  R> COUNT 2DUP + ALIGN >R  ;
   head XSQUOTE,4,(S"),docolon
            DW RFROM,COUNT,TWODUP
            DW PLUS,ALIGN,TOR
            DW EXIT

;C S"       —            compile in-line string
;  COMPILE (S")  [ HEX ]
;  22 WORD C@ 1+ ALIGN ALLOT ; IMMEDIATE
   immed SQUOTE,2,S",docolon
```

```
            DW LIT,XSQUOTE,COMMAXT
            DW LIT,22H,WORD,CFETCH,ONEPLUS
            DW ALIGN,ALLOT,EXIT

;C ."        —           compile string to print
;   POSTPONE S"  POSTPONE TYPE ; IMMEDIATE
    immed DOTQUOTE,2,.",docolon
            DW SQUOTE
            DW LIT,TYPE,COMMAXT
            DW EXIT

; NUMERIC OUTPUT
================================
; Numeric conversion is done l.s.digit first, so
; the output buffer is built backwards in memory.

; Some double-precision arithmetic operators are
; needed to implement ANSI numeric conversion.

;Z UD/MOD   ud1 u2 — u3 ud4   32/16->32 divide
;   >R 0 R@ UM/MOD  ROT ROT R> UM/MOD ROT ;
    head UDSLASHMOD,6,UD/MOD,docolon
            DW TOR,LIT,0,RFETCH
            DW UMSLASHMOD,ROT,ROT
            DW RFROM,UMSLASHMOD,ROT,EXIT

;Z UD*      ud1 d2 — ud3          32*16->32 multiply
;   DUP >R UM* DROP  SWAP R> UM* ROT + ;
    head UDSTAR,3,UD*,docolon
            DW DUP,TOR,UMSTAR,DROP
            DW SWOP,RFROM,UMSTAR,ROT,PLUS,EXIT

;C HOLD  char —        add char to output string
;   -1 HP +!  HP @ C! ;
    head HOLD,4,HOLD,docolon
            DW LIT,-1,HP,PLUSSTORE
            DW HP,FETCH,CSTORE,EXIT

;C <#   —  begin numeric conversion
;   PAD HP ! ;          (initialize Hold Pointer)
    head LESSNUM,2,<#,docolon
            DW PAD,HP,STORE,EXIT

;Z >digit   n — c        convert to 0..9A..Z
;   [ HEX ] DUP 9 > 7 AND + 30 + ;
    head TODIGIT,6,>DIGIT,docolon
            DW DUP,LIT,9,GREATER,LIT,7,AND,PLUS
            DW LIT,30H,PLUS,EXIT

;C #         ud1 — ud2 convert 1 digit of output
;   BASE @ UD/MOD ROT >digit HOLD ;
    head NUM,1,#,docolon
            DW BASE,FETCH,UDSLASHMOD
            DW ROT,TODIGIT
            DW HOLD,EXIT

;C #S   ud1 — ud2      convert remaining digits
;   BEGIN # 2DUP OR 0= UNTIL ;
    head NUMS,2,#S,docolon
NUMS1:      DW NUM,TWODUP,OR,ZEROEQUAL
            DW qbranch,NUMS1
            DW EXIT

;C #>   ud1 — c-addr u   end conv., get string
;   2DROP HP @ PAD OVER - ;
    head NUMGREATER,2,#>,docolon
            DW TWODROP,HP,FETCH
            DW PAD,OVER,MINUS,EXIT

;C SIGN  n —           add minus sign if n<0
;   0< IF 2D HOLD THEN ;
    head SIGN,4,SIGN,docolon
            DW
ZEROLESS,qbranch,SIGN1,LIT,2DH,HOLD
SIGN1:      DW EXIT

;C U.    u —display u unsigned
;   <# 0 #S #> TYPE SPACE ;
    head UDOT,2,U.,docolon
            DW LESSNUM,LIT,0,NUMS
            DW NUMGREATER,TYPE
            DW SPACE,EXIT

;C .        n —          display n signed
;   <# DUP ABS 0 #S ROT SIGN #> TYPE SPACE ;
    head DOT,1,'.',docolon
            DW LESSNUM,DUP,ABS,LIT,0,NUMS
            DW ROT,SIGN,NUMGREATER
            DW TYPE,SPACE,EXIT

;C DECIMAL  —          set number base to decimal
;   10 BASE ! ;
    head DECIMAL,7,DECIMAL,docolon
            DW LIT,10,BASE,STORE,EXIT

;X HEX      —            set number base to hex
;   16 BASE ! ;
    head HEX,3,HEX,docolon
            DW LIT,16,BASE,STORE,EXIT

; DICTIONARY MANAGEMENT
========================

;C HERE    — addr      returns dictionary ptr
;   DP @ ;
    head HERE,4,HERE,docolon
            dw DP,FETCH,EXIT

;C ALLOT   n —          allocate n bytes in dict
;   DP +! ;
    head ALLOT,5,ALLOT,docolon
            dw DP,PLUSSTORE,EXIT

; Note: , and C, are only valid for combined
; Code and Data spaces.

;C ,    x —  append cell to dict
;   HERE ! 1 CELLS ALLOT ;
    head COMMA,1,',',docolon
            dw HERE,STORE,lit,1,CELLS,ALLOT,EXIT

;C C,   char —           append char to dict
;   HERE C! 1 CHARS ALLOT ;
    head CCOMMA,2,'C,',docolon
            dw HERE,CSTORE,lit,1,CHARS,ALLOT,EXIT

; INTERPRETER
==================================
; Note that NFA>LFA, NFA>CFA, IMMED?, and FIND
; are dependent on the structure of the Forth
; header.  This may be common across many CPUs,
; or it may be different.

;C SOURCE   — adr n   current input buffer
;   'SOURCE 2@ ;        length is at lower adrs
    head SOURCE,6,SOURCE,docolon
            DW TICKSOURCE,TWOFETCH,EXIT

;X /STRING  a u n — a+n u-n   trim string
;   ROT OVER + ROT ROT - ;
    head SLASHSTRING,7,/STRING,docolon
            DW ROT,OVER,PLUS,ROT,ROT,MINUS,EXIT

;Z >counted  src n dst —            copy to counted str
;   2DUP C! CHAR+ SWAP CMOVE ;
    head TOCOUNTED,8,>COUNTED,docolon
            DW TWODUP,CSTORE,CHARPLUS
            DW SWOP,CMOVE,EXIT

;C WORD   char — c-addr n   word delim'd by char
;   DUP  SOURCE >IN @ /STRING   — c c adr n
;   DUP >R   ROT SKIP — c adr' n'
;   OVER >R  ROT SCAN             — adr" n"
;   DUP IF CHAR- THEN             skip trailing delim.
;   R> R> ROT -   >IN +! update >IN offset
;   TUCK -   — adr' N
;   HERE >counted      —
;   HERE     — a
;   BL OVER COUNT + C! ;   append trailing blank
    head WORD,4,WORD,docolon
            DW DUP,SOURCE,TOIN
            DW FETCH,SLASHSTRING
            DW DUP,TOR,ROT,SKIP
            DW OVER,TOR,ROT,SCAN
            DW DUP,qbranch,WORD1,ONEMINUS  ; char-
WORD1:      DW RFROM,RFROM,ROT,MINUS
            DW TOIN,PLUSSTORE
            DW TUCK,MINUS
            DW HERE,TOCOUNTED,HERE
            DW BL,OVER,COUNT,PLUS,CSTORE,EXIT

;Z NFA>LFA   nfa — lfa    name adr -> link field
;   3 - ;
    head NFATOLFA,7,NFA>LFA,docolon
            DW LIT,3,MINUS,EXIT

;Z NFA>CFA   nfa — cfa    name adr -> code field
;   COUNT 7F AND + ;  mask off 'smudge' bit
    head NFATOCFA,7,NFA>CFA,docolon
            DW COUNT,LIT,07FH,AND,PLUS,EXIT

;Z IMMED?    nfa — f    fetch immediate flag
;   1- C@ ;  nonzero if immed
    head IMMEDQ,6,IMMED?,docolon
            DW ONEMINUS,CFETCH,EXIT

;C FIND   c-addr — c-addr 0   if not found
;C           xt  1          if immediate
;C           xt -1          if "normal"
;   LATEST @ BEGIN    — a nfa
;           2DUP OVER C@ CHAR+             — a nfa a
nfa n+1
;           S=           — a nfa f
;           DUP IF
;           DROP
;           NFA>LFA @ DUP        — a link link
;           THEN
;   0= UNTIL                — a nfa  OR  a 0
;   DUP IF
;           NIP DUP NFA>CFA      — nfa xt
;           SWAP IMMED?          — xt iflag
;           0= 1 OR     — xt 1/-1
;   THEN ;
    head FIND,4,FIND,docolon
            DW LATEST,FETCH
FIND1:      DW TWODUP,OVER,CFETCH,CHARPLUS
            DW SEQUAL,DUP,qbranch,FIND2
            DW DROP,NFATOLFA,FETCH,DUP
FIND2:      DW ZEROEQUAL,qbranch,FIND1
            DW DUP,qbranch,FIND3
            DW NIP,DUP,NFATOCFA
            DW SWOP,IMMEDQ,ZEROEQUAL,LIT,1,OR
FIND3:      DW EXIT

;C LITERAL  x —          append numeric literal
;   STATE @ IF ['] LIT ,XT , THEN ; IMMEDIATE
; This tests STATE so that it can also be used
; interpretively.  (ANSI doesn't require this.)
    immed LITERAL,7,LITERAL,docolon
            DW STATE,FETCH,qbranch,LITER1
            DW LIT,LIT,COMMAXT,COMMA
LITER1:     DW EXIT

;Z DIGIT?   c — n -1   if c is a valid digit
;Z             — x  0   otherwise
;   [ HEX ] DUP 39 > 100 AND +      silly looking
;   DUP 140 > 107 AND -   30 -       but it works!
;   DUP BASE @ U< ;
    head DIGITQ,6,DIGIT?,docolon
            DW DUP,LIT,39H,GREATER
            DW LIT,100H,AND,PLUS
            DW DUP,LIT,140H,GREATER,LIT,107H,AND
            DW MINUS,LIT,30H,MINUS
            DW DUP,BASE,FETCH,ULESS,EXIT

;Z ?SIGN   adr n — adr' n' f  get optional sign
;Z  advance adr/n if sign; return NZ if negative
;   OVER C@              — adr n c
;   2C - DUP ABS 1 = AND    — +=-1, -=+1, else 0
;   DUP IF 1+             — +=0, -=+2
;           >R 1 /STRING R>         — adr' n' f
;   THEN ;
    head QSIGN,5,?SIGN,docolon
            DW OVER,CFETCH,LIT,2CH,MINUS,DUP,ABS
            DW LIT,1,EQUAL,AND,DUP,qbranch,QSIGN1
            DW ONEPLUS,TOR,LIT,1
            DW SLASHSTRING,RFROM
QSIGN1:     DW EXIT

;C >NUMBER  ud adr u — ud' adr' u'
;C             convert string to number
;   BEGIN
;   DUP WHILE
;           OVER C@ DIGIT?
;           0= IF DROP EXIT THEN
;           >R 2SWAP BASE @ UD*
;           R> M+ 2SWAP
;           1 /STRING
;   REPEAT ;
    head TONUMBER,7,>NUMBER,docolon
TONUM1:     DW DUP,qbranch,TONUM3
            DW OVER,CFETCH,DIGITQ
            DW ZEROEQUAL,qbranch
            DW TONUM2,DROP,EXIT
TONUM2:     DW TOR,TWOSWAP,BASE,FETCH,UDSTAR
            DW RFROM,MPLUS,TWOSWAP
            DW LIT,1,SLASHSTRING,branch,TONUM1
TONUM3:     DW EXIT

;Z ?NUMBER  c-addr — n -1         string->number
;Z             — c-addr 0  if convert error
;   DUP  0 0 ROT COUNT            — ca ud adr n
;   ?SIGN >R  >NUMBER             — ca ud adr' n'
;   IF   R> 2DROP 2DROP 0   — ca 0   (error)
;   ELSE 2DROP NIP R>
;           IF NEGATE THEN  -1  — n -1   (ok)
;   THEN ;
    head QNUMBER,7,?NUMBER,docolon
            DW DUP,LIT,0,DUP,ROT,COUNT
            DW QSIGN,TOR,TONUMBER,qbranch,QNUM1
            DW RFROM,TWODROP,TWODROP,LIT,0
            DW branch,QNUM3
QNUM1:      DW TWODROP,NIP,RFROM
```

```
            DW qbranch,QNUM2,NEGATE
QNUM2:      DW LIT,-1
QNUM3:      DW EXIT

;Z INTERPRET    i*x c-addr u -- j*x
;Z              interpret given buffer
; This is a common factor of EVALUATE and QUIT.
; ref. dpANS-6, 3.4 The Forth Text Interpreter
;  'SOURCE 2!  0 >IN !
;  BEGIN
;  BL WORD DUP C@ WHILE         -- textadr
;             FIND         -- a 0/1/-1
;             ?DUP IF      -- xt 1/-1
;             1+ STATE @ 0= OR    immed or interp?
;             IF EXECUTE ELSE ,XT THEN
;             ELSE         -- textadr
;             ?NUMBER
;             IF POSTPONE LITERAL              converted
ok
;             ELSE COUNT TYPE 3F EMIT CR ABORT  err
;             THEN
;             THEN
;  REPEAT DROP ;
   head INTERPRET,9,INTERPRET,docolon
            DW TICKSOURCE,TWOSTORE
            DW LIT,0,TOIN,STORE
INTER1:     DW BL,WORD,DUP,CFETCH,qbranch,INTER9
            DW FIND,QDUP,qbranch,INTER4
            DW ONEPLUS,STATE
            DW FETCH,ZEROEQUAL,OR
            DW qbranch,INTER2
            DW EXECUTE,branch,INTER3
INTER2:     DW COMMAXT
INTER3:     DW branch,INTER8
INTER4:     DW QNUMBER,qbranch,INTER5
            DW LITERAL,branch,INTER6
INTER5:     DW COUNT,TYPE,LIT,3FH,EMIT,CR,ABORT
INTER6:
INTER8:     DW branch,INTER1
INTER9:     DW DROP,EXIT

;C EVALUATE  i*x c-addr u -- j*x  interprt string
;  'SOURCE 2@ >R >R  >IN @ >R
;  INTERPRET
;  R> >IN !  R> R> 'SOURCE 2! ;
   head EVALUATE,8,EVALUATE,docolon
            DW TICKSOURCE,TWOFETCH,TOR,TOR
            DW TOIN,FETCH,TOR,INTERPRET
            DW RFROM,TOIN,STORE,RFROM,RFROM
            DW TICKSOURCE,TWOSTORE,EXIT

;C QUIT     --  R: i*x --    interpret from kbd
;  L0 LP !  R0 RP!   0 STATE !
;  BEGIN
;           TIB DUP TIBSIZE ACCEPT  SPACE
;           INTERPRET
;           STATE @ 0= IF CR ." OK" THEN
;  AGAIN ;
   head QUIT,4,QUIT,docolon
            DW L0,LP,STORE
            DW R0,RPSTORE,LIT,0,STATE,STORE
QUIT1:      DW TIB,DUP,TIBSIZE,ACCEPT
            DW SPACE,INTERPRET
            DW STATE,FETCH,ZEROEQUAL
            DW qbranch,QUIT2
            DW CR,XSQUOTE
            DB 3,'ok '
            DW TYPE
QUIT2:      DW branch,QUIT1

;C ABORT    i*x --  R: j*x --   clear stk & QUIT
;  S0 SP!  QUIT ;
   head ABORT,5,ABORT,docolon
            DW S0,SPSTORE,QUIT   ; QUIT never returns

;Z ?ABORT   f c-addr u --             abort & print msg
;  ROT IF TYPE ABORT THEN 2DROP ;
   head QABORT,6,?ABORT,docolon
            DW ROT,qbranch,QABO1,TYPE,ABORT
QABO1:      DW TWODROP,EXIT

;C ABORT"  i*x 0  -- i*x   R: j*x -- j*x  x1=0
;C         i*x x1 --      R: j*x --     x1<>0
;  POSTPONE S" POSTPONE ?ABORT ; IMMEDIATE
   immed ABORTQUOTE,6,ABORT",docolon
            DW SQUOTE
            DW LIT,QABORT,COMMAXT
            DW EXIT

;C '    -- xt  find word in dictionary
;  BL WORD FIND
;  0= ABORT" ?" ;
;   head TICK,1,',docolon
            DW link      ; must expand
            DB 0         ; manually
link   DEFL $            ; because of
            DB 1,27h    ; tick character
TICK:  call docolon
            DW BL,WORD,FIND,ZEROEQUAL,XSQUOTE
            DB 1,'?'
            DW QABORT,EXIT

;C CHAR   -- char        parse ASCII character
;  BL WORD 1+ C@ ;
   head CHAR,4,CHAR,docolon
            DW BL,WORD,ONEPLUS,CFETCH,EXIT

;C [CHAR]  --           compile character literal
;  CHAR  ['] LIT ,XT  , ; IMMEDIATE
   immed BRACCHAR,6,[CHAR],docolon
            DW CHAR
            DW LIT,LIT,COMMAXT
            DW COMMA,EXIT

;C (    --     skip input until )
;  [ HEX ] 29 WORD DROP ; IMMEDIATE
   immed PAREN,1,(,docolon
            DW LIT,29H,WORD,DROP,EXIT

; COMPILER
=====================================

;C CREATE  --           create an empty definition
;  LATEST @ , 0 C,      link & immed field
;  HERE LATEST !        new "latest" link
;  BL WORD C@ 1+ ALLOT          name field
;  docreate ,CF         code field
   head CREATE,6,CREATE,docolon
            DW LATEST,FETCH,COMMA,LIT,0,CCOMMA
            DW HERE,LATEST,STORE
            DW BL,WORD,CFETCH,ONEPLUS,ALLOT
            DW LIT,docreate,COMMACF,EXIT

;Z (DOES>)  --          run-time action of DOES>
;  R>      adrs of headless DOES> def'n
;  LATEST @ NFA>CFA    code field to fix up
;  !CF ;
   head XDOES,7,(DOES>),docolon
            DW RFROM,LATEST,FETCH
            DW NFATOCFA,STORECF
            DW EXIT

;C DOES>   --           change action of latest def'n
;  COMPILE (DOES>)
;  dodoes ,CF ; IMMEDIATE
   immed DOES,5,DOES>,docolon
            DW LIT,XDOES,COMMAXT
            DW LIT,dodoes,COMMACF,EXIT

;C RECURSE  --          recurse current definition
;  LATEST @ NFA>CFA ,XT ; IMMEDIATE
   immed RECURSE,7,RECURSE,docolon
            DW LATEST,FETCH,NFATOCFA
            DW COMMAXT,EXIT

;C [         --             enter interpretive state
;  0 STATE ! ; IMMEDIATE
   immed LEFTBRACKET,1,[,docolon
            DW LIT,0,STATE,STORE,EXIT

;C ]         --             enter compiling state
;  -1 STATE ! ;
   head RIGHTBRACKET,1,],docolon
            DW LIT,-1,STATE,STORE,EXIT

;Z HIDE      --             "hide" latest definition
;  LATEST @ DUP C@ 80 OR SWAP C! ;
   head HIDE,4,HIDE,docolon
            DW LATEST,FETCH,DUP
            DW CFETCH,LIT,80H,OR
            DW SWOP,CSTORE,EXIT

;Z REVEAL   --             "reveal" latest definition
;  LATEST @ DUP C@ 7F AND SWAP C! ;
   head REVEAL,6,REVEAL,docolon
            DW LATEST,FETCH,DUP
            DW CFETCH,LIT,7FH,AND
            DW SWOP,CSTORE,EXIT

;C IMMEDIATE   --   make last def'n immediate
;  1 LATEST @ 1- C! ;   set immediate flag
   head IMMEDIATE,9,IMMEDIATE,docolon
            DW LIT,1,LATEST,FETCH
            DW ONEMINUS,CSTORE
            DW EXIT

;C :         --             begin a colon definition
;  CREATE HIDE ] !COLON ;
   head COLON,1,:,docode
            CALL docolon   ; code fwd ref explicitly
            DW CREATE,HIDE,RIGHTBRACKET
            DW STORCOLON
            DW EXIT

;C ;
;  REVEAL  ,EXIT
;  POSTPONE [  ; IMMEDIATE
   immed SEMICOLON,1,';',docolon
            DW REVEAL,CEXIT
            DW LEFTBRACKET,EXIT

;C ['] --      find word & compile as literal
;  '  ['] LIT ,XT  , ; IMMEDIATE
; When encountered in a colon definition, the
; phrase  ['] xxx  will cause   LIT,xxt  to be
; compiled into the colon definition (where
; (where xxt is the execution token of word xxx).
; When the colon definition executes, xxt will
; be put on the stack.  (All xt's are one cell.)
;   immed BRACTICK,3,['],docolon
            DW link      ; must expand
            DB 1         ; manually
link   DEFL $            ; because of
            DB 3,5Bh,27h,5Dh      ; tick character
BRACTICK: call docolon
            DW TICK   ; get xt of 'xxx'
            DW LIT,LIT,COMMAXT    ; append LIT action
            DW COMMA,EXIT         ; append xt literal

;C POSTPONE  --   postpone compile action of word
;  BL WORD FIND
;  DUP 0= ABORT" ?"
;  0< IF   -- xt  non immed: add code to current
;            def'n to compile xt later.
;            ['] LIT ,XT  , add "LIT,xt,COMMAXT"
;            ['] ,XT ,XT   to current definition
;  ELSE ,XT              immed: compile into cur. def'n
;  THEN ; IMMEDIATE
   immed POSTPONE,8,POSTPONE,docolon
            DW BL,WORD,FIND,DUP
            DW ZEROEQUAL,XSQUOTE
            DB 1,'?'
            DW QABORT,ZEROLESS,qbranch,POST1
            DW LIT,LIT,COMMAXT,COMMA
            DW LIT,COMMAXT,COMMAXT,branch,POST2
POST1:      DW COMMAXT
POST2:      DW EXIT

;Z COMPILE  --   append inline execution token
;  R> DUP CELL+ >R @ ,XT ;
; The phrase ['] xxx ,XT appears so often that
; this word was created to combine the actions
; of LIT and ,XT.  It takes an inline literal
; execution token and appends it to the dict.
;   head COMPILE,7,COMPILE,docolon
;           DW RFROM,DUP,CELLPLUS,TOR
;           DW FETCH,COMMAXT,EXIT
; N.B.: not used in the current implementation

; CONTROL STRUCTURES
==========================

;C IF        -- adrs   conditional forward branch
;  ['] qbranch ,BRANCH  HERE DUP ,DEST ;
;  IMMEDIATE
   immed IF,2,IF,docolon
            DW LIT,qbranch,COMMABRANCH
            DW HERE,DUP,COMMADEST,EXIT

;C THEN   adrs --       resolve forward branch
;  HERE SWAP !DEST ; IMMEDIATE
   immed THEN,4,THEN,docolon
            DW HERE,SWOP,STOREDEST,EXIT

;C ELSE   adrs1 -- adrs2   branch for IF..ELSE
;  ['] branch ,BRANCH  HERE DUP ,DEST
;  SWAP  POSTPONE THEN ; IMMEDIATE
   immed ELSE,4,ELSE,docolon
            DW LIT,branch,COMMABRANCH
            DW HERE,DUP,COMMADEST
            DW SWOP,THEN,EXIT

;C BEGIN   -- adrs      target for bwd. branch
;  HERE ; IMMEDIATE
   immed BEGIN,5,BEGIN,docode
            jp HERE

;C UNTIL   adrs --   conditional backward branch
;  ['] qbranch ,BRANCH  ,DEST ; IMMEDIATE
;  conditional backward branch
   immed UNTIL,5,UNTIL,docolon
            DW LIT,qbranch,COMMABRANCH
            DW COMMADEST,EXIT
```

```
;X AGAIN    adrs —       uncond'l backward branch
;   ['] branch ,BRANCH  ,DEST ; IMMEDIATE
;   unconditional backward branch
    immed AGAIN,5,AGAIN,docolon
            DW LIT,branch,COMMABRANCH
            DW COMMADEST,EXIT

;C WHILE    — adrs      branch for WHILE loop
;   POSTPONE IF ; IMMEDIATE
    immed WHILE,5,WHILE,docode
            jp IF

;C REPEAT   adrs1 adrs2 —           resolve WHILE loop
;   SWAP POSTPONE AGAIN POSTPONE THEN ;
IMMEDIATE
    immed REPEAT,6,REPEAT,docolon
            DW SWOP,AGAIN,THEN,EXIT

;Z >L   x —   L: — x        move to leave stack
;   CELL LP +!  LP @ ! ;(L stack grows up)
    head TOL,2,>L,docolon
            DW CELL,LP,PLUSSTORE
            DW LP,FETCH,STORE,EXIT

;Z L>    — x   L: x —       move from leave stack
;   LP @ @  CELL NEGATE LP +! ;
    head LFROM,2,L>,docolon
            DW LP,FETCH,FETCH
            DW CELL,NEGATE,LP,PLUSSTORE,EXIT

;C DO       — adrs   L: — 0
;   ['] xdo ,BRANCH   HERE  target for bwd branch
;   0 >L ; IMMEDIATE    marker for LEAVEs
    immed DO,2,DO,docolon
            DW LIT,xdo,COMMABRANCH,HERE
            DW LIT,0,TOL,EXIT

;Z ENDLOOP   adrs xt —   L: 0 a1 a2 .. aN —
;   ,BRANCH  ,DEST     backward loop
;   BEGIN L> ?DUP WHILE POSTPONE THEN REPEAT ;
;             resolve LEAVEs
; This is a common factor of LOOP and +LOOP.
    head ENDLOOP,7,ENDLOOP,docolon
            DW COMMABRANCH,COMMADEST
LOOP1:      DW LFROM,QDUP,qbranch,LOOP2
            DW THEN,branch,LOOP1
LOOP2:      DW EXIT

;C LOOP    adrs —   L: 0 a1 a2 .. aN —
;   ['] xloop ENDLOOP ;  IMMEDIATE
    immed LOOP,4,LOOP,docolon
            DW LIT,xloop,ENDLOOP,EXIT

;C +LOOP   adrs —   L: 0 a1 a2 .. aN —
;   ['] xplusloop ENDLOOP ;  IMMEDIATE
    immed PLUSLOOP,5,+LOOP,docolon
            DW LIT,xplusloop,ENDLOOP,EXIT

;C LEAVE    —    L: — adrs
;   ['] UNLOOP ,XT
;   ['] branch ,BRANCH   HERE DUP ,DEST  >L
;   ; IMMEDIATE         unconditional forward branch
    immed LEAVE,5,LEAVE,docolon
            DW LIT,unloop,COMMAXT
            DW LIT,branch,COMMABRANCH
            DW HERE,DUP,COMMADEST,TOL,EXIT

; OTHER OPERATIONS
==============================

;X WITHIN   n1|u1 n2|u2 n3|u3 — f   n2<=n1<n3?
;  OVER - >R - R> U< ;  per ANS document
    head WITHIN,6,WITHIN,docolon
            DW OVER,MINUS,TOR,MINUS
            DW RFROM,ULESS,EXIT

;C MOVE    addr1 addr2 u —          smart move
;           VERSION FOR 1 ADDRESS UNIT = 1 CHAR
;  >R 2DUP SWAP DUP R@ +         — ... dst src src+n
;  WITHIN IF  R> CMOVE>           src <= dst < src+n
;           ELSE  R> CMOVE  THEN ;          otherwise
    head MOVE,4,MOVE,docolon
            DW TOR,TWODUP,SWOP
            DW DUP,RFETCH,PLUS
            DW WITHIN,qbranch,MOVE1
            DW RFROM,CMOVEUP,branch,MOVE2
MOVE1:      DW RFROM,CMOVE
MOVE2:      DW EXIT

;C DEPTH    — +n         number of items on stack
;   SP@ S0 SWAP - 2/ ;   16-BIT VERSION!
    head DEPTH,5,DEPTH,docolon
            DW SPFETCH,S0,SWOP
            DW MINUS,TWOSLASH,EXIT

;C ENVIRONMENT?  c-addr u — false   system query
;                  — i*x true
;   2DROP 0 ;           the minimal definition!
    head ENVIRONMENTQ,12,ENVIRONMENT?,docolon
            DW TWODROP,LIT,0,EXIT

; UTILITY WORDS AND STARTUP
=====================

;X WORDS    —          list all words in dict.
;   LATEST @ BEGIN
;           DUP COUNT TYPE SPACE
;           NFA>LFA @
;   DUP 0= UNTIL
;   DROP ;
    head WORDS,5,WORDS,docolon
            DW LATEST,FETCH
WDS1:       DW DUP,COUNT,TYPE,SPACE
            DW NFATOLFA,FETCH
            DW DUP,ZEROEQUAL,qbranch,WDS1
            DW DROP,EXIT

;X .S       —           print stack contents
;   SP@ S0 - IF
;           SP@ S0 2 - DO I @ U. -2 +LOOP
;   THEN ;
    head DOTS,2,.S,docolon
            DW SPFETCH,S0,MINUS,qbranch,DOTS2
            DW SPFETCH,S0,LIT,2,MINUS,XDO
DOTS1:      DW II,FETCH,UDOT,LIT
            DW -2,XPLUSLOOP,DOTS1
DOTS2:      DW EXIT

;Z COLD     —           cold start Forth system
;   UINIT U0 #INIT CMOVE          init user area
;   TIB COUNT INTERPRET           interpret CP/M cmd
;   ." Z80 CamelForth etc."
;   ABORT ;
    head COLD,4,COLD,docolon
            DW UINIT,U0,NINIT,CMOVE
            DW TIB,COUNT,INTERPRET
            DW XSQUOTE
            DB 35,'Z80 CamelForth v1.0  19 Aug 1994'
            DB 0dh,0ah
            DW TYPE,ABORT        ; ABORT never returns
```

```
; LISTING 3.
;
; ===============================================
; CamelForth for the Zilog Z80
; (c) 1994 Bradford J. Rodriguez
; Permission is granted to freely copy, modify,
; and distribute this program for personal or
; educational use.  Commercial inquiries should
; be directed to the author at 221 King St. E.,
; #32, Hamilton, Ontario L8N 1B5 Canada
;
; CAMEL80D.AZM: CPU and Model Dependencies
;   Source code is for the Z80MR macro assembler.
;   Forth words are documented as follows:
;*   NAME   stack — stack    description
;   Word names in upper case are from the ANS
;   Forth Core word set.  Names in lower case are
;   "internal" implementation words & extensions.
;
; Direct-Threaded Forth model for Zilog Z80
;   cell size is   16 bits (2 bytes)
;   char size is    8 bits (1 byte)
;   address unit is 8 bits (1 byte), i.e.,
;               addresses are byte-aligned.
;
; ===============================================

; ALIGNMENT AND PORTABILITY OPERATORS
===========
; Many of these are synonyms for other words,
; and so are defined as CODE words.

;C ALIGN    —           align HERE
    head ALIGN,5,ALIGN,docode
noop:  next

;C ALIGNED  addr — a-addr         align given addr
    head ALIGNED,7,ALIGNED,docode
            jr noop

;Z CELL     — n          size of one cell
    head CELL,4,CELL,docon
            dw 2

;C CELL+    a-addr1 — a-addr2      add cell size
;   2 + ;
    head CELLPLUS,5,CELL+,docode
            inc bc
            inc bc
            next

;C CELLS    n1 — n2    cells->adrs units
    head CELLS,5,CELLS,docode
            jp twostar

;C CHAR+    c-addr1 — c-addr2   add char size
    head CHARPLUS,5,CHAR+,docode
            jp oneplus

;C CHARS    n1 — n2    chars->adrs units
    head CHARS,5,CHARS,docode
            jr noop

;C >BODY    xt — a-addr           adrs of param field
;   3 + ;        Z80 (3 byte CALL)
    head TOBODY,5,>BODY,docolon
            DW LIT,3,PLUS,EXIT

;X COMPILE,  xt —       append execution token
; I called this word ,XT before I discovered that
; it is defined in the ANSI standard as COMPILE,.
; On a DTC Forth this simply appends xt (like , )
; but on an STC Forth this must append 'CALL xt'.
    head COMMAXT,8,'COMPILE,',docode
            jp COMMA

;Z !CF    adrs cfa —   set code action of a word
;   0CD OVER C!          store 'CALL adrs' instr
;   1+ ! ;       Z80 VERSION
; Depending on the implementation this could
; append CALL adrs or JUMP adrs.
    head STORECF,3,!CF,docolon
            DW LIT,0CDH,OVER,CSTORE
            DW ONEPLUS,STORE,EXIT

;Z ,CF    adrs —         append a code field
;   HERE !CF 3 ALLOT ;  Z80 VERSION (3 bytes)
    head COMMACF,3,',CF',docolon
            DW HERE,STORECF,LIT,3,ALLOT,EXIT

;Z !COLON   —           change code field to docolon
;   -3 ALLOT docolon-adrs ,CF ;
; This should be used immediately after CREATE.
; This is made a distinct word, because on an STC
; Forth, colon definitions have no code field.
    head STORCOLON,6,'!COLON',docolon
            DW LIT,-3,ALLOT
            DW LIT,docolon,COMMACF,EXIT

;Z ,EXIT    —           append hi-level EXIT action
;   ['] EXIT ,XT ;
; This is made a distinct word, because on an STC
; Forth, it appends a RET instruction, not an xt.
    head CEXIT,5,',EXIT',docolon
            DW LIT,EXIT,COMMAXT,EXIT

; CONTROL STRUCTURES
============================
; These words allow Forth control structure words
; to be defined portably.

;Z ,BRANCH   xt —    append a branch instruction
; xt is the branch operator to use, e.g. qbranch
; or (loop).  It does NOT append the destination
; address.  On the Z80 this is equivalent to ,XT.
    head COMMABRANCH,7,',BRANCH',docode
            jp COMMA

;Z ,DEST   dest —        append a branch address
; This appends the given destination address to
; the branch instruction.  On the Z80 this is ','
; ...other CPUs may use relative addressing.
    head COMMADEST,5,',DEST',docode
            jp COMMA

;Z !DEST   dest adrs —    change a branch dest'n
; Changes the destination address found at 'adrs'
; to the given 'dest'.  On the Z80 this is '!'
; ...other CPUs may need relative addressing.
    head STOREDEST,5,'!DEST',docode
            jp STORE

; HEADER STRUCTURE
==============================
; The structure of the Forth dictionary headers
; (name, link, immediate flag, and "smudge" bit)
; does not necessarily differ across CPUs.  This
; structure is not easily factored into distinct
; "portable" words; instead, it is implicit in
; the definitions of FIND and CREATE, and also in
; NFA>LFA, NFA>CFA, IMMED?, IMMEDIATE, HIDE, and
; REVEAL.  These words must be (substantially)
; rewritten if either the header structure or its
; inherent assumptions are changed.        (end listings.)
```

Regular Feature

Group Reviews

SUPPORT GROUPS FOR THE CLASSICS

TCJ Staff Contacts

TCJ Editor: Bill D. Kibler, PO Box 535, Lincoln, CA 95648, (916)645-1670, GEnie: B.Kibler, CompuServe: 71563,2243, E-mail: B.Kibler@Genie.geis.com.

Z-System Support: Jay Sage,1435 Centre St. Newton Centre, MA 02159-2469, (617)965-3552, BBS: (617)965-7259; E-mail: Sage@ll.mit.edu. Also sells Z-System software.

32Bit Support: Rick Rodman, BBS:(703)330-9049 (eves), E-mail: rickr@virtech.vti.com.

Kaypro Support: Charles Stafford, 4000 Norris Ave., Sacramento, CA 95821, (916)483-0312 (eves). Also sells Kaypro upgrades, see ad inside back cover. CompuServe 73664,2470 (73664.2470@cis).

S-100 Support: Herb Johnson, CN 5256 #105, Princeton, NJ 08543, (609)771-1503. Also sells used S-100 boards and systems, see inside back cover.

6800/6809 Support: Ronald Anderson, 3540 Sturbridge Ct., Ann Arbor, MI 48105.

Regular Contributors:
Dave Baldwin, Voice/FAX (916)722-3877, or DIBs BBS (916) 722-5799 (use "computer", "journal", pswd "subscriber" as log on), Internet dibald@netcom.com, CompuServe 70403,2444.

Brad Rodriguez,Box 77, McMaster Univ., 1280 Main St. West, Hamilton, ONT, L8S 1C0, Canada, Genie: B.Rodriguez2, E-mail: b.rodriguez2@genie.geis.com.

Frank Sergeant, 809 W. San Antonio St., San Marcos, TX 78666, E-mail: fs07675@academia.swt.edu.

Tilmann Reh, Germany, E-mail: tilmann.reh@hrz.uni-siegen.d400.de. Has many programs for CP/M+ and is active with Z180/280 ECB bus/Modular/Embedded computers. USA contact Jay Sage.

Helmut Jungkunz, Munich, Germany, ZNODE #51, 8N1, 300-14.4, +49.89.9614574, or CompuServe 100024,1545.

Ron Mitchell, Apt 1107, 210 Gloucester St., Ottawa Ontario, Canada, K2P 2K4. GEnie as R.Mitchell31, or CompuServe 70323,2267.

USER GROUPS

Connecticut CP/M Users Group, contact Stephen Griswold, PO Box 74, Canton CT 06019-0074, BBS: (203)665-1100. Sponsors East Coast Z-fests.

Sacramento Microcomputer Users Group, PO Box 161513, Sacramento, CA 95816-1513, BBS: (916)372-3646. Publishes newsletter, $15.00 membership, normal meeting is first Thursday at SMUD 6201 S st., Sacramento CA.

CAPDUG: The Capital Area Public Domain Users Group, Newsletter $20, Al Siegel Associates, Inc., PO Box 34667, Betherda MD 20827. BBS (301) 292-7955.

NOVAOUG: The Northern Virginia Osborne Users Group, Newsletter $12, Robert L. Crities, 7512 Fairwood Lane, Falls Church, VA 22046. Info (703) 534-1186, BBS use CAPDUG's.

The Windsor Bulletin Board Users' Group: England, Contact Rodney Hannis, 34 Falmouth Road, Reading, RG2 8QR, or Mark Minting, 94 [illegible]ndley Common, Lakenheath, Brandon, Suffolk, IP27 9BZ, Phone 0842-860469 (also sells NZCOM/Z3PLUS).

L.I.S.T.: Long Island Sinclair and Timex support group, contact Harvey Rait, 5 Peri Lane, Valley Stream, NY 11581.

Coleco ADAM:
ADAM-Link User's Group, Salt Lake City, Utah, BBS: (801)484-5114. Supporting Coleco ADAM machines, with Newsletter and BBS.

Adam International Media, Adam's House, Route 2, Box 2756, 1829-1 County Rd. 130, Pearland TX 77581-9503, (713)482-5040. Contact Terry R. Fowler for information.

AUGER, Emerald Coast ADAM Users Group, PO Box 4934, Fort Walton Beach FL 32549-4934, (904)244-1516. Contact Norman J. Deere, treasurer and editor for pricing and newsletter information.

MOAUG, Metro Orlando Adam Users Group, Contact James Poulin, 1146 Manatee Dr. Rockledge FL 32955, (407)631-0958.

Metro Toronto Adam Group, Box 165, 260 Adelaide St. E., Toronto, ONT M5A 1N0, Canada, (416)424-1352.

Omaha ADAM Users Club, Contact Norman R. Castro, 809 W. 33rd Ave. Bellevue NE 68005, (402)291-4405. Suppose to be oldest ADAM group.

Vancouver Island Senior ADAMphiles, ADVISA newsletter by David Cobley, 17885 Berwick Rd. Qualicum Beach, B.C., Canada V9K 1N7, (604)752-1984.

Northern Illiana ADAMS User's Group, 9389 Bay Colony Dr. #3E, Des Plaines IL 60016, (708)296-0675.

OS-9 Support:
San Diego OS-9 Users Group, Contact Warren Hrach (619)221-8246, BBS: (619)224-4878.

Atari Support:
ACCESS, PO Box 1354, Sacramento, CA 95812, Contact Bob Drews (916)423-1573. Meets first Thurdays at SMUD 59Th St. (ed. bldg.).

Forth Support:
Forth Interest Group, PO Box 2154, Oakland CA 94621 510-89-FORTH. International support of the Forth language. Contact for list of local chapters.

OTHER PUBLICATIONS

The Z-Letter, supporting Z-System and CP/M users. David A.J. McGlone, Lambda Software Publishing, 149 West Hillard Lane, Eugene, OR 97404-3057, (503)688-3563. Bi-Monthly user oriented newsletter (20 pages+). Also sells CP/M Boot disks, software.

The Analytical Engine, by the Computer History Association of California, 1001 Elm Ct. El Cerrito, CA 94530-2602. A ASCII text file distributed by Internet, issue #1 was July 1993. E-mail: kcrosby@crayola.win.net.

Z-100 LifeLine, Steven W. Vagts, 2409 Riddick Rd. Elizabeth City, NC 27909, (919)338-8302. Publication for Z-100 (a S-100 machine).

The Staunch 8/89'er, Kirk L. Thompson editor, PO Box 548, West Branch IA 52358, (319)643-7136. $15/yr(US) publication for H-8/89s.

The SEBHC Journal, Leonard Geisler, 895 Starwick Dr., Ann Arbor MI 48105, (313)662-0750. Magazine of the Society of Eight-Bit Heath computerists, H-8 and H-89 support.

Sanyo PC Hackers Newsletter, Victor R. Frank editor, 12450 Skyline Blvd. Woodside, CA 94062-4541, (415)851-7031. Support for orphaned Sanyo computers and software.

the world of 68' micros, by FARNA Systems, PO Box 321, Warner Robins, GA 31099-0321. E-mail: dsrtfox@delphi.com. New magazine for support of old CoCo's and other 68xx(x) systems.

Amstrad PCW SIG, newsletter by Al Warsh, 2751 Reche Cyn Rd. #93, Colton, CA 92324. $9 for 6 bi-monthly newsletters on Amstrad CP/M machines.

Historically Brewed, A publication of the Historical Computer Society. Bimonthly at $18 a year. HCS, 10928 Ted Williams PL., El Paso, TX 79934. Editor David Greelish. Computer History and more.

Other Support Businesses

Hal Bower writes, sells, and supports B/PBios for Ampro, SB180, and YASBEC. $69.95. Hal Bower, 7914 Redglobe Ct., Severn MD 21144-1048, (410)551-5922.

Sydex, PO Box 5700, Eugene OR 97405, (503)683-6033. Sells several CP/M programs for use with PC Clones ('22Disk' format/copies CP/M disks using PC files system).

Elliam Associates, PO Box 2664, Atascadero CA 93423, (805)466-8440. Sells CP/M user group disks and Amstrad PCW products. See ad inside back cover.

Discus Distribution Services, Inc. sells CP/M for $150, CBASIC $600, Fortran-77 $350, Pascal/MT+ $600. 8020 San Miguel Canyon Rd., Salinas CA 93907, (408)663-6966.

Microcomputer Mail-Order Library of books, manuals, and periodicals in general and H/Zenith in particular. Borrow items for small fees. Contact Lee Hart, 4209 France Ave. North, Robbinsdale MN 55422, (612)533-3226.

Star Technology, 900 Road 170, Carbondale CO, 81623. Epson QX-10 support and repairs. New units also avialble.

Star-K Software Systems Corp. PO Box 209, Mt. Kisco, NY 10549, (914)241-0287, BBS: (914)241-3307. 6809/68000 operating system and software. Some educational products, call for catalog.

Peripheral Technology, 1250 E. Piedmont Rd., Marietta, GA 30067, (404)973-2156. 6809/68000 single board system. 68K ISA bus compatible system. See inside front cover.

Hazelwood Computers, RR#1, Box 36, Hwy 94@Bluffton, Rhineland, MO 65069, (314)236-4372. Some SS-50 6809 boards and new 68000 systems.

AAA Chicago Computers, Jerry Koppel, (708)202-0150. SS-50 6809 boards and systems. Very limited quanity, call for information.

MicroSolutions Computer Products, 132 W. Lincoln Hwy, DeKalb, IL 60115, (815)756-3411. Make disk copying program for CP/M systems, that runs on CP/M sytems, UNIFROM Format-translation. Also PC/Z80 CompatiCard and UniDos products.

GIMIX/OS-9, GMX, 3223 Arnold Lane, Northbrook, IL 60062, (800)559-0909, (708)559-0909, FAX (708)559-0942. Repair and support of new and old 6800/6809/68K/SS-50 systems.

n/SYSTEMS, Terry Hazen, 21460 Bear Creek Rd, Los Gatos CA 95030-9429, (408)354-7188, sells and supports the MDISK add-on RAM disk for the Ampro LB. PCB $29, assembled PCB $129, includes driver software, manual.

Corvatek, 561 N.W. Van Buren St. Corvallis OR 97330, (503)752-4833. PC style to serial keyboard adapter for Xerox, Kaypros, Franklin, Apples, $129. Other models supported.

Morgan, Thielmann & Associates services NON-PC compatible computers including CP/M as well as clones. Call Jerry Davis for more information (408) 972-1965.

Jim S. Thale Jr., 1150 Somerset Ave., Deerfield IL 60015-2944, (708)948-5731. Sells I/O board for YASBEC. Adds HD drives, 2 serial, 2 parallel ports. Partial kit $150, complete kit $210.

Trio Comapny of Cheektowaga, Ltd., PO Box 594, Cheektowaga NY 14225, (716)892-9630. Sells CP/M (& PC) packages: InfoStar 1.5 ($160); SuperSort 1.6 ($130), and WordStar 4.0 ($130).

Parts is Parts, Mike Zinkow, 137 Barkley Ave., Clifton NJ 07011-3244, (201)340-7333. Supports Zenith Z-100 with parts and service.

READER TO READER from page 6

napolis, MD. He wanted me to go over it to see if the hardware looked usable and then see if we could get it working with system disks.

After opening it up, I found two processor chips, an 8086 and a Z80. There were parallel and a serial port, 256 k of RAM and two 5.25 full height drives. The case appeared to be an upscale copy of the Kaypro design. Applying power with no boot disk revealed a decent little monitor program in ROM. Playing with the monitor, I determined that the RAM and both floppy drives were good, that the video circuits and keyboard all functioned, as did the 8086 processor. Now to find boot disks! One email call for help drew a note from Don Maslin saying that he could supply the system disks for the cost of media and handling! Could this be REAL? I sent a check and within the week had the magic media to make the little jewel come alive! There were two versions of MS-DOS, version 1.0 and 2.0, as well as CP/M 2.2 for the Z80 side. I booted each disk and made backup copies of the disks (on which I had just installed write-protect tabs). That done, I played a bit with each system to see how it worked.

The MS-DOS disks offered the basic tools of that OS and all seemed to work. I do not have an MS-DOS machine unless you count the Z-100 which my son runs in CP/M. I never pursued that side of the Z-100 since most of the software was written for PC-DOS which was just enough different that the Z-100 would scream 'WILD INTERRUPT' and quit. Therefore, my evaluation of the 8086 side of this machine is limited to just a quick cruise through the OS utilities. Version 2.0 came with several little utilities that I found nice. The first was called OPTION.EXE and allows one to configure the machine hardware. This configuration is saved in Non-Volatile RAM and will remain in force until changed by the program or over-ridden by an application. (Nice applications should return the system to the way they found it.) This was important to me since the configuration of the hardware also impacted the how the system worked when under the Z-80 processor.

Opt. A —	Set the amount of memory: up to 640 k.
Opt. B —	Set the number of printers: (0 - 3)
Opt. C —	Enable/disable the A/D game port.
Opt. D —	Startup video mode selection.
Opt. E —	Auto Boot Enable/Disable.
Opt. F —	Set Number of serial ports: (0 - 7)
Opt. G —	Set Keyboard key tone. (click - beep)
Opt. H —	Set Color or B&W display. (This one is green ;-)
Opt. I —	Set PC/XT Compatibility mode. (Not tested)
Opt. J —	Set Serial Port A (baud, parity, stops, bits)
Opt. K —	Set Serial Port B (baud, parity, stops, bits)
Opt. L —	Set number of disk drives (1 - 2)
Opt. M —	Set Color Palettes (Set this one to green ;-)
Opt. N —	Redirect line printer (serial or parallel port)
Opt. O —	Set serial ports (Seequa or IBM Style, untested)
Opt. P —	Set scroll Option (fast or slow)
Opt. R —	Reset options to factory defaults.

The other utilities are software to establish a RAM disk, a print spooler, and a communications program (a quick port of the CP/M stuff on bulletin boards).

Now for the GOOD stuff! The Z-80 and CP/M 2.2! The system disk sported the basic CP/M environment utilities and booted a 64k system with CCP at E400, BDOS at EC06, and BIOS at FA00. Exploration of this system was just a bit slow at first until I was able to define the disk format so I could work with the OS on my regular machine with Z-System utilities. The ASM and LIB files supplied are generic MDS-800 files and do not contain the Seequa - Chameleon specifics. Therefore, to find out things about the system, I needed to PEEK ram in the BIOS area. This, too, was a bit frustrating as some of the BOOT code is overwritten when no longer needed - good programming practice to conserve memory, but tough on peekers.

The break came during formatting and sysgen process on a new disk. I noticed that the disk space decreased when the disk was sysgened. This meant that there was a directory entry for the boot system. Direct read of the disk using the monitor revealed a CPM80.SYS file with high bits set to SYSTEM status. Clearing this flag gained access to the system tracks using DUMP or DDT. Now I was cooking! Examination of the bios Disk Parameter information allowed me to set up my machine to read/write the Chameleon's disk format. Also, I now had the machine specific information on ports, etc. From this file, we can build the correct CBIOS.ASM file to allow development of the system. I'll not go any further down this path — don't want to steal all of the fun! The disk format information is of vital interest to any one wishing to develop or assist someone with one of these machines. So, I'll divulge what I know:

Disk type — Soft sectored, 5.25 inch, 40 track, Single or Double Sided, Double Density.

```
STAT DSK: d:DSK: command displayed the following:
                (where d: is the disk drive being read)
     2528 : 128 byte record capacity
      316 : k-byte drive capacity
       64 : 32 byte directory entries
       64 : Checked directory entries
      256 : Records / Extent
       16 : Records / Block
       32 : Sectors / Track
        1 : Reserved track
```

Listening to the disk drive stepping during format hinted that we wrote outside to inside tracks on side 0 and inside to outside tracks on side 1. This proved to be the case for formatting, but not for disk reads and writes. So, set your emulator for SSDD or DSDD, DD on Track 0 - side 0, all sectors side 0 then all sectors side one (DSDD), 5.25 inch media, 300 RPM, 2k allocation size, and 512 physical sector size. Configure the rest as follows:

	Single	Double Sided
Skew factor	1	1
Start sector	1	1
Physical sectors/track	8	8
Physical tracks/side	40 (28h)	40
Logical sectors/track	32 (20h)	32
Block shift	3	4
Block mask	7	15 (0Fh)
Extent mask	1	1
Disk size -1	155 (9Bh)	157 (9Dh)
Directory maximum -1	63 (3Fh)	63
Allocation 0, 1	192, 0 (0C0h)	128, 0 (80h, 0h)
Check size	16 (10h)	16
Track offset	1	1
Skew table	1, 2, 3, 4, 5, 6, 7, 8	(same)

Next, I was interested in how the unit would work with ZCPR. Everything worked under NZCOM (ZCPR 3.4) without a single hitch. The terminal behaves very well when treated as a TVI-920. The resulting system (full ZCPR system) only occupied 8k, leaving a 57k TPA. Since I don't have schematics or much experience with this machine, I can't speak to the ease or expense of adding a hard disk. If this feat can be reasonably accomplished, one could easily have a really neat luggable unit.

To sum it all up, keep your eyes pealed at yard sales and flea markets. This little box can be a lot of fun for not much money! - Ken.

Great Ken! Sounds like you had lots of fun. Thanks. Bill.

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

Volume Number 4:
- Issues 26 to 31
- 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
- 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
- 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
- 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
- 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
- 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:

- 15 copies now available -

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.

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.

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.

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.

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.

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

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.

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.

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.

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.

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.

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.

The Computer Journal Back Issues

• 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

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

ssue 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

ssue 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

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

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

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

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

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

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

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!

Issue Number 58:
· Multitasking Forth
· Computing Timer Values
· Affordable Development Tools
· Real Computing
· Z-System Corner
· Mr. Kaypro
· DR. S-100

Issue Number 59:
· Moving Forth
· Center Fold IMSAI MPU-A
· Developing Forth Applications
· Real Computing
· Z-System Corner
· Mr. Kaypro Review
· DR. S-100

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

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

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

Issue Number 63:
· SCSI EPROM Programmer part II
· Center Fold XEROX 820
· DR S-100
· Real Computing
· Multiprocessing Part II
· Z-System Corner
· 6809 Operating Systems
· Reminiscing and Musings
· IDE Drives Part II

Issue Number 64:
· Small-C?
· Center Fold last XEROX 820
· DR S-100
· Real Computing
· Moving Forth Part IV
· Z-System Corner
· Small Systems
· Mr. Kaypro
· IDE Drives Part III

Issue Number 65:
· Small System Support
· Center Fold ZX80/81
· DR S-100
· Real Computing
· European Beat
· PC/XT Corner
· Little Circuits
· Levels of Forth
· Sinclair ZX81

Issue Number 66:
· Small System Support
· Center Fold: Advent Decoder
· DR S-100
· Real Computing
· Connecting IDE Drives
· PC/XT Corner
· Little Circuits
· Multiprocessing Part III
· Z-System Corner

Issue Number 67:
· Small System Support
· Center Fold: SS-50/SS-30
· DR S-100
· Real Computing
· Serial Kaypro Interrupts
· Little Circuits
· Moving Forth Part 5
· European Beat

Issue Number 68:
· Small System Support
· Center Fold: Pertec/Mits 4PIO
· Z-System Corner II
· Real Computing
· PC/XT Corner
· Little Circuits
· Multiprocessing Forth Part 4
· Mr. Kaypro

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

	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

File System and More

The Computer Corner

By Bill Kibler

For many issues in TCJ I have talked about a universal operating system. We have reviewed CP/M and will be reviewing OS9 in greater detail as well. In the last two issues I mentioned and reviewed Forth Inc's polyForth. Since my interest is really on building a universal operating system that any 8 bit user could use or adapt with little problem, my interest in polyForth has to do with their features that would guide us in a better design of the universal system.

There are many Forths on the market, just as there are many types of clone machines. Sometimes it is nice to have the original product and polyForth is as close to the original Forth as one can get. The difference between getting an original Forth and an the original IBM PC is that the Forth version works very well unlike those first PC's.

I am not here to sell polyForth but to explain how looking at it can help us better understand the task ahead, if we want to truly produce an operating system that can run on any small 8 bit system. We can help that understanding by looking at features that might be useful to implement.

One feature that polyForth has, and which is common in all operating systems, is virtual I/O. They have chosen a method of virtual disk I/O that has many advantages that I feel you need to know about. To understand their choice also requires a little background in how operating systems handle disk I/O.

DISK I/O

Since a large percentage of our readers are CP/M based, let us use that as a base implementation. To review the idea of CP/M, it is to isolate the hardware features from the software requirements. Thus if hardware platforms vary in how they do I/O (terminals, printers, disks, etc.) the running software program still runs on all variations of hardware possible.

This isolation between hardware and software came by separating the disk and I/O requests from those of the actual hardware. The BDOS provides a list of BASIC (DOS- DISK Operating System) functions, that make requests of the BIOS (BASIC Input & Output System) that is specific to the hardware design. Although easy to see why this was done, before CP/M almost all programs were written for specific hardware and would not run on other platforms.

Through the use of the BDOS interface our software program gets a VIRTUAL view of the I/O. In the case of the terminal, we can just send data to it without understanding the physical type or interface protocol used. The same is true for the DISK I/O. The BDOS sets up an imaginary disk structure which is used the same in all implementations.

Since this article is really food for thought, I will leave the greater details for a later time. The simple details of CP/M however relies on the use of BLOCK PARAMETER TABLES. These tables convert the virtual disk data locations into a physical disk track and sector location. Thus a request for a certain block of data, gives a corresponding track and sector value to the BIOS for retrieval.

Forth in the beginning

The good points of this type of operation are the isolation of physical disk design from the program, and similar operation across platforms. A negative point is the need to have single disk capacities large enough for a given program. Since each disk has it's own block table and they may be different from disk to disk or disk type, it doesn't provide a truly VIRTUAL disk structure then.

Forth was designed in the early days of computing and saw this problem from a different view point. The virtual concept started with terminal character size. All terminals then, were 60 characters by 16 lines. This size also amounts to a 1K (1024 bytes) buffer space. So Chuck Moore used this screen size as the controlling factor in the design of Forth.

His idea was limiting coding design to ONE screens worth. These single screen worth he called BLOCKS. From a programmers point of view, each block would represent a single idea or procedural part of the program. The disk file system then gets broken into 1K segments that each represent a single program function. Much like the disk parameter table, each block is given a single block number. Unlike CP/M however, since Forth rides on top of the operating system, the block numbering need not end when the disk is full.

Before the days of hard disk systems, four or more floppy disks were not unusual. Most programs could use the other disks, but only when directly referenced. Your program would not run if the disks were out of order, or a failure happened to one of the drives. I remember being unable to use programs because they hard coded drive designations and made no means for altering them.

Chuck's answer was virtual block numbering. At boot time you mount the drives and their beginning and last block number (more on mounting later). Since you always have access to the source code, changing this mounting procedure is possible in cases of problems. You can also mount as many as you like and have their block number continue from one disk to the other. From the programmers view point, they have a virtual disk of unlimited size. From a users view, I mount the range of blocks I need at the moment as well as being unconcerned about disk size.

Now many systems mount disks. Mounting a disk, is in essence telling the disk operating system, the size and nature of the media just attached to the system. All PCDOS disk have an ID table in the first sector that provides this information (track/sectors/sides). UNIX systems use mounting to attach the disk into the directory structure. Operating systems like NT will also have some method of mounting that tells the system which operating system to emulate when loading and running programs from that disk.

TOO many variations

For readers of TCJ, having so many variations of disk systems and disk formats is a big problem. I find the Forth method something worth considering. Frank Sergeant's PYGMY Forth uses this method as well as polyForth. Try PYGMY Forth, it's free to start with, and see how Frank arranged his blocks and programs. Consider then, that each grouping of blocks could be a separate disk or ROM.

Suppose you have an embedded controller that needs a large menu of user functions. Our 64K system could have the main portion in the lower 32K or 32 blocks of program. The upper 32K might be multiple 32K ROMs. The program deals with one list of blocks, while the kernel switches banks of ROMs as the block numbers increase (64,96,128, etc...).

I found blocks very beneficial when doing my master's tutorial program. The ability to load a given block by number made the tutorial a snap to produce. Trying to use pointers or markers in a single file structure would have made the program many more times complex and very difficult to change.

DO IT VIRTUALLY

Whatever operating system you currently use, you are doing virtual I/O. When working with embedded controllers and small systems, the design or features of the virtual I/O become important. If you have massive amounts of horse power, disk space, and screen to squander, then understanding or taking advantage of your I/O is probably of little concern.

When we continue the discussion of an universal small system operating platform, usage of the virtual I/O will be very important. I think some concept along the lines of Forth's block system, with it's ability to span multiple disks should be considered.

Z180ISA and CP/M CDROM?

I talked to ZWorld last week after reading their newsletter. Seems they just released a Z180 PC104 board. The fine print however indicated an ISA bus (PC/XT format) card will be released in Oct/Nov of this year. They promise to send me more information soon. So I guess I can stop designing one myself.

Actually I would like to know if anyone has started (or finished) porting ZCPR/CPM to any of ZWorld's products or any modern products like the PC104 board. We really need to have a list of new products and their implementations. So drop me a note if you've done this. I think the PC104 might offer some real impressive features, not to mention speed and display options. What's the PC104 interface? Well the PC Bus is rather large and for embedded work small is the main consideration. So people started stacking very small size cards. The standard pin configuration is the small as the XT and AT bus, just they are vertical pin and sockets, not the card edge and no backplane BUS! (More later.)

I called Walnut Creek CDROM (800-786-9907) just before each issue to see if the CP/M CDROM is ready. This time I was promised it would be mastered next weekend (Sept 19) with shipment the following week. Now I have been promised this before, but the lady said it really is going then and I will get mine soon. Hopefully by the time you get this issue.

SMALL PLC?

I was reading B&B catalog (815-434-0846) the other day at work and saw they have a small PLC program for PCs. It is designed to work with their line of parallel I/O devices, but can be adaptedeasily to any I/O structure. I have run it and it works, only it uses simple ladder structure. Real complex ladder functions aren't possible. It looks like it is done in Pascal and would provide some simple training on hoe PLC's work. It will run in the background, so I guess you could use it on an XT or any system you have on all the time.

The items I like are data acquisition products, all based on serial and parallel port usage. Their parallel port is $99 and with the $99 price of the PLC program it all would be a bit high for a PLC system with only 8 I/O lines, but then with the cost of PC/XT's at $5 each, it still comes out cheaper than most commercial PLC products. Maybe we can do one for less, hmmmmm...

Speaking of PC/XT's, I saw something that made me appreciate CP/M. I have a flyer explaining a program that will run on DOS, or Windows, or OS2. What caught my eye was the needed space for each platform. Both windows and OS2 needed 400K of disk space with 160K of memory, all for a simple background program. How do I know it is simple, the DOS version needs 40K of disk space and 4K of RAM. The only expression I can think of to compare this to is a fly swatter to a nuclear bomb. Now of course CP/M proably would only need 4K of disk and about half a K of RAM, but then that is only a guess.

Next time?

My pile of to be talked about is getting large so next time maybe a slightly longer Computer Corner? Will see.

TCJ CLASSIFIED

CLASSIFIED RATES!
$5.00 PER LISTING!

TCJ Classified ads are on a prepaid basis only. The cost is $5.00 per ad entry. 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.

Commercial Advertising Rates:

Size	Once	4+
Full	$120	$90
1/2 Page	$75	$60
1/3 Page	$60	$45
1/4 Page	$50	$40
Market Place	$25	$100/yr

Send your items to:
The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535

Historically Brewed. The magazine of the Historical Computer Society. Read about the people and machines which changed our world. Buy, sell and trade "antique" computers. Subscriptions $18, or try an issue for $3. HCS, 10928 Ted Williams Place, El Paso, TX 79934.

Wanted: Good complete floating-point package (IEEE single and/or double precision) for the 8051 Micro. Should be public domain, but commercial better than nothing. Send info to tilmann.reh@hrz.uni-siegen.d400.de.

Wanted: Mother board for one of the following Kaypro 4X, 2X, or possibly a 1984 Kaypro 2. Charles Brown, Box 1046, Twin Peaks, CA 92391 (909)337-3049.

For Sale: Radio Shack Model II, 64K, very nice, runs great, CPM2.24, lots of manuals and software. $65 plus shipping. Call 805-491-3421. Ask for Charlie or leave message.

For Sale: 2 XT computers, $50 each, 2 101-Key KB's, $12 ea. 3 Epson printers, $50 ea. 30 copies of CC;Mail DOS platform pack w/8 users, V3.2, $100 ea/offer on all. Compaq portable, IBM KB's, HP terminal, Toshiba 321 SL, Xerox workstation CPU (80186 & NS processors) circa 1982: Make offers. Misc. electronic parts; fans, fuses, tubes, power supplies, ask. Mark Mowery, 400 W. 5th #42, Grandview, WA 98930-1254, Tel. (509)882-2940.

Notice: PseudoCorp has moved. The new address is : 921 Country Club Road, Suite 200, Eugene, OR 97401.

6811 and 8051
Hardware & Software

Supporting over thirty versions with a highly integrated development environment..

Our powerful, easy to use FORTH runs on both the PC host and Target SBC with very low overhead

Low cost SBC's from $84 thru developers systems.

For brochure or applications:
AM Research
4600 Hidden Oaks Lane
Loomis, CA 95650
1(800)947-8051
sofia@netcom.com

DIBs Electronic Design

Dave Baldwin

6619 Westbrook Dr.
Citrus Heights, CA 95621

Voice/Fax (916) 722-3877
DIBs BBS (916) 722-5799

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

$79.95 **68HC811**
A1 Version - $59.95
Single Board Computer
Develop Your Own Projects
SBC-E2

Programs completely from PC via RS-232. **2048** Bytes EEPROM. 256 Bytes RAM. 24 - TTL I/O Bits. 8 - 8 bit A/D Inputs. SPI.

SBC-E2 is low power CMOS, <20 ma, 5 volts DC. 3.1" x 3.6". FREE Bootloader, 480 pages of documentation, schematics, utilities, sample programs and source code included.
Add $3.50 shipping. MD residents add %5 tax.
Pre-paid or COD only.

LDG Electronics
410-586-2177

1445 Parran Road
St. Leonard MD
20685

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:	$25
Reinsertion:	$20
Full Six issues	$100

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

NEW MAGAZINE
the world of 68' micros
supporting
Tandy Color Computer
OS-9 & OSK

$23/year for 8 issues
$30/year Canada/Mexico
$35/year overseas

Published by:
FARNA Systems
P.O. Box 321
Warner Robins
GA 31099-0321

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

THE FORTH SOURCE

Hardware & Software

MOUNTAIN VIEW PRESS

Glen B. Haydon, M.D.
Route 2 Box 429
La Honda, CA 94020

(415) 747 0760

PCB's in Minutes From LaserPrint!*

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

* Or Photocopier Use household iron to apply.

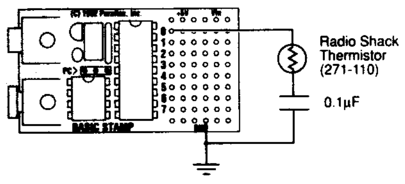
Press-n-Peel

PnP BLUE or	**PnP WET**
For High Precision Professional PCB Layouts	Easy Hobby Quality PCB's
1. LaserPrint	1. LaserPrint
2. Iron-On	2. Iron-On
3. Peel-Off	3. Soak-Off w/ Water
4. Etch	4. Etch
An Extra Layer of Resist for Super Fine Traces	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

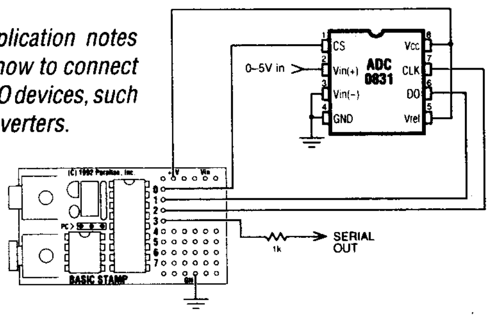
BASIC Stamp

$39 single-board computer runs BASIC



The Stamp can measure resistance with just a few low-cost parts.

Helpful application notes show you how to connect common I/O devices, such as A/D converters.



- BASIC language includes instructions for serial I/O, PWM, potentiometer input, pulse measurement, button debounce, tone generation, etc.
- Has 8 digital I/O lines, each programmable as an input or output. Any line can be used for any purpose.
- Small prototyping area provides space for connecting signals and extra components.
- Powered by 5-12 VDC or 9-volt battery.
- Consumes just 2 mA (typical) or 20 µA (sleep).
- Special cable connects Stamp to PC parallel port for programming.
- Programming Package includes PC cable, software, manual, and technical help for $99.
- Individual Stamps may be purchased for $39.
- Requires 8086-based PC (or better) with 3.5" disk drive.

PARALLAX

Parallax, Inc. • 3805 Atherton Road, #102 • Rocklin, CA 95765 • USA
(916) 624-8333 • Fax: 624-8003 • BBS: 624-7101

TCJ The Computer Journal
Post Office Box 535
Lincoln, CA 95648-0535
United States

BULK RATE
US POSTAGE
PAID
Lincoln, CA
PERMIT NO. 91

ADDRESS CORRECTION REQUESTED
FORWARDING AND RETURN POSTAGE
GUARANTEED

Telephone: (916) 645-1670