Providing Support Around The World

The Computer Journal

Issue Number 67 May/June 1994 US$4.00

ISSN # 0748-9331

TCJ - For Having Fun With Any Computer!

68XXX COMPUTER PRODUCTS From
Peripheral Technology

68000 System Boards with 4 Serial/ 2 Parallel Ports, FDC, and RTC.

PT68K4-16 with 1MB $299.00
PT68K2-10 w/ 1MB (Used) $149.00
REX Operating System Included

OS9 V2.4 Operating System $299.00
With C, Editor, Assembler/Linker
SCULPTOR V1.14:6 for Business Software Development - requires any version of OS9/68K. $79.00

Other 68XXX products available!

1480 Terrell Mill Rd. #870
Marietta, GA 30067
404/973-2156

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.

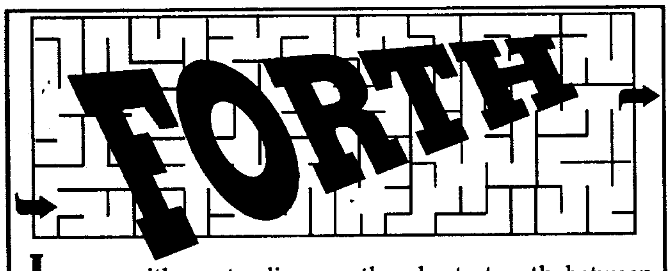
Quality Solutions
PseudoCorp has been providing quality solutions for microprocessor problems since 1985.

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
716 Thimble Shoals Blvd, Suite E
Newport News, VA 23606
(804) 873-1947 FAX: (804)873-2154



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

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
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.

Subscription rates within the US: $24 one year (6 issues), $44 two years (12 issues). All funds must be in U.S. dollars drawn on a U.S. bank. Send subscription, renewals, address changes, or advertising inquires to: *The Computer Journal*, P.O. Box 535, Lincoln,CA 95648.

TCJ The Computer Journal

Issue Number 67 May/June 1994

EDITOR'S COMMENTS

Welcome to issue number 67, our special on Forth. Well actually not a special just a couple of big articles.

We start out as usual with a much larger Reader to Reader section, which means it is not usual since it is larger. However the TCJ mail bag is still very large and as such I expect to have as many or more letters in the issues to come.

Of special note are two mini-articles in Reader to Reader. Charles Shattuck treats us to why he likes polyFORTH. The review covers a product with over 20 years of development! So if you want to know about a truly mature and excellent Forth read on. For those unable to make the Trenton Computerfest, Ron Mitchell gives us a quick review of some of this years activities.

Our man in Europe is back and Helmut Jungkunz gives you the software side of Amstrad's history. I didn't myself know of Helmuts skill or past experiences till now, but it sure seems he was well mixed up in giving users more power in using the Amstrad system.

Dr. S-100 was also at the Trenton Computerfest, but unlike Ron, Herb mainly bought and sold. He did get some Z-Fest discussions and pizza consumed, so all was not purely business. Herb also answers letters and ask for your input on the IDE S-100 project.

Ronald Anderson is next up in the pages and gives us some of his experiences, both of past and present. For those not aware of the 6800 or 6809 CPU internals, he gives us an insight in his beginners page.

Paging on as we must, produces the centerfold, which supports Ronald's work, by documenting the SS-50 Bus. I provide the pin out and jumpers from the Gimix SS-50 bus system. As you will see, the bus is based on the 6800 family and is much simpler than the S-100 bus design. The SS-30 was also a great way to save money in the early days when board design and construction was so expensive.

For you Forth people and actually anyone wanting to know about interrupts on a Z80, Walter Rottenkolber shows us how to do serial interfacing on a Kaypro II, all in Forth. I have found over the years that many programmers and hacker don't fully understand interrupts or how to handle them. Walter takes a very good stab at helping anyone get a better understanding of what goes on and how to deal with it. So if your serious about understanding what you are doing read on.

We take a break from heavy hitting with Rick Rodman's Real Computing. Rick gives us the good news that his Tiny-TCP is done and working. Good going Rick. The code which is on the BBSs' is ready and waiting for experimentation. Rick also updates us on the death of a bus and some Windows programming tips.

For those wondering about Brad Rodriguez's moving Forth series, the code is in. Brad has the first listing of his Z80 Forth primitives. Now this is part 5 of the series and if you haven't been following his work this will seem a bit over much. Brad's other four articles however have slowly explained the how and why of building your own Forth. He now brings together all those concepts and design decisions you got to consider into one listing. My compliments and congratulations go out to Brad for this massive project.

From the massive we jump back to the Little Circuits as Dave Baldwin discusses design problems your hardware layout should avoid. Dave's ''Little Circuits'' cover one of those little points that many would be designers get in trouble over, wire capacitance and inductance. Dave reminds you that PC board traces are wires too.

Truly last as usual is my own short column this time. I add a few more comments on using PLC's and provide a short Forth snippet of possible PLC words. Now if one of you readers has actually done a complete PLC system in Forth, please let me know and we will see about an article on it.

I think I am catching up on back orders and mail, but alas I have found a few misplaced items the other day. Please give me a call if you feel something has fallen through the cracks. My office hours are normally Mondays 9 to 11 AM ONLY. I almost had to go out of town on business but instead just changed my days off around so I could put more effort in fixing problems over the phone instead of in person. So when I say normally in my office on Mondays, I mean normally baring unplanned interruptions.

It has been some time since we had inquiries about 68306's and explained how you could buy them. I have yet however to receive any articles on using them. So please, if you have started or completed work on that chip, let us all know. Your letters or mini articles are always welcome by those about to wonder down that rocky design path to knowledge.

With that I give you issue number 67 for your reading enjoyment.

Bill Kibler.

READER to READER

Letters to the Editor

All Readers

MINI Articles

Dear Editor,

I am trying to get more specific information about *The Computer Journal.* I would like to receive a copy of it to be able to write a review about it in our newsletter ZXir QLive Alive! which is dedicated tot he Timex Sinclair computers. And also those that are using other computers to emulate the T/S computers.

I do not know the cost, the frequency of publication or the number of pages of the magazine. Please inform me of the data.

Something that you might want to publish is that the Timex Sinclair Users go to the Dayton (Ohio) Computerfest the last weekend of August and have several tables in the flea market area devoted to T/S computers only. We started that in 1991 and it is now an annual thing. The majority of the T/Sers stay at the Red Roof Inn North which is about 9 miles from the Hara Arena which is the location of the Computerfest. I like to arrive about mid afternoon on Friday and greet (at Red Roof) the other T/Sers as they check in. For me it is getting more and more like a reunion than anything else. Meet and thrash out problems, greet old friends, swap equipment, find things, and just socialize in general.

I am looking for book on CP/M hopefully a tutorial or manual. I have a disk interface (AERCO FD-68) for a Timex/Sinclair 2068 that has its own DOS plus a DOS called RP/M that is supposedly compatible with CP/M but the AERCO manual does not have much information but refers you to the CP/M manual. It is not a must have situation but I would like to find a manual to satisfy curiosity. I went to the meeting of ISTUG (Indiana Sinclair Timex Users Group) April 30th which is a 150 mile one way drive for me. So you see I am a dedicated T/Ser. I also have 3 different disk interfaces for the T/S 2068 computer none of which are compatible. One of the disk interfaces is still being manufactured so it is not a dead machine yet.

TIMEX/SINCLAIR STILL LIVES.

Sincerely yours, Donald S. Lambert, Auburn, IN.

Well Donald I hope you get my mailing in time for the computerfest. I will ad some flyers for TCJ, plus a few samples, like #65, our Sinclair Z80 special. My work will keep me from going to Dayton, but hopefully people like you will drop me some letters I can print to let our readers know what they missed.

RP/M is actually functionally equal to CP/M 2.2 and any CP/M manual or book will work ($15 for an official CP/M 2.2 manual set from Lambda Publishing - see inside back cover). RP/M was sold and written by Jack Dennon, of microMethods, PO Box 909, Warrenton, OR 97146 (503)861-1765. Jack disassembled CP/M 1.4 for his Hayden book ''CP/M Revealed'' only to have to redo it again as version 2.2 came out while his book was being readied for printing. After that he decided to take what he learned and wrote RP/M from scratch. Jack says his version is a little like ZCPR with some of the extra features. The main difference is, RP/M will work on 8080 CPU's.

Several OEM's bought RP/M for use with their systems, AERCO being one. Jack says most of the small OEM's really didn't pay but two other OEM's did make it worth his time, somewhat. The older version of RP/M's manual, of which I lost mine in the last move, provided a complete listing of RP/M. Thus RP/M gave you a look at how the insides of a CP/M compatible operating system might be done. He has a newer version that runs on top of a Z280 emulator on the PC. He is not actively selling any RP/M versions, but if pressured to sell he collects $129 for it with printed manuals, slightly less if all on disk. Jack is sending me a sample to review, so I can keep my readers up to date.

Since RP/M could be supplied with source on disk, and your 2068 disk controller builder's might have changed things, we can't promise, but feel rather strongly that you could purchase NZCOM from Lambda Publishing and run it. Since Jay Sage has spent the last several years explaining the how and why of using ZCPR or NZCOM in TCJ, our back issues are probably a good place to learn about ZCPR and CP/M for that fact.

So please let me know how your work with using CP/M or RP/M (or NZCOM) goes, and how Dayton T/Sers spent their time this year. Thanks for contacting TCJ. Bill Kibler

Dear Bill.

Reading ''Reader to Reader'' in the issue 66 I had a bad conscience. So I have now collected the files regarding ZX81 ROMs on a disk, which I send you.

Please note there are some old copyrights, but to the best of my knowledge, the firms don't exist anymore. Anyhow, I think the software is so good it deserves to be used more.

Now you can do with it what you think best.

Regards, Gorm Helt-Hansen, Denmark.

```
Directory of B:\ZX81_ROM

ASZMIC  HEX    11537 04-26-94  10:52a
FORTH   HEX    23056 04-26-94  10:54a
FORTH   DOC    37502 04-29-94  11:14a
ASZMIC  BIN     4096 04-27-94  12:29a
FORTH   BIN     8192 04-27-94  12:25a
ASZMIC  DOC    23661 04-29-94   1:13a
8 file(s)     108044 bytes
```

ZX81-FORTH.
EPROM: 2764.
Checksum: C925.
Copyright (c) 1983 Skywave Software.

ZX-81 Forth matches the fig-FORTH commands, although ZX81-FORTH is not fig-FORTH. It was not possible to include all the fig-FORTH words because of ROM space limitations. ZX81-FORTH also contains some non-standard words so that multitasking can be accomplished. ZX81-FORTH is multi-tasking. This gives the programmer the ability to write real-time routines.

ASZMIC.HEX.
Eprom type: 2732.
Sumcheck: 2A44
Copyright (c) 1982 Comprocsys Ltd.

ASSEMBLY LANGUAGE DEVELOPMENT SYSTEM for the Sinclair ZX81.

After connecting the power to your ZX81 you will have a clear screen, except for a funny little character towards the bottom left and a blinking cursor on the line just above it. The speed of the blink identifies which mode ASZMIC is in: fast blink means EDIT and slow blink means DEBUG. With 16k of memory on the system you will be in DEBUG mode but if you are using a "bare" machine then you will be in EDIT mode. Shifting between EDIT & DEBUG modes is achieved by the use of the Shift 9 & Shift E keys. Experiment by pressing Shift 9 (DEBUG) and Shift E (EDIT) alternately, and watch what happens to the cursor blink rate. There is only one difference between EDIT and DEBUG modes, but that is an important one. When you hit newline in DEBUG mode the line you have just finished will be passed to the Command Interpreter. In EDIT mode you just start a new line.

Thanks Gorm, the code is very much appreciated. I will be putting this up on DIBS and JW's BBS, both of which are listed in user group section. The bad part is, I will have to burn a ROM and test them out. I tried them with the ZX81 emulators and was unable to get either one to work. Not sure why the emulators didn't work, but maybe some one else can try who has worked on the emulators longer than myself.

Anyway Gorm, I know some others who will appreciate these files a whole lot. Thanks again! Bill.

Dear Bill

Thank you for the sample issues of *TCJ*. I especially enjoyed the article about the ZX-81 in issue #65, and the readers comments is issue #66.

Some points of interest to you and your readers: Sinclair Research manufactured 6 computers: ZX-80, ZX-81, Spectrum (3 versions) and QL. Timex Corporation manufactured 3 computers: TS1000, TS1500 and TS2068.

ZX-80: B&W video, 1K internal memory, 4K operating system. Later Sinclair released a modification that consisted of a new keyboard overlay and 8K operating system. (Same operating system as ZX-81.) This unit had approximately 30 integrated circuit chips.

ZX-81: B&W video, 1K internal memory, 8K operating system. Feranti Semiconductor developed a special 40 pin I.C. for Sinclair, which replaced 25 I.C.'s used in the ZX-80. The ZX-81 used 5 I.C.'s.

TS1000: Same computer as the ZX-81, except with 2K internal memory.

TS1500: B&W video, 16K internal memory, modified 8K operating system used in the ZX-81 and TS1000.

The ZX-80, ZX-81 TS1000 and TS1500 all used same software.

Spectrum: Color video, 3 versions - 16K, 48K, and 128K internal memory. Very few Spectrums made it to the U.S.A.

TS2068: Color video, 48K internal memory, different operating system than Spectrum. If operating system ROM was replaced with special after market EPROM, Spectrum programs could be run on TS2068.

All above computers used Z-80A CPU's.

QL: Color video, 68000 CPU, 16 bit Super Sinclair computer.

This is a thumb nail history of the Sinclair and Timex computers.

Best Regards, Fredric Stern, L.I.S.T. Newsletter Editor, PO Box 264, Holbrook, NY 11741. (Long Island Sinclair Timex user group)

Thanks for the short history, Frederic. I am starting to understand why so many have good feelings about the Sinclair machines. For one reason, as you showed us, there were many versions, with many features, and all for a very reasonable price. Again, Thanks. Bill Kibler.

Dear Bill:

I have been remiss in sending you money; I hope that I will not lose any issues of *TCJ* due to my getting hung up in trying to keep my clients happy. I do have a couple of items, however, that you and your readers may find useful.

First, at a recent Motorola seminar on TPU (timing co-processor in the 68332 chip), I passed on your complaint to Motorola people that "real people" (i.e. non-professionals) don't have reasonable access to state -of-the-art chips. I was assured that Wylie and Hamilton-Hallmark are two distributo[illegible]ave telemarketing divisions that will [illegible]e orders for small quantities (read that as [illegible]") of anything they sell, over the telephone, from anybody, presumably paid for with credit card. I have not yet followed up on this.

Second, I would like to recommend to you and all, and particularly to Brad Rodriguez, a terrific source for proto-

type-quantity printed circuit boards. You might also want to approach them about advertising in *TCJ*. AP Circuits will make printed circuit boards from Gerber files that you send to their bulletin board, with turn-around of four (yes that's 4) working days, for amazingly low prices. The last boards I had done, which were about 2.5" x 3.5", cost me $75 for two boards, including Federal Express shipping. There are no artwork or setup charges. If you call their bulletin board, you can download a file with their rates and requirements. Also, you can download a public domain version of the printed circuit board layout program PROTEL EASYTRAX for the cost of the telephone call (it's about 700K bytes, as I remember), with which to do the board layout and produce the Gerber files. Here's their address and phone numbers:
AP Circuits, #14-3650 19 St. NE, Calgary, Alberta, Canada T2E 6V2, (403) 250-3406, BBS: (403) 291-9342 (8N1).

While I'm playing with my new word processor, let me say how much I enjoy Brad Rodriguez' articles on Forth and the 6809. I am also pleased that you have corralled Ronald Anderson. I let my subscription to 68 Micro lapse a few years ago while I was out of the country, so I had lost track of him. I think he'll be a fine addition to *TCJ*.

And, of course, here's my subscription check.

Sincerely, Wilfred S. "Steve" Brown, Houston TX.

Thanks for the help Steve. Yes we have heard about more companies taking credit card calls for small quantities. I think it is a market pressure change, but even with a minimum charge it means you can get the parts you need.

AP Circuit had been recommended before. We had not however heard from anyone actually using them. Since you seem to be a satisfied users of their services and since they seem to be doing things right (like giving away a program to do the design with), I feel good about recommending them to my readers.

Adding Ronald Anderson has been wonderful, and I can tell he was looking for an outlet of his writing energy. He is several articles ahead and grinding them out faster than I can produce issues. As to which is more prolific, Ron or Brad, beats me, but TCJ's current growth certainly is owed to their work. I send them my thanks as often as possible.

And to you Steve, thanks for your letter. Bill.

Dear Bill,

To update Brad Rodriguez's comment (*TCJ* #65) on Jameco, their most recent catalog (#941) for Feb-Apr 1994, drops the minimum order, but adds a $5.00 service charge if the order is under $25.00. This is similar to the policy of JDR and Digi-Key.

I agree with Brad's observations on languages, and your comments. Some of the most incomprehensible code I've seen was written in Pascal. With a little effort you can take any language and create a Write Once Read Never program.

I've adapted Frans van Duinen's PDE to my Laxen and Perry Forth83 for my Kaypro II. Though not as fancy, it provides an integrated programming environment as pleasant as any Turbo or Visual compiler. Another reason I prefer Forth is that debugging Forth is easier than other languages, esp. using the compilers available for eight-bit systems.

The C/80 compiler from Software Toolworks is supposed to have been derived form Small-C. Together with the Math-Pak, it provides all the basic C functions and data types except Typedef, bit structures, and register assignment. One big internal change was to assign local variables to memory locations rather than a stack frame. It generates assembly code fairly quickly. The assembler needs to tap into a compiled library. You could use a commercial assembler, such as RMAC. The supplied assembler unfortunately, is very disk intensive and slow. However, C/80 does show that Small-C could evolve into a fairly complete C compiler.

Yours truly, Walter J. Rottenkolber.

From an earlier letter by Walter:

Dear Bill,

When I first started in computers ten years ago, the 8-bit home/personal computer had fleshed out to the full 64 Kbytes RAM address space. And with this expanded RAM came an explosion of interest in adapting languages other than BASIC and assembler to the personal computer. As a result, there appeared magazine articles, books, and inexpensive compilers for a wide variety of languages. Over time, I collected and tried many of them. Some were obscure ("BPCL", Algol, Pistol, Mouse). Other were well known, but the mini-versions had limited usefulness and distribution (COBOL, Fortran, Lisp). Of them all, I'd say that five had the widest use, interest, documentation, and compiler availability -- Forth, Pascal, 'C', BASIC, and assembler.

Everyone tends to pick one or two of these because they 'work' for them. The languages not only do the job, but they fit in with the way the person thinks and likes to program. For me they are Forth and assembler. My Forth (an L&P F83 with PDE) has an integrated programming environment better than Borland's Turbo-Pascal, and both open up the computer on a more elemental level than the other, more abstract, languages.

I can read source code in the other languages, but I would object to being forced to write a demonstration program in them. I can sense from the comments of the other authors that they have similar feelings. As a result, I believe that the best compromise is for the working program to be in the author's language of choice, but that the essential algothrim be described in well commented pseudocode. After all, that's what the text of the article is for.

Since the readers are not at the same level of experience and knowledge, an article will have points of confusion despite the author's best efforts and intentions (I know!). A Q&A column separate from the letters to Editor column, would provide a forum for answering these questions. All this presupposes that the reader is willing to make the effort to learn. Nothing will help the lazy reader.

One project, implied in your article, would be to round up, develop, or find sources of languages. After all, F.I.G.'s efforts led to fig-Forth being available on most 8-bit platforms of the early 1980's. The process broke down for Forth-83, alas. The comments in your article suggest that Small-C had a similar distribution. BASIC and assembler were widespread though their availability may now be spotty for some systems. The status of Pascal is unknown to me. The Kaypro had Turbo-Pascal, JRT Pascal, and tiny Pascal (at least). Ideally, all five languages should be available for as many systems as possible.

A good start would be to develop a master list of all languages available for all systems of reader interest, not just the big five. And there are some systems I'd never heard of before -- Dolphin-8 and Propoise-8, indeed. And some strange languages -- Mouse. It would give the readers a good excuse to dust off their disk 'junk' box and revisit the past. Just for fun, I'm including my list.

A listing of languages for Kaypro II, CP/M computers. The (K-??) are disks from MicroC Kaypro library (now for sale by Lambda).

Algol -- ALGOL80.

Basic -- CBASIC2 (DRI), EBASIC, MBASIC (Microsoft), OBASIC (Microsoft), SBASIC (Kaypro), ZBASIC (Northstar), TINYBASIC v1.0 & v3.1.

Assembler Z80 -- HD64180A&B, Z80MR V1.2 (K-25), M80 (MicroSoft), CROWE-Z80 [Z80 (K-10) & ASMZ PD by C.C. Software], SPASM (Intel mnemonics, single pass), ZSM (for inclusion in C programs), Z80.PAS (compiler in Pascal).

Assembler 8080 -- ASM (DRI), LINKASM, MAC (DRI).

'C' -- C/80 v3.1 with mathpac (Software Toolworks), Small-C v1.1 & 2.0 & 2.1 (K7-8, K35-36).

Cobol -- Nevada COBOL (Ellis Computing).

Forth -- FIG-FORTH v1.1 (K-12&13), F83 v1.01 & 2.01 (K-32), UNIFORTH sampler , EFORTH51.

Fortran -- Nevada FORTRAN

Lisp -- LISP/80 (Software Toolworks), ILISP (based on SCHEME), LISP (in pascal), XLISP (for Kaypro).

Pascal -- TURBO-PASCAL v3.01A (Borland), JRT PASCAL v3.0 (JRT Systems), LITTLE-PASCAL (K16).

Pilot (computer aided instruction) -- Nevada PILOT (Ellis Computing), ZPILOT (in Z80code), PILOT/80 (in basic).

Misc -- APL-Z, BCPL (by Martin Richards & Colin Whitby-Strevens), OCTAL, MOUSE, PBS (basic preprocessor), STAGE2 (macro-preprocessor).

Yours truly, Walter J. Rottenkolber.

Thanks Walter and it seems you too have hit on the best way to handle the code problem. What is needed now is a standard way of doing pseudo code comments. I guess we could use Pascal style of coding as the comment format. I once worked with a programmer that did all her design in Pascal and then did the assembly work from that. After all wasn't Pascal written to teach and explain programming concepts.

You have a rather good sample of programming languages and I guess now that we are finding out who has what, some articles on personal experience with these older versions is needed. I would not want to recommend someone using an older version that was so buggy that the result was unusable.

I can see too that we have yet to get you to explain your PDE setup. I am sure many programmers have gone to Turbo products due to the integrated programming environment. Made me switch! How about it? Thanks for your never ending work! Bill.

Dear Mr. Kibler

I just received your complimentary copy of *TCJ*. I like it, good mag. Please enter my subscription for the next year. I will also order some back issues when I have time to figure out which ones I need.

Basically I am jumping into the middle of things. Yes, like most kids, (52 is still a kid) I had my share of computers. The Sinclair 1000 & 1500, a VIC 20, and a TI 99. I still have all of these and use the VIC 20 regularly as a dedicated RTTY terminal, and the TI 99 for games (mostly for the grandkids).

Never being into programming, I didn't use these old computers for anything but for software that was preprogrammed on tapes or cartridges. Budgetary and time restraints of raising a family didn't leave much for computing. Now I am picking up where I left off fifteen years ago. This is where *TCJ* fits in, helping me to learn and re-learn the past.

There is also the revolution aspect. Back in 1987 I bought my first 8088 XT. It had an unbelievable 256K of memory, and a monstrous 720K disk drive. Compared to my previous 2K of memory and no disk drive this computer would last me a lifetime. WRONG. I soon found that it was inadequate and had to get an AT 286 with 1M of memory, a 32 meg hard drive, 2 floppy drives and 512K of mem just on the video card. This lasted a couple of years, and found that due to the large programs I needed more. So off to a 386/40 with 8Meg mem, a 260 meg hard drive etc. This now isobsolete. WAIT STOP! Something is wrong! I am doing very little more now, than I did on Sinclair 1000. So back to the beginning, it makes a lot more sense, and a lot more fun.

I enjoyed your article on PLC's in the Computer Corner. A week ago I would have probably skipped over this article, but I just returned from one week cram course on the GE Series 1 programmable controller, so I was able to read and understand it. By the way the GE controller uses a Z80 with 2K of mem, and controls up to 128 I/O functions. The OS is in ROM, with all the functions entered from a keypad, i.e. dedicated keys for all the logic operatives, contact, internal and external coils, timers etc. Too bad I was not able to get a copy of the firmware, it would have been interesting.

Finally a couple of years ago I acquired an old TRS 80, 16B. This is quite a machine. It has a 68000 main processor with a Z80 co-processor to handle I/O's, 1Meg mem and dual eight inch floppies. It is very fast and if I can ever figure it out, it looks like there is a lot of potential there.

Thank again Bill for a fine magazine. I will be looking forward to many years of it.

Yours Truly, Lew King, Industry, PA.

Thanks for your subscription Lew. Yes I had a model 16 for awhile and they run Xenix, MicroSofts version of Unix. I sort of wished I had kept it, but I was running out of space.

As to the madness of the clone, use them and leave them problem, I can't agree more strongly. You want fun and real knowledge of how things work, keep your old machine and start writing your own programs. I guess one could learn CP/M inside and out in six months or so, while just getting programs to work on the latest DOS could take that long and wouldn't work right with the next upgrade. Pure madness.

To have fun with computing means knowing everything you are dealing with pretty well. That equates to keeping it simple. XT's are just barely simple, AT's a little bit over the edge but learnable, 386/486's may only have a few parts on the board, but software complexity is definitely not for those wanting to have fun.

Thanks again Lew and welcome to TCJ.

Dear Bill,

The back numbers arrived safely. Thank you. This concludes with an order for more.

I noticed mention of E Roche & Windsor BBUG on pg 45 of issue 65. This BB is the remnant of the CPMSDOS-UGUK, which was wound up last year. Roche was a prolific contributor to its newsletter & to The Disc Library News of the Windsor BBUG. Names and addresses of WBBUG of which you may be already aware by now, are:
Sysop: Peter Catley
11 Haslemere Rd.
Windsor SL4 5ET
England, UK
Disc Librarian: Rodney Hannis
34 Falmouth Rd.
Reading
England, UK
I was led to *TCJ* by Rodney.

Idea! I got much of my software from PD sources. Other late comers like me might enjoy thumb-nail biographies of authors who have shown us the way. Ward Christensen/LASM/DU, Dave Rand/NSWP, etal.

Sincerely HK Fraser, Blairgowrie, Scotland, UK.

Thank you for your order and letter Mr.Fraser. I like your idea and hopefully they or people with knowledge of them will see this and send me some Bio's. I printed some information last issue about the UK group, hoping that people would know that at least the BBS and Disc libraries are still available.

Many support groups have disbanded, partly due to lack of interest, but more I think to people being a bit too busy with so many other projects, that their hobbies got left behind. The important thing to remember is that most of the BBS's and libraries of public domain software are still available. So you might not find a meeting to attend, but help and programs are often just a phone call away.

Thanks again. Bill.

Dear Bill,

Thanks for advertising my equipment. All three machines sold, along with the tech manual. I am pretty amazed that they all went so quickly -- one to a man in Texas just a couple of weeks ago. In fact, I could have sold the K-1 with hard drive four or five times!

I am enclosing a listing of the last of the stuff I have. As you will notice, I am pricing these things to move. Could you please list them in the *TCJ* "for sale" section for me? I am enclosing a check for $10 to pay for this and the first ad you ran.

Thanks for helping me find good homes for these machines. I will always have found memories of my start in computing with Kaypros and the Z-System.

Sincerely, Dave Templin, West Sacramento, CA.

Good to hear they sold quickly Dave. I think many people have the wrong impression that nobody wants older machines. My sons classmate's father was about to use 7 Z100s for land fill. My wife was taking the son to a birthday party at their house and saw them stacked in a corner awaiting burial in the back yard. A few words with the Father and they ended up in my garage, free.

Well one had been dropped wasting the monitor tube, but all else worked. We found that out as we swapped boards and tested them. The we in this case is a friend who is still using his Z100 I sold him way back when. He had since bought a spare unit when the local Heath/Zenith dealer went out of business. To say we had fun testing and fixing would be an understatement. I think we both could have played with these repairable units for weeks without complaint. I sent him home with two and a half machines. He now has at least three spares for every part in his own Z100!

With the prices you asked, getting spare complete systems is cheap and easy. Users who aren't hardware oriented can just buy a complete spare or two and feel safe for many years to come. Remember computers don't get old just sitting in some corner waiting to be used, there is not an expiration dates on these pieces of history.

Thanks Dave and good luck. Bill.

Dear Bill Kibler,

Please pass this on to Ronald W. Anderson. It's a response to his request in the TCJ #66 "Small System Support" column for a "C whiz" to write and explain why his example comparing the use of array indexes versus pointers is "clumsy." I suppose I qualify as a C whiz, since I make my living as a C programmer. We who "C" prefer pointers because they generally produce smaller, faster code. It's that simple. True, pointers may seem cryptic to C neophytes, but they are very important to the language. Now, pointers are much better because...

First, take a look at my example file, TESTP.C. It's practically identical to Ronald's example on page 34 of TCJ #66. I've left out the incorrect versions that printed out the terminating null. We're interested in comparing the two functionally equivalent versions, one that uses array indexing, and the other that uses a pointer. Two minor differences in my code from Ronald's. I didn't use the braces around the string constant that initializes the character array, string. They don't hurt, but are not necessary in this case. Second, I initialized pointer s by assigning string to it. An array name in C is the address of the 0th element, i.e. equivalent to &string[0]. Next, squint a bit and look at the assembly output from C80. This is the result of compiling TESTP.C by the CP/M command line:

```
A>C80 -M1 TESTP.C
```

The -M1 switch means to create an assembly file output in a format that the Microsoft M80 Macro Assembler program can use. I used this compiler switch so the mnemonics would be standard 8080. I've edited the file and put in some white space and comments. The array indexing version begins at the label .f, and the pointer version at .h. The code of both versions is virtually identical except for one big difference. The indexing version has four more machine instructions, and they are costly. The instructions are:

```
LXI D,string
DAD D
```

There are two occurrences of this sequence inside the loop to print one character. What it's doing is easy to understand. Each array indexing operation, string[n], requires that the address of string be fetched, and the value of n added to it. The pointer version doesn't need to do this, because the pointer s is simply used.

My test string has 25 characters, so when the array indexing version executes it performs 100 more machine instructions (on an 8080) to print the string than does the pointer version. Programs that perform a lot of iteration through arrays will suffer both a performance degradation, as well as increased code size, if they use indexing rather than pointers. There are many times that indexing into an array is a valuable option to have in the programmer's bag of tricks, but there is ample reason to prefer pointers in cases like Mr. Anderson's example.
+
Very truly yours,
Richard E. Brewster, Richmond VA

```
/* File: testp.c 4/24/94      Richard Brewster
Purpose: To see if array subscript or pointer is
more efficient in terms of assembly operations.

Ref: TCJ #66, p. 34 - Article by Ronald W.
Anderson
*/

char string[] = "This is the test string.\n";
                    /* Braces not necessary */
int main(void)
{
        char *s;
        int n;

        /* Array subscript version */
        n = 0;
        while (string[n])
                putchar(string[n++]);

        /* Pointer version */
        s = string;/* same as s = &string[0] */
        while (*s)
                putchar(*s++);

        return 0;
}
```

```
;TESTP.MAC
;C/80 Compiler 3.1 (4/11/84) - (c) 1984 The
Software Toolworks

CSEG
string:: DB     84,104,105,115,32,105,
         DB     115,32,116,104
         DB     101,32,116,101,115,116
         DB     32,115,116,114,105
         DB     110,103,46,10,0

         PUBLIC main
         DSEG
.d:      DW     0          ; char *s;
.e:      DW     0          ; int n
         CSEG

main:    LHLD   .e
         PUSH   H
         LHLD   .d
         PUSH   H

         LXI    H,0        ; n = 0;
         SHLD   .e

; while (string[n]) putchar(string[n++]);

.f:      LHLD   .e

         LXI    D,string   ; the offending
                           ; instructions
         DAD    D

         MOV    A,M
         ORA    A
         JZ     .g
         LHLD   .e
         INX    H
         SHLD   .e
         DCX    H

         LXI    D;string   ; here they are
                           ; again
         DAD    D

         CALL   g.##
         PUSH   H
         CALL   putchar
         POP    B
         JMP    .f

.g:      LXI    H,string   ; s = string
         SHLD   .d

; while (*s) putchar(*s++);

.h:      LHLD   .d
         MOV    A,M
         ORA    A
         JZ     .i
         LHLD   .d
         INX    H
         SHLD   .d
         DCX    H
         CALL   g.##
         PUSH   H
         CALL   putchar
         POP    B
         JMP    .h

; Done

.i:      LXI    H,0
.j:      XCHG
         POP    H
         SHLD   .d
         POP    H
         SHLD   .e
         XCHG
```

```
        RET
        EXTRN  putchar
        EXTRN  g.
        END
```

Thanks Richard for the sample code and I am glad you put the command line to generate assembler code in. My experience with "C" has taught me to always look at the assembler output. You can see some pretty ugly things. I guess the way to keep it from being so "ugly" is knowing which method or option to use. In this case pointers seems better. Thanks again. Bill.

Dear Bill:

My name is Larry Campanell and I am a Computer Hacker ...

After reading the last few issue's Reader-to-Reader, I felt compelled to join the ranks of Computaholics Anonymous. Here is my story.

The computer bug bit me back in '81 when I bought the bare boards for a computer called the LNW-80. This was a TRS-80 Model I clone with some significant extensions (e.g. hi-res graphics, color and a 4Mhz Z80). I almost bought the ZX80 (kit) but, decided on LNW due to the abundant supply of software available for the Model I.

As my computing interests grew, so did my LNW. First, 5.25" drives, then 8" drives, then a double density adapter, then a CP/M adapter, then a hard disk. All of the hardware modifications were done from scratch, either by wire-wrapping a daughter board or photo-etching from a layout in a magazine. The hard disk I used had a proprietary interface so, after building the host adapter, I then had to write the driver software. The LNW is very flexible. If a TRSDOS (or compatible - there were at least six Operating Systems for the Model I/III/IV family) disk was in the boot drive, it became a CP/M 2.2 machine. I still use the LNW, although it is more often used to copy information between various 5.25" and 8" disk formats.

At the '86 Trenton Computer Festival flea mart, I bought a XEROX 16/8 Professional Computer. It is very similar to XEROX 820, but it also has an 8086 daughter card. Like the LNW, it takes on the personality of its boot disk. Although this time, the choices were CP/M 80, CP/M-86 and MS-DOS. I don't use the MS-DOS mode very much as the machine is not PC hardware compatible. Only very simplistic text-oriented MS-DOS programs will work on it. However, with a Qume Sprint 11/55 Daisywheel printer and WordStar running under CP/M, it became my document processing workhorse (the LNW could have performed those duties but, whenever I really needed to write something, the LNW was usually in the middle of some sort of hardware modification).

The following year's TCF landed me an EPSON PX-8 (or Geneva). To me, this little 8.5" by 11" notebook that ran CP/M 2.2 was the best thing since sliced bread. All of the work I was doing on the LNW or XEROX could go with me anywhere I went! I could create my WordStar documents on the road and print them when I got home or access a BBS with its built-in modem. In fact, I'm using it now to create this letter while on a business trip. I get a few stares when I use it in public - after all, it doesn't do Windows. By the way, does anyone have a PF-10 they'd like to sell?

But alas, I eventually fell prey to the Big Brother mind set of computing. For, in the past few years I've picked up an XT clone and a 386SX system. Things seem to be different now. A "hacker" used to be someone who could crack the most sophisticated copy protection scheme or use his computer to verify that the light in his refrigerator really did go out when the door was closed. In the MS-DOS world, it seems a "hacker" is simply someone who knows how to manipulate *.ini files to get the latest version of a multi-megabyte application to run without crashing. Oh, I still get the urge to do something "exciting" and I'll find myself wire-wrapping a SCSI host adapter for one of my CP/M machines or writing some code to display GIF files on the LNW. As you have said in many of your recent *TCJ* issues, the older "classics" are very good tools for learning how a computer works. Has anyone interfaced a CD-ROM to a CP/M machine?

I thank you for letting me tell my story. As with any group therapy, the first step to recovery is publicly admitting you have a problem.

Sincerely, Larry Campanell, Blue Bell, PA.

Thanks Larry for your confession and comments. I feel at times, especially when trying to find something in my rather large pile of computers, that professional help is needed (shrink or garbage truck?). However that is what TCJ is for, a place to find others with the same affliction.

The more complex the Clone machines become, the stronger and easier it is to see that my position is very valid. Do your learning and playing on the older machines, you might just learn something besides a new swear word.

Thanks for the good comments, Larry. Bill.

polyFORTH
by Charles Shattuck

I recently had the opportunity to get polyFORTH for MS-DOS. Over the past twelve years or so I've used a number of different Forths, starting with a commercialized FIG FORTH for my old 8 bit Atari, then a Forth 79 for an Apple][, F83 for the Atari ST and one for MS-DOS, and for the last four years F-PC for MS-DOS machines. I've written Forth assemblers and target compilers for three different embedded micro-controllers as well. I feel as though I've finally got the Forth I've been looking for.

polyFORTH is a product of FORTH, Inc., the company started in 1973 by Chuck Moore, the inventor of Forth, and Elizabeth Rather, currently president of FORTH, Inc. polyFORTH has 21 years of development and use behind it, which must qualify it as one of the most mature MS-DOS development systems around. Let me try to explain why I like it so much.

I learned to program in Forth. Well actually I used BASIC for about three months before getting frustrated and temporarily giving up. Then I read about Forth in some computer magazine and I got one for my Atari 800. At that time what I really wanted out of computers was to learn how they worked and to be able to control one myself. Forth was the answer for me because everything is out in the open and understandable. Forth is so simple that it is possible for a single individual to understand everything that happens in it. It's not necessarily easy, but it's possible. Even more so in polyFORTH because the extraneous details have been weeded out and the written documentation is truly excellent.

If you know Forth at all you have probably heard the arguments between those who like BLOCKS and those who like TEXT FILES. I like blocks. I learned on blocks so I'm biased, but I also love the idea of simple and portable virtual memory. For those who don't know, blocks are 1024 byte packets of data on disk which are transparently brought into main memory and then stored back on disk. They can be used as 16 lines of 64 characters for source code editing purposes or they can be used to store data base records. Each block is referenced by number so for example "600 BLOCK" arranges for block number 600 to be in one of the block buffers in main memory, and the address of the block buffer is placed on the stack. "600 LIST" would print the block on the terminal screen as 16 lines of 64 characters. The polyFORTH editor is a combination of text commands and cursor and function key commands which work on a LISTed block.

F83 has a very similar editor which I used for several years, but this one just feels better. I especially love the way the top of the screen freezes when you LIST a block and your commands scroll by on the bottom part of the screen. I often want to refer to source code while debugging and now that's automatic. In F-PC there are commands in the editor to do the same sort of thing, but it probably takes 10 or 12 keystrokes to set it up and just as many to undo it. In polyFORTH to go back to full screen scrolling you simply type PAGE.

There were actually two features that really make polyFORTH stand out for me. The first is the multi-user operating system. Both F83 and F-PC are multitasking single-user systems. It is easy to define background tasks such as print spoolers but it is not easy to attach terminals and multiple users. polyFORTH was designed from the very beginning as a multi-user system and it shows. polyFORTH is reentrant so that each terminal task can share code with the others. Block buffers are shared by all tasks, but pains have been taken to avoid conflicts via facility variables which are similar to semaphores in other systems. I was able to configure and start using a terminal via COM1 in just a few minutes. Example code is included to support serial multiplexers that allow up to 16 terminals to share a single interrupt. The task switching is so efficient that friends report having had four programmers working on a single PC-XT without bogging down. Sixty-four terminals serviced by a single 68010 has been reported as well. When a task is asleep it only consumes a single instruction cycle in the round robin polling loop. In general if a terminal task is awaiting I/O, such as a keystroke, it is asleep. The keyboard interrupt wakes it up long enough to handle that keystroke and then the task puts itself to sleep again to wait for the next key. This is so simple and so efficient yet no other Forth I've used does it this way.

The data base toolset included with polyFORTH was designed, as was the rest of polyFORTH, for use in realtime systems. When a tradeoff must be made between 'convenience' and efficiency, efficiency wins. Even so the tools are also pretty convenient. There are words for defining 'FILES' which are named regions of contiguous blocks. There are words for defining 'RECORDS' which are fixed in length and fit into the blocks within a 'file'. There are 'FIELDS' which are offsets into records and there are access words which fetch, store, and display fields of different data types. There are words which help manage ordered index files and there are words which allow chaining of one record to another, possibly in a different file. Chaining allows you to get around the fixed record lengths by tacking on extra storage for records that need it.

I've wanted to write a book checkout and library cataloging program for a long time. I worked in a library for nine years before becoming a professional programmer, and now I believe I have the perfect set of tools for the job. With an efficient data base and multiple terminals on cheap PC hardware I should be able to produce an excellent system at a fraction of the cost of the popular mini-computer based systems on the market. If anyone is curious maybe I can write some articles about that in the future. One feature that might be of interest to TCJ readers is using classic computers as smart terminals. A lot of schools have Apple]['s around for example, and an average school library could probably be handled by a PC-XT, AT, or 386 and two to four Apple]['s as terminals. Other topics of interest might be the included target compiler, which can produce ROMable headerless code and the polyFORTH assembler, which seems to be an attempt at making assembly language more portable across different processors.

Thanks for the mini article Charlie. What you didn't mention is that you bought this on a special offer. Normally polyForth costs $995, but for FIG members they made a special offer of $295 to the end of May. Now this is the complete polyForth system, with all source code and manuals.

I asked Steve Agarwal, Forth Inc.'s (800-55-FORTH), sales manager if they would continue this offer beyond the end of May. Steve said he might, but would have to consider it further. One problem is that Forth Inc. is NOT set up to sell software to the general public so to speak. Steve is the ONLY person taking orders, as they normally deal with coporate purchases. However, if enough people call maybe we can get them to release some of their older, but still useful to us, products at prices normal people can afford.

Thanks again Charlie. Bill

"Year 19 and Going Strong"
by Ron Mitchell

They thought we were all dead. We proved them wrong.

Each year the organizers of the Trenton Computer Festival are tempted to drop CP/M and Z-System from the list of events, and each year it appears once more.

As I left Ottawa by car on Friday morning April 15 for the 500 mile journey southward to Trenton New Jersey, I wondered if there was anything left to journey southward for. There had been more talk during the previous week about people not coming than anything else.

Oh me of little faith!

Here are my impressions of a truly remarkable computer event that has been going strong for nineteen years.

The Trenton Computer Festival is a two day event held in mid April each year and sponsored by the Mercer County Community College, the Amateur Computer Group of New Jersey, Trenton State College, the Central Jersey Computer Club, the New York Amateur Computer Club, the Philadelphia Heath User Group and others. Over the years it has acquired a reputation as something of an eastern 'Oshkosh' of computing. Like the mammoth fly-in held each year in Wisconsin, Trenton draws people together who only see each other once a year - at Trenton.

This year's edition held on April 16 and 17 was no exception. It featured a wide array of topics designed for computer enthusiasts of every stripe. Included was everything new and old. National exhibitors were out in force, Aldus, Apple, Borland, C/A, Cobb, Corell, Microsoft, Wordperfect and many more. The list published in the Festival's printed program showed no fewer than 21 major computer players on the national and international scene. Each of these corporations staged a booth to display their wares.

The talks and forums spanned both days. Subject, no matter what it was, found a slot somewhere in the weekend proceedings. There were beginner's sessions on Lotus 123, databases, windows, you name it. There were user group sessions for Unix followers, Amiga followers, Mac enthusiasts and even for the Apple II. Despite rumors to the contrary, there was a CP/M and Z-System conference. As always the sheer variety and diversity of the sessions offered made Trenton a place for everyone.

This year's guest speaker was Mr. Steven Levy, author and Contributing Editor for Wired and MacWorld Magazines. Mr Levy spoke on the "Revolution of Look and Feel" during his Saturday afternoon address and the "Coming of Cryptoanarchy" at the evening banquet. The biographical material provided notes that Steven Levy "lives in New York City and western Massachusetts with his wife, son, and six computers.

Probably not a CP/M machine among them!

These computers have long since been left to a small but dedicated group of users who remain firmly convinced that small is beautiful and that writing efficient and compact code is still a desirable thing to do. In room MS-170 of the Maths and Sciences building at Mercer Community College there gathered about 15 (give or take) CP/M and Z-System supporters for a few hours of friendly chit chat and comparing of notes. They soon discovered that there is still development in the Z-System world that is worthy of note.

Hal Bower described the latest efforts of the ZSDOS development team as work on the BP BIOS (BP= Banked and Portable) continues. Over the past year there has been the addition of NZTIME, a modification which improves the accuracy of the timekeeping routine. The previous version lost time badly during disk read/write operations. The modification reads the clock card instead of depending on interrupts.

The introduction of directory hashing has speeded directory access noticeably. 'Hashing' is a technique for searching a list of items that involves picking a spot before the item being searched in an attempt to get as close to it as possible without actually passing it.

Hal went on to state that the addition of a 1.7 Meg floppy capability lies in future plans for BP-BIOS. This improvement will be possible through the use of Jim Thale's I/O board.

For those not familiar with the enhancements provided by Z- System, it should perhaps be noted here that the performance of a 64K CP/M compatible computer can be significantly enhanced at very reasonable cost. Z-System provides a powerful console command processor (CCP) adding additional packages of capabilities such as the resident command package (RCP) and the flow control package (FCP). More interesting is the concept of the TCAP or Terminal Capabilities feature which makes it unnecessary to install Z-compatible programs for the requirements of different terminals. Once installed, the system is capable or reading the TCAP and adapting program operation accordingly. Z-System also provides for the use of "Alias's" which allow the user to develop system commands based on short, easy-to-prepare text files.

BP-BIOS brings a more efficient usage of banked systems to the user and allows significantly improved capability in terms of the number and size of hard drives that can be operated by the system. Presently Jim Thale is using a 200 MEG Connor drive with his development system, and Hal sees no reason why hard drives sizes could not approach 400 MEG.

Hal noted some other news from the Z-System world, some bad some good. The bad news is that Bridger Mitchell, long time member of the ZSDOS development team and author of such notable Z-System programs as Backgrounder, has announced that he will no longer be active on the Z scene. The good news is that Bridger has agreed to turn over much of his work to Hal who is now making sure that we secure as much of it as possible while it is still available.

Also positive news: as announced last year at TCF93, Jay Sage reconfirmed that Z3Plus and NZ.COM are to remain available for the incredibly low price of $20.00 US each. This represents a real deal from SAGE Microsystems and should immediately be snapped up by anyone interested in improving the performance of their CP/M system. Z3Plus is for CP/M 3.0 systems and NZ.COM is for those with CP/M 2.2 machines. Either way, you can't beat that value.

CP/M-ers continued to meet throughout the day on Saturday in between trips to the flea market being held adjacent to the main show and the many sessions being conducted inside. At any moment in the conference room you would be likely to run across Jay Sage, Hal Bower, Bruce Morgan, Blair Groves, Ian Cottrell and Howard Goldstein. These people have each in their own way made a consistent contribution to the CP/M and the Z world over many years. Listening to them chat for an hour or two is an education in itself.

Later Saturday evening there was continuance of a tradition well established by the CP/M attenders over the past few years. There is a banquet held as part of the main Trenton Computer Festival. It has become the custom of the CP/M group to hold an informal evening of its own apart and separate from these proceedings. What these people lack in numbers they more than compensate for in what amounts to a sinful ability to consume pizza. Those taking their nourishment with 'dead fish' somehow manage to consider themselves a cut above the rest and will hotly compete for the title of 'Greatest Glutton'. I was there, but I do not know who won this year's contest. Some say it was Lee Bradley but reports have it that Lee cheated.

All I know is that I was not part of any of this. I prefer my pizza without green things and anchovies.

Following this massive pig-out, the balance of Saturday evening was social and also according to tradition. Back in the party room at the Stage Depot Inn a few more local CP/M-rs joined us and there was the usual round of introductions where we each said where we were from and what we were up to in CP/M. Ian Cottrell managed to pose his annual answer less question:

If one synchronized swimmer drowns, do all the others on the team have to drown too?

And then there was the one that he intended to ask but didn't:

What was the best thing before sliced bread?

Both of which top last year's answer less question which was:

Who brings baby storks?

Think about it. No doubt we shall be pondering these eternal mysteries until next year when more will be added.

The 1994 Loonie award for outstanding contributions to the CP/M community went to Hal Bower for his work on BP-BIOS. Congratulations Hal! You deserve it for your many years of work on our behalf.

The Loonie Award? Well, this is also a tradition. Known by only a few in the CP/M world, this coveted award was hand designed and built by Canadian CP/M-ers in an attempt to export Canada's Loonie dollar coin into the United States. The trophy is essentially a Canadian one dollar coin perched precariously atop a polished wooden base. Each year Ian Cottrell fights through insidious urges to award the trophy to himself and consults with a committee of his peers to choose a CP/M hobbyist who has made a significant contribution.

Hal's contribution has been quite significant. He gets to keep the Loonie for the next 12 months.

On Sunday, the second day of TCF94, I dropped my remaining cash at the flea market which was as impressive this year as ever. Despite have been deluged and almost blown apart by early morning rain and wind on Saturday there was no shortage of bargains on the Sunday. For many of us the flea market is the main attraction, providing as it does a few history lessons in computing as well as many incredible bargains. There are quite literally acres of displays, tables, equipment, and things to tempt your wallet.

Notes for next year:

Bring a cart
Bring a rucksack,
Bring a truck,
Bring an umbrella.
Apply sun screen even if it is raining.
Stay away from the Flea - Not!

Find out when TCF 95 is and book time NOW!

Thanks for the mini article Ron. Herb Johnson was there and reported on his experiences which seemed to be very different from yours. I am glad both of you reported on it. Now all I need to do is get more people to let me know about these events early (so I can send flyers with you) and make sure I get more than one report!

By the way Ron, it seems like you have a good grasp of the problems beginners have faced getting started on ZCPR. Maybe you would be interested in writing about them. Hope so and thanks! Bill.

We need articles on subjects that are of interests to our readers. Those interests now span small and older eight bit systems, through the obsolete IBM PC/XT style of computers.

Projects which use surplus parts available from current vendors, showing how to debug and develop the needed knowledge of the used system, is of interest to our readers and advertizers as well.

Send your letters to:

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

The European Beat

by Helmut Jungkunz

Regular Feature

All Users

AMSTRAD

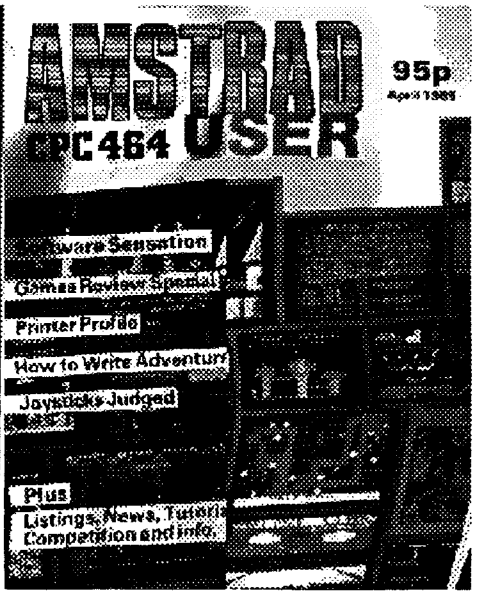
More Sugar for Computers (More about AMSTRADs)

Well, last time I told you the basics about AMSTRAD. I presented the three different kinds of Game-Computers (Colour Personal Computer CPC), the 464 (cassette drive integrated), the 664 (3" disk drive integrated) and the 6128 (128K RAM and disk drive integrated, CP/M-Plus too). This time I want to shine a little more light on the other side of the business. What kind of software was there in the beginning, where did it come from, and what besides games was available (outside of CP/M)?

There was a company in England, Eden AMSOFT, that produced quite a palette of games for the AMSTRADs, even if some of them may have been not too inventive. They were still doing business with AMSTRAD recently at the introduction of the AMSTRAD Notepad. A small company in Spain tried to convince Alan Sugar that they would be their ideal central distributor for Spain. Although in the beginning AMSTRAD did not believe this, they did business with them because the company could produce excellent game software, including INDESCOMP -- a name already known from other shores of the Z80 market. There were lots of others. There were games, music packages, and office bundles -- everything.

The central distributors in France could only sell what they received from AMSTRAD, but all outside-UK distributors -- whether in France, Spain, or Germany -- made incredible sales in a short time. AMSTRAD France, for instance, sold about 291 million francs worth of AMSTRAD gear between 1985 and 1986, and the German division, Schneider, sold 32.5 million pound sterling of CPC between 1984 and 1985. However, their price for the "Personal Computer Wordprocessor", the PCW, was totally disagreeable to AMSTRAD. Where AMSTRAD had expected a price equivalent to 399 pounds, Schneider charged around 700 pounds! Besides, all of AMSTRAD's efforts to sell the machine as a text processing device seemed to be ignored by Schneider; instead, they sold it as a computer, in a market niche that was already pretty narrow for AT-clones. So many AMSTRADs were sold that it is no wonder that they form the heart of the European Z80 scene to this day. This probably made Commodore pretty sick at that time, especially when a computer magazine's benchmark revealed the inferior performance of the C64 floppy compared to the CPC tape drive! (The C128 was not much better than that.)

There were quite a few magazines for Z80 computers at that time. In England there were magazines that originally covered the Acorn BBC computer and the Sinclair line but soon picked up the AMSTRAD and started a regular series about them. Some had a hard time, since AMSTRAD ran it's own mag and had funny ideas about others publishing info on "their" computers! You could, for instance, join the AMSTRAD USER GROUP (UK) when buying an AMSTRAD computer and



receive the house magazine "AMSTRAD 464 USER" as a club news magazine. In Germany the magazine "CHIP" was one of the biggies then and always allowed easy comparison of quite unequal machines. Another one, "c't-magazine", described the AMSTRAD machinery on a very sophisticated level. But the most important two for AMSTRADs were "Happy Computer" by Markt&Technik and "CPC-International", where in Germany the word "international" normally is the biggest lie you can read.

The latter two magazines not only described the functional layout of the CPCs (and the PCWs also) but also presented lots of listings that one could either type in or order by disk from the editors. Even hardware projects were initiated by them. This is were I came in, by the way.

I myself never had anything in mind with computers at all. I am a sound engineer (as in acoustics) and liked to play a little music myself then. My girlfriend, her cousin, her cousin's boyfriend and I rented a big flat (apartment to you yanks) to be able to have nice, voluminous rooms at a moderate price. After some time, as life goes sometimes, the other couple broke up, and we had a vacancy for a room. We put up an ad, since we couldn't afford the place ourselves, and a guy moved in who happened to bring along a Sinclair ZX81. On that cutey I played with BASIC and fell in love with the super-primitivo flight simulator, just too good!

Unfortunately, shortly afterward we had to give up the whole place, due to a very annoying affair with the company that was originally renting out the house. They sold the building and urged everone out.

After a while, I met the computer guy again. He had gotten himself a job in the editorial department of "Happy Computer" and advised me not to buy a Sinclair, as I had wanted to, but to wait and buy the new Schneider CPC (AMSTRAD's German distributor put their name on it at that time). This I did and soon got into a discussion about a colour modulator for the CPC to connect it to any TV-set. Since I, too, had the green monitor on my CPC (464), I was very interested. My special friend (Andreas Hagedorn) told me that several people had already tried and that it would be either too complicated or too expensive to build the units others had developed. I thought a while and said that I could possibly build one for very little money. I had the restriction to use only parts available everywhere, so I had to compromise, but I went to work.

After six weeks, the layout was not only ready but tested, photographed, and written about in a very long article describing everything, including the parts list, the operating principles, and the methods used to adjust the circuitry. When this appeared in "Happy Computer in September, 1985, it was a real success for them. It opened many doors for me, and I sort of became a specialist for everything having to do with video. I wrote several articles on monitors, RAM disks, and other stuff for "Happy Computer" and went to computer fairs with them, where I demonstrated their software (remember, M&T -- Markt&Technik -- was the company selling all the CP/M products for the AMSTRADs). Thus I got all the software for free. This, again, enabled me to contribute knowledge to our club, SCUG (Schneider/Amstrad/CPC User Group).

Once I was asked to test a RAM disk by a newcomer company (FECH & OTTEN). It turned out the product was so poorly designed that it was useless, so I used to refer to it as FRECH & ROTTEN (frech is somewhat like fresh in that respect). I stated my opinion so clearly in my article that it never got published. CPC-International, the rival magazine, published a very euphoric test on the same thing. I was puzzled! But -- I only met one living person who actually bought that RAM disk, and -- boy! -- was he stuck with problems with that device!

Meanwhile, in England, a company called dk'tronics sold their RAM expansion units, which were pretty well behaved in the sense of AMSTRAD compatibility but showed some strange bugs due to a hardware error in their "Operating System" unit design (the device was split in two parts, a basic O.S. part and the expansion RAM).

Shortly after this, I was given another RAM disk for testing. I plugged it in, followed the usual intructions to the point -- and was impressed! That developer had stuck completely to all the AMSTRAD routines, behaviours, and BASIC RSX conventions, and everything worked like a charm. Oh, you don't know what RSXs are? The abbreviation RSX generally stands for Resident System eXtension and is sort of a software implementation of a new command into your existing system. AMSTRAD CPCs allow for this RSX technique even from within their excellent LOCOMOTIVE BASIC. I wrote to the developer of the RAM disk for more information, and he called me back on the phone. We had a few long chats, during which some of my ideas flowed back to him, and I offered support for people who had bought M&T's WordStar, dBASE, or Multiplan for the CPCs with Vortex RAM disks. Mind you, they had a nasty patch in them, in order to be able to use German umlauts, loaded from a routine in the VORTEX RAM-disk BIOS. Due to that patch, those programs, of course, would not run on any other machine then. So the patches had to be removed. Still, some problems were left, and so both my address and our club address were put onto every disk that came with the DOBBERTIN RAM disk.

The DOBBERTIN RAM disk enables the CPCs to use a 63K CP/M-2.2 TPA! The only disadvantage lies in the BIOS of the CPC itself. CP/M-2.2 uses a standard method for calling BIOS routines that is different from that in CP/M-Plus. These standardized jump addresses are often referred to as system vectors. The CPC's CP/M-2.2 system vectors are at a very low address, so they would be in the middle of the TPA if you stuck to AMSTRAD's conventions. Whereas Vortex had patched and repatched their CP/M to be able to use those system vectors, DOBBERTIN had disabled them, since the reliability of such a situation is pretty bad. The complete CP/M-2.2 fit into the system tracks, so no extra BIOS file had to be loaded. This may seem unimportant -- but only for those never stuck with a disk capacity of only 169 K!

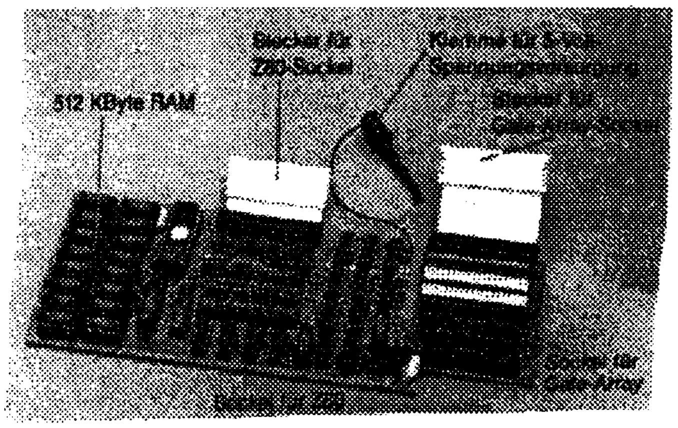
There is a very nice side effect with that RAM disk. When you have a minimum of 128 K RAM, you can run CP/M-Plus (with a teeny-weeny patch from DOBBERTIN electronics) on both the 464 and the 664! Beautiful! I met the DOBBERTIN people (father and son, hardware and software) at one of the fairs, and we had a long talk over dinner. I convinced them that a good hard disk for the CPC was not only needed but could be sold in numbers. Vortex had had a hard disk out, but didn't do too well, due to problems with the BIOS and failure to stick to AMSTRAD conventions. They also suffered from a "court call" by AMSTRAD, accusing them of stealing their BIOS and modifying it.

After a short while, the first DOBBERTIN hard disk came into my hands for testing. As I had expected, you just plugged it in, and there you had a hard disk in BASIC, CP/M-2.2 small, CP/M-2.2 with 63K TPA, and CP/M-Plus in a modified BIOS, loadable from either an extended system track or a BIOS file.

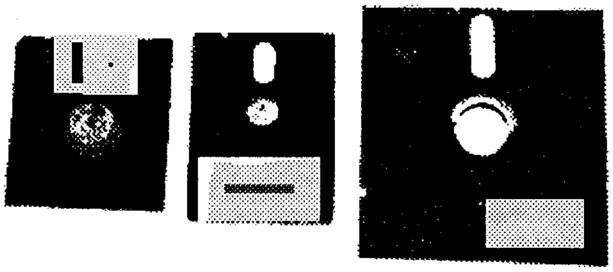
So, just for the fun of it, I put a different, bootable section on each of the four 5 MB partitions (20 MBs is the only size they make for CPCs). Needless to say, with an SYSCOPY command you could exchange all CP/M-2.2 boot sectors with any floppy disk. The CP/M-Plus boot sector, though, was too big for a floppy system track in one of the CPC disk formats. And this is another interesting point here: since there are so many CPC programs out that want to see the typical CPC-type sector IDs on floppy disk, DOBBERTIN chose a system format of 80 tracks double- sided, the sector IDs following the CPC's rules.. Since the VORTEX disk format had already become a sort of standard, DOBBERTIN accepts the VORTEX-format via AUTO-LOGIN in CP/M-2.2 or by pressing CTRL-C. This doesn't apply for CP/M-Plus, but, nevertheless, it is possible to read a VORTEX disk in CP/M-Plus as well. For easier handling, I forced some of our club people to help me soup up a quick format switch program, so that within DOBBERTIN CP/M the various disk formats could be changed: DOBBERTIN System (DS80), DOBBERTIN Data (DS80), VORTEX System (DS80), B360K (SS80), CPC System (SS40 on a DS80 drive).

In CP/M-Plus, the login programs for the CPCs and the PCWs are very much the same, except that the side-bit information on the CPC starts with a "0", whereas the PCW wants a high-bit "8". So the disk parameter block would be 01 for a double-sided CPC drive and 81H for the PCW. Some of the programs for PCWs are written with absolute addressing, so they would only run with one particular machine configuration. Again, members of our club helped to get this mess straightened out. Now those programs use BDOS calls to get the address for the DBP (disk parameter block) and use the standardized BIOS call to properly jump to machine routines. In my high CPC times, I had my CPC 464 equipped with the RAM disk and the hard disk from DOBBERTIN, a second drive (5 1/4) from VORTEX, another 5 1/4, switchable as A: or B:, and a Hitachi 3" drive that allowed for a 2ms steprate.

As you might have guessed by now, my CPC times are almost over. If I run CP/M, then I use my CPU280 from Tilmann Reh, about which I will write next time, when I'll describe the vast market of home-brew and kit computing in Germany.



VORTEX RAM expansion Card


Floppy disk size comparisons, standard 3.5 inch, Amstrad/ Hitachi 3.0 inch, and standard 5.25 inch.

Mr. Helmut Jungkunz.

Regular Feature

Intermediate

Trenton/Z-Fest & Letters

Dr. S-100

By Herb R. Johnson

Trenton Computerfest

For many years the Trenton Computerfest has been the largest computer show and flea market on the East Coast, if not in the country. It certainly is one of the oldest and best-known. This was the first Trenton show I attended in several years, and the first one where I had a spot in the outside market. Although there were many exhibits and talks inside, I was too busy outside to go in, but I would recommend many of them to TCJ readers as they included Z-system and CP/M subjects. However, the only "classic" vendors were outside.

There were about 500 outside vendors in the two small parking lots, doing business in tents, trailers and trucks. Maybe two-thirds of them were selling from their car trunks and a fold-up table as I was: those are the sellers I was interested in. They would have the old and odd equipment I wanted: more to the point, they would not be selling at dealer prices! They would just want to "get rid of the stuff" or "see that it gets to a good home." Oddly enough, many of the dealers with big rental trucks were also "getting rid of stuff": this was their annual basement clean-out of instruments or machines that they couldn't sell mail order, or trade-ins from their customers.

Another factor that drove down selling prices was the weather. As is traditional, it rained. This year the downpour was during the first morning (Saturday). For those of us in the know, this is a BUYING OPPORTUNITY and I took full advantage of it. I left all my stuff in the car and sloshed through the isles, peering through the water lenses formed on the plastic tarps that covered the folding tables. I found one guy who works for some instrument company: on his table was a stack of oscilloscopes that I recognized as valuable. He was hiding in his car. I knocked on his window and he rolled it down. I asked what was the price range on his 'scopes. He said "$75 to $250 dollars." I was startled. "I just want to get rid of the stuff...I'm not going to be here tomorrow." I asked about the most expensive, an HP 100MHz storage scope. "$250." "Would you take two-twenty-five?" "Look, it's an $800 scope" I couldn't argue with that and gave him the money. I lugged it to a nearby building to test it and all seemed to work, after I figured out all the controls.

I had hoped to buy an IMSAI (S-100 system with the familiar red and blue front-panel switches) as they usually surface at these flea markets. I almost had one when the person behind me raised the price before I could close the deal!! Turns out to be one of my customers, who bought a Xitan (Technical Design Labs) system from me some months ago. I was a little more disappointed when, later in the day, someone asked me for a "cheaper" IMSAI system. I spent a few hours late that evening putting a system together, using cheaper but reliable cards. The next day he hesitated, hoping to find a better, cheaper system "at the other end of the lot" so he said. At the end of the day he came back and offered about two-thirds what I asked and he was surprised when I declined his offer.

So what did I get? Well, I picked up a Heath/Zenith H89 (Z80 system) and a Heath/Zenith Z100 (S-100 with 8085 and 8088) for about $100 total; a few sets of S-100 cards; and a few monochrome composite monitors for $10 or so. I sold an old Apple II+, an AT&T 7300 Unix system, and a Wyse 50 terminal among other things.

Z-Fest

The Trenton Computerfest is a traditional time and place for the East Coast Z-System people to congregate. Although I am not a "Z-person," I certainly am partial to their processor of choice, the Z-80. Since I share this magazine with several of these authors, I took the opportunity to see them at dinner. A few phone calls and misdirections later, I found about fifteen of them in the back room of a pizza parlor, competing for eating the most slices of anchovy pizza (ick!). Late as I was, I declined the offer to compete and cleaned up the less desired pepperoni and Canadian bacon slices. After dinner, we retired to the nearby hotel for discussions. Ian Contrell was the host and toastmaster, conducting introductions in the circle we formed around him. Bruce Morgan, Jay Sage, and many others were there: I did not take notes so pardon my copious omissions. Perhaps the copious availability of beer was a factor.

One attendee has a project in the works that the Dr. (I) would prescribe to you: a "universal" Z80 interface. Simply put, it is a daughter card that plugs into a Z80 socket that includes an 8255 3-port parallel I/O chip. The designer says this would offer a standard bit of hardware for others to design software around. I encouraged him to complete the work and write it up for TCJ: I hope he does so.

Another subject: S-100 IDE hard drive interface

I have just received the PLA (Programmed Array Logic) devices from my Australian colleague that he designed for an Z80 to IDE interface. As he prototyped them on an S-100 system, they could be a product if such makes sense. I will be considering the completion and support of such a card very soon, probably before you read this article. I anticipate the cost to you for a card, with software to integrate into your BIOS, would be about $150. Why "so much"? Mostly the cost of chips and board. I have to anticipate other costs, including printing and shipping, and some development costs.

I would be VERY PLEASED to see a show of support from my readers. Is Dr. S-100 writing a prescription for a cure when there is no disease? Or is the "cost of treatment" worth it to YOU? It is important to let me know. Even a few letters tells me that there are many more people who haven't written that would be interested. I should tell you that, in the Heath/Zenith Z100 world that a similar interface for SCSI devices sold for $260 just a few years ago!

I also need a bit of reader consideration on the following questions: should this card have a floppy disk controller interface? It would add about $30-$40 to the cost. Should it have a serial interface, for a "plug and play" BIOS? That would add maybe $25 or so. And, what machines would you run this on? IF they are all 4MHz Z80 systems, then I don't have to worry about timing problems: if you want this on your Altair 8080, then it could be a problem. The Doctor needs your input!!

LETTERS:

Amstrad vs. QX-10, DR's GSX graphics standard

Got another letter from Roche Emmanuel from France: "I sent you a copy of (the last?) Windsor Bulletin Board User Group for you to mention it and its contents, not to mention my name." Oh well.. Roche continues to discuss this group, the impact of the Amstrad PCW's in Europe, the QX-10, and Digital Research's GSX graphics standard. Z-system people may want to read this letter closely: can a comparable graphics standard be ported to Z-system?

"WBBUG was the successor of the "CP/M User Group (UK)" [United Kingdom], the biggest such group in Europe in its heyday. Its Journal was wonderful but never regular. Unfortunately, when Amstrad launched its CP/M micros (which were incredibly successful here: more than 1.250 million sold!), this group was silent for more than one year....just at the time of the biggest arrival of newcomers. As a result, five magazines were created just to advertise for those.

Personally, I am quite knowledgeable about the PCW: I keep disassembling its BIOS, from time to time, to be able one day to fit four 800K drives. But my personal favorite is the Epson QX-10, which is almost unknown in Europe. It was even more expensive than an IBM-PC back then! In my opinion this is the best non S-100 Z80 microcomputer ever made. [Thanks for the distinction!] For example, it is consistently 10% faster than the PCW, as the PCW uses one large ASIC to decrease the number of chips on the motherboard. But this ASIC [a large custom integrated circuit] slows down everything. By contrast the QX-10 is a forest of chips, as NEC engineers did everything at that time to speed it up.

Lastly, the big advantage of the QX-10 is its graphics display controller, the NEC uPD7220, which except for three CP/M microcomputers, was the standard of the first UNIX workstations. For example, this chip can draw 128 squares on the screen faster than you can see them drawn...really amazing!

Which brings us to GSX. GSX from Digital Research was a microcomputer implementation of a mainframe graphics standard. It was the most powerful, by a huge margin, graphics system ever made to run under CP/M. [Would any of the Heath/Zenith users care to comment on this?] Unfortunately, it was only sold by Digital Research Inc (DRI) for one year, before they jumped into the 16-bits wonderland and almost disappeared with Concurrent DOS. (Two years later, Amstrad sold its CP/M microcomputers by the millions and saved DRI from bankruptcy.)

The Amstrads were sold with the latest versions of all the DRI CP/M products: CP/M Plus, GSX, CBasic 2, Pascal MT+. They also sold DR Graph and DR Draw, the only GSX application programs owned and sold by DRI. Two other packages, DR Kernal and DR Plot, which allowed a programmer to use portable graphics, were not sold. A big advantage of GSX is that it works with device drivers. I have managed so far to find 22 (mostly printer) drivers. So, all my GSX programs, without any change, will produce outputs on 22 devices! And, the resolution was so high (32K X 32K) that a screen only gives a small idea of the drawing that one is going to obtain from the printer. I also happen to have a color QX-10 (640 X 480 X 8 colors, four times the Amstrad...) and I also have an HP graphics plotter and its driver: this time the resolution is numbered in several thousands of steps! When one sees a color plot made on this plotter, it is impossible to imagine that it was made on a CP/M microcomputer!

After closing out Concurrent CP/M and CP/M -86, DRI released some source code for GSX-86, the 16-bit version. The sentence you quoted [??] was written because some of the remaining CP/M programmers understand that GSX was a powerful graphics system, and some would like to use it more. Unfortunately, it is no longer supported by DRI, which did not release its source after leaving the market. Besides, DRI is dead now. [This is not strictly true: they are a part of Novell. But they are not supporting any CP/M products. David McGlone of Lambda Software has a license from "DRI" for CP/M-80.]

So, some "lone hackers" continue in Europe to use and maintain GSX. I happen to know that one such person, well-known among the former members of the CP/M User Group (UK), has managed to disassemble the biggest GSX printer driver for the Epson FX-80. This 16K driver produced a full 111 page long disassembly. This source is not yet in the CP/M Software library but it will be by July-August, after the last check-outs. I forgot to mention that CP/M User Group (UK) did produce a library, which was continued by the WBBUG. 122 volumes have been produced so far, about 30 Mbytes.

For any information about WBBUG and its CP/M Software Library, write to:

Peter Catley
11 Haslemere Road
Windsor
Berkshire SL4 5ET
England

Whew! The Doctor can talk the talk, but it's people like Roche that "walk the walk" and do the real work in the CP/M community. Thanks for this highly informative letter and the reference to our British colleagues and the European market.

An S-100 system "kit" (no instructions included)

Mike Michels of Canton IL brings a number of "patients" to the Dr's examining table. As usual my immediate comments are in []'s. Unfortunately these patients are not responsive....

"First of all, I want to thank you for your Dr. S-100 column in TCJ. I read and re-read these many times and have it as a handy reference source. I especially like the way you have alternated in your coverage of CP/M and S-100, and how the two are interwoven. [Thanks!] You have clarified several area that I have had confused for a long time.

"Not being a 'techno-wizard'' but rather a self-learner, I have a strong interest in the way computers and their components work, and how they can be configured to interface with just about anything. As a "wannabe" tinkerer, I appreciate your patience and willingness to help those who are now just starting out learning about CP/M and S-100 systems. It seems (even with the various specialized system support publications) that they assume their readers know more about their systems than I do.

Would you please provide a listing of some useful books that would explain about the S-100 system? I have obtained one and would like your assistance in helping me to build this into a fully operational system. However, not knowing what I have nor what I need this is going to be a long-term learning experience for me which I hope to pass on to my children as well as others. Perhaps we can make this an article if TCJ readers are interested [!]: something about what I can do with these cards or how we build these components into a working system. The cards are:

Compupro RAM XX [this is a static memory card]
Cromemco Blitz Bus with 21 slots [an S-100 backplane board]
Cromemco 8K Bytesaver II card [2708 EPROM reader/writer card]
Cromemco TU-ART card [dual UART (serial) card]
Cromemco ZPU card [2/4MHz Z80 processor card]
Morrow Disk Jockey @D/B card [8" floppy controller w/serial port]
SD Sales Expandoram card [dynamic RAM card]

The person I bought these card from had them for a potential project but had never turned them on (or so he said). He had no idea how much total memory it has, or even if they work. No docs, no power supply, no drives.

I've also acquired some parts of a "Four Phase" system with one keyboard, two video display terminals (both fire up ok) and several Diablo 630 printers [!!] I would like to have the specs to interface them all together as part of the "system".

The Diablo 630 printers have a BNC connector for a network connection of some sort [this is correct]. I would like to interface these to a serial (RS-232) or parallel (Centronics) interface. I was directed to a place called 'The Printer Works': however, they have not responded to my letters in over two months. Enclosed are details of the boards and connectors in the printers. I will also contact Robert Grey: thanks for providing his contact info.

Thank you very much for any assistance you can provide in these matters.

A little hand-holding and consultation

Well, I can certainly see a collection of several Diablo 630 daisy-wheel printers and a few "crates" of computers as a legacy for your children! I vaguely remember the Four-Phase systems as an office system, but I don't remember much beyond that. You might look through back issues of Byte magazine for ads or even articles. This brings me to your request for "useful books". The most useful books are whatever you can get your hands on! It's rather unlikely (and expensive) to try to find computer books of the 1970's and '80's through a bookseller.

Your first resource is always the local public library, followed by the universities. Fortunately for us, computer interests were high in the late 1970's and many libraries stocked up on the books and magazines of the era. My strategy with a new library is to find the "computer" sections (there are usually MORE THAN ONE!) and to scan the titles on the shelves. You might get lucky and see a title reference to "S-100": but probably not, as few books were written exclusively on bus-based systems or the S-100 in particular. You will probably see a number of "CP/M" books, and some on particular systems. Pull them down from the shelf and page through them: read the table of contents, the introduction, etc. You'll probably have more luck at the University: if you ask a librarian there, you can probably get a "courtesy card" for a small deposit or fee for borrowing privileges. Generally anyone can read and copy books in a public university library; private universities may be more restrictive.

The prescription

As for your collection of cards: you have lucked out! Except for the SD Systems RAM card, you have a nice set of boards that should work together. Dr. S-100 has a few general recommendations that these cards happen to follow:

1) Try to stick with ONE manufacturer of cards, or at least cards of a similar vintage. In your case, the Cromemco and Morrow cards are of the same era, with the Compupro RAM XX a little later.

2) Generally, avoid dynamic RAM cards and use static RAM, subject to the first rule. That would eliminate the SD Sales card, I believe. What is this rule about? Simply put, the old S-100 dynamic RAM cards were not often built well, and were marginally designed at their operating clock speed. Also, there were no standards for refreshing the D-RAMS, so different manufacturer's methods could conflict. As I said in an older article, DRAMS can be recognized as "small" chips with 16 or 18 pins (there are exceptions) with numbers like 4116, 4164, etc. Static RAMS were generally wider 24-pin packages with numbers like 6116; or small packages with numbers like 2102, 2114, and so on. Dynamic RAM cards had more chips, and funny delay lines, or even timers and DRAM controller chips.

I should explain that "static" RAM's use a circuit for each bit that generally includes a couple of transistors that flip from one state to another, and which retain a bit value as long as power is applied to the chip. "Dynamic" RAM's use a "silicon capacitor" for each bit of storage, which must be "recharged" by reading it at regular intervals before the charge leaks out. D-RAM's are simpler, taking less chip space and allowing more bits per chip (for our era, about four times as many bits). Even IBM-PC's require a "refresh cycle" for their dynamic RAM's: that's what the "clock tick" timer on the PC initiates 18 times a second.

Procedures and instruments

There are other "rules", but let's not be too formal here. What you need, Mike, is a STRATEGY for bring up these cards! The first priority is INFORMATION: on your processor (Z80) and other IC's, on the cards, on S-100 systems, on CP/M. Your library is your best bet. A quick hint on IC's: catalogs from JDR Microelectronics and DigiKey will at least list IC's by number and function. Their addresses can be found in any electronics magazine. Again, I'll have manuals on the cards themselves (Readers: I charge for shipping and copying, by the way, it's WORK to save and maintain this info!)

Next are TOOLS: an oscilloscope, a voltmeter/ohmmeter, soldering iron and solder sucking device, the usual fine hand tools. A supply of PARTS, including IC's, resistors, etc. would be useful. From a system's point of view, you first need a working backplane and power supply; followed by a working CPU; then working RAM; then working I/O with a serial port and some kind of monitor or boot up process; and then a bootable disk. You happen to be in luck that I can provide you with manuals and even a bootable disk for these cards. I'll contact you in a separate letter with details. By the way, Readers, the Dr. DOES offer these services to you as well: write with a description of your system and its boards and we'll "prescribe" some docs to help you for a modest fee.

Let's start with the system basics: you need a power supply that gives you four voltages. Plus eight volts (+8) at several amps to be converted to the +5 on each card; a minus 18 and plus 18 volt supply (+/- 18 V) at an amp or so for RS-232 and the Cromemco PROM Bytesaver card; and a minus eight (-8) which is occasionally used. (Note: even the IBM-PC has a -5V pin, though it is hardly used!). Don't forget to have fuses on those lines, especially the +8 line! Confirm that you have the proper voltages on the proper S-100 lines. They can be a little higher with no cards on the bus, but under load they should not drop below +/- 7 volts and +/- 15 volts. Back issues with my articles, or references from the books you found, or markings on the bus, will tell you where to connect the power supply.

Specifics on the cards

Next, put the Cromemco ZPU on the bus, apply power and take a scope to see what is going on. Immediately check that the on board voltage regulator is doing its job: delivering +5 volts!! Then, look for signals on the Z80 pins, notably RESET, clocks, address and data. Then check the bus lines for similar signals, seeing that no lines are "stuck" when they should be bouncing between high and low. If all seems reasonable, power down and put the RAM card (Compupro RAM XX) on the bus: you may need to set some switches to get it to operate.

Use your oscilloscope to confirm address, data and control lines are operating. In addition, if the board is being addressed by the processor, the "chip enable" lines on the RAM chips will be toggled (switched from high to low on the oscilloscope). Without a front panel or monitor program, which I happen to know the Morrow DJ 2D does not have, you can only "watch the lines wriggle", but tracking even this activity will build up your confidence in your system and increase your experience in observing bus behavior.

Speaking of the Morrow DJ 2D: what you have is a floppy disk controller card that will operate two 8-inch floppy disk drives, single sided, at single or double density. Also, it has a boot up program on ROM that will start on power up; a "software" UART that will interface to a terminal; and ROM support for a CP/M BIOS that can be booted off a diskette. If you have enough memory on the Compupro XX, you can run a system with these two cards and the ZPU processor card! Repeat the procedure used on the Z80 and memory card, namely check the disk controller chip for reasonable signals, including the "chip enable" lines.

Addresses

I would encourage anyone who writes me to include their address and phone numbers, and any network mailbox address they might have. You'll likely get more help from another reader than from me!

Mike Michels, RR 3 Box 139, Canton IL 61520.

Regular Feature

68xx/68xxx Support

6800/09 and More

Small System Support

By Ronald W. Anderson

Vintage 680(X) Computers and Other Stuff

This time I am going to start out with a discussion precipitated by a letter from Tilmann Reh from Germany. His letter was a response to my letter that was published with my first column. I had made some comments about it being easy to talk directly to the hardware on a PC, indicating that I was able to get a serial port running that way after having had no luck at all through a BIOS call or the facilities of Turbo C.

He made some very valid points about it being much better NOT going to direct hardware control, working at least through the BIOS calls to avoid writing software that will run on one system and crash on another. The point is well taken. My "excuse" follows:

Realize that I have always been working with embedded control and measurement systems. The company that employs me builds balancing machines. That is, we build machines that balance rotating parts for many applications, for example, flywheels, crankshafts, fans, blowers, pump impellers, grinding wheels, saw blades, motor armatures and lots more. Some of our machines are very simple and some are multi-station automated transfer lines that balance parts "untouched by human hands".

Our balancers all use a microprocessor based computer to do everything after we amplify filter and digitize the unbalance signal coming from transducers on the machine. In the simplest ones, the computer does calculations and tells the operator how to balance the part. The operator reads the angle and depth of the hole to drill or the amount of material to add or remove to do the job. Of course the machine has to know the drill diameter, maximum allowed depth and a lot of other information. Things get complex very quickly.

Of course, we have competitors. Rather than try to make our code portable, it is to our advantage to make it somewhat hardware dependent in order to make it non-portable. It is also difficult NOT to make embedded control systems hardware dependent. Should our machine code (the output of a compiler) fall into our competitor's hands, they would have to duplicate our hardware in order to use it.

I hope we all realize that disassembling someone else's machine code, particularly if it is the output of a compiler, would probably be more work than writing a program from scratch. However, a competitor might disassemble selected portions of our code to see how we calculate some particular parameter (if he could find the right section of code). Being hardware dependent makes it unprofitable even to try to use our source code, though it might be easier to pick up some of our "trade secrets" from our code. So Tilmann, I have grown used to thinking about hardware dependent code as desirable. I very much understand that most programmers try to write portable code, and therefore think differently.

The file transfer from 6809 to PC project I described in my letter was a one of a kind project done for a consulting client so all that mattered was that it would run on my development system and his. As a matter of fact it failed that test on the first try. It turned out that my code wouldn't work if the computer had SMARTDRV running. The computer was caught tied up with it's read or write cache buffer and it missed information coming from the 6809 system. I discovered the problem accidentally one night and resolved it quickly (by disabling SMARTDRV for the data transfer).

Tilmann also was critical of my having written my screen editor to talk directly to the Video Ram. Not very portable, he said. I wrote the editor when I had a PC-XT which ran at 1/40 of the speed of my present 386 machine. Any other scheme of getting characters on the screen was intolerably slow at that time. My 2 MHz 6809 with a 19.2Kbaud terminal was much faster than the PC-XT using BIOS calls. Presently I could probably use BIOS calls, but my editor is done. I don't use serial terminals on PCs, so I haven't had any compatibility problems. I'm not running (nor am I planning to run) a BBS, so my editor doesn't have to run over a remote serial terminal link either. Aside from those considerations it has run on the old 8086 PC-XT, several 286 clones, a 286 and a 386 system with EGA monitor, my present 386 system at home and several 286, 386, and 486 systems at work with VGA and SVGA graphics boards. I have to ask how much more portable it could be. Lastly the direct video RAM access is a full twelve times faster than using BIOS calls, by actual measurement.

To put it quite directly, I see little reason to degrade the performance of my own tool that I am not planning to offer for sale! (I've already offered to give it away. That way I don't have to support it <:-)).

What DID give me fits in porting my program from a 68000 in C to the PC, was my unfamiliarity with the multiple memory models. I use a 200K edit buffer, and I discovered that though C's malloc() memory allocation function didn't complain, (It gave me a valid pointer to a buffer), it only gave me access to 64K of buffer. After a week of hair pulling I used the "if all else fails read the manual" approach and found a function called farmalloc() which successfully allocated a 200K buffer.

Who in his right mind would prefer an X86 with it's segment registers to a 68000 that can seamlessly address 8 megabytes of memory without much more than a fleeting thought about it? (Apparently IBM would, raising doubts in my mind as to their sanity). I know, a 386 or 486 running under OS/2 can address memory linearly as does the 68000. The limitation used to be in the 8086 and the 286. The 386 and 486 can address large memory directly but are limited by MS-DOS or PC-DOS.

6809 Computers

Last Friday, I was trying to get ahead on some projects at work since the end of our fiscal year is coming and we want to look good for our annual report. I made some interesting discoveries while trying to debug a program.

When you don't initialize a Motorola 6821 (a dual port parallel interface device) both ports are set up as inputs by the power-on reset. Apparently when you then write to the data direction register, thinking you are writing to the data register, the impedance at the associated input pin changes. Strangely when you write a 1 to that bit of the DDR, the voltage level at the open or lightly loaded input terminal actually goes down. My 74LS640 buffers were light loads and could read the state of the DDR through the associated I/O pin, though what came out was the complement of what I wrote in.

The reason the supposed output port was not initialized was that I had called the wrong port initialization routine in my program. I had very carefully checked the correct routine to be sure it was set up to initialize the port correctly, but then I called a previously used routine that had been carried over into my program by accident. I had put pieces of a number of old programs together to make a new one for some different hardware, and had missed at least that problem.

Let me tell you how a not too bright and pretty tired engineer / programmer can trap himself. I wanted to output a unique pattern of 8 bits since I had seven of them connected, three to some LEDs and four to solid state I/O modules that had indicator LEDS, so I could see their states. I decided something like hex 55 (01010101) or hex AA (10101010) were too symmetrical so I settled on hex 96 (10010110). I ought to be able to tell if the bits were in the right order, oughtn't I? Well, the technician who wired this rather different one of a kind computer had reversed the data bits end for end so they were (01101001), and then because the port wasn't initialized correctly, they got inverted by the input I thought was an output, i.e. 1's and 0's interchanged, so the two errors cancelled out and I got 10010110, just what I had expected would prove everything was working correctly. Some late night sessions, you just can't win! In this case two wrongs did make a right or a double negative cancelled itself out or something like that.

PC's Again

As I write this paragraph it is Sunday January 30. Last Thursday we had an ice storm in Ann Arbor. The main streets were nice and clear but when I got to the side street that feeds the court on which I live, I noted glare ice. I parked at the head of the court and walked the rest of the way to find a man in a car stuck in front of my mailbox. He had tried to turn into the court to turn around and had ended up about 50 yards down the street, about half way from the head to the dead end circle. All of his attempts to go up toward the connecting street resulted in his going down toward the end of the court. Unfortunately our court is sloped from the connecting street all the way down to the end.

I decided to work at home on Friday. Since I had some software drivers I had written for a stepper motor driver card I thought I would test them. I had brought the card home on Thursday and started working on the debug of the code I had written previously when I had the instruction manual which preceded the card by a couple of weeks. By Friday night at 11:00 I still had not gotten any stepper pulses out of the three axis driver board.

On Saturday I decided that I had narrowed the problem down to the settings in just one of the many control registers on the board, so I started a little "trial and error" programming and on about the 4th try I had pulses coming out of the board, but they didn't shut off after the programmed number. A little more experimenting and suddenly I had it all working. I have a list of six or seven errors in the instruction manual. To be fair to the supplier, a disk came along with the board with some drivers written in BASIC and some in C. They supplied a header file and the executable code module. Since we really want to know "all about" our hardware, I had decided to do the driver myself. I've written a letter to the supplier asking for a corrected manual. They conveniently supplied a bug report sheet at the end of the manual but I had three or four times as much as I could fit on their form.

We've used several boards from Industrial Computer Source with excellent results previously. We had used their multiple parallel port boards in several flavors. The manual for those had contained errors also, but fortunately they included the data sheet for the parallel port chip so we had no trouble using it. In the present case the manual contradicted itself, indicating the use of a bit in one of the registers that had been indicated unused in another portion of the manual. It seemed to me as though they had expected all of their customers to used their canned drivers, so, though they wrote detailed programming information, they didn't go through the steps they outlined to see if they would work.

I suspect this was a new manual for them, a new product line. I'll be interested in the reply that I receive from them. If it is interesting enough, I'll pass it along here next time.

Last time I ran across something like this we had bought a computer controlled "smart servo controller" specified by OUR customer for a project. We hadn't used this product before, but had always used our own servo controller cards and built the smarts into our 6809 based computer. The controller would work just fine for a long time and then suddenly it would go crazy as if it had been recording all the motions for an hour, and then decided to play them back. The supplier assured us that we must be doing something wrong.

At long last I wrote a letter to the supplier complaining that the software wasn't robust enough. It was up to the user (us) to insure that we didn't send it a command to move 0 distance. Of course if I had a terminal in front of me and was sending it commands, I could remember not to do that, but we had to build a test into our program that controlled the servo, to trap zero distance moves.

A while later after several long conversations with engineers and programmers at the servo package supplier, in which I included threats to replace their package with our own that worked or at least gave us full access to the software, I received a call one Friday afternoon in which they told me that they had reproduced the problem in their lab, and that the chief engineer would be at our door on Monday morning with new EPROM program chips to plug into the several servo controllers that we were using. By then our customer had agreed to let us use our own servo controllers but we agreed to giving the supplier another chance.

The new program worked fine. I quizzed the chief engineer and he said that we were using the program in an unusual way. Most customers would program a motion profile into the controller and just run it over and over again. We had to tell the controller different information each time. Due to some small oversight, the stack could overflow and it then "started over again" quite literally replaying the previous moves that it had made.

The chief engineer said that he had passed my letter around to all of the programmers and made them read it! Perhaps this is why we are adverse to using someone else's software. We'd rather make the mistakes ourselves and then have full access to the code so we can debug it when a problem occurs. The bugs in the purchased servo controllers had cost our project several weeks, and for a time had lost us the confidence of our customer.

Beginner's Page

Last time I included a section just for beginners in computing. I plan to do this regularly. I thought we might discuss the 6800 and the 6809 a bit today. In order for any discussion of Assembler programming to make any sense whatever, we need to first look at the "architecture" of the processor chips. Both of these processors access an 8 bit data bus, and both can access 65,536 bytes of memory via a 16 bit address bus. I'll mention a few assembler instructions in the discussion that follows. If you understand them you'll be a little bit ahead. If not, don't sweat. We'll get into a lot more detail in a later column.

If you are familiar with the intel "80" processors you might be surprised to find that the Motorola processors don't have a separate I/O bus with I/O addressing. Peripheral devices on the 680X processors simply use memory address space. SouthWest Technical Products Co. (SWTPc) used the 4K block of memory from $E000 to $EFFF for I/O. They used F000 to FFFF for ROM monitor space. Of course I/O would never need 4K nor would the ROM be that big, so some address space was wasted. At the time these systems came along, however, 56K of user memory seemed to be many times what anyone would use(!). How times have changed!

Let's list the features of each of the chips:

6800		6809	
Accumulator A		Accumulator A	
Accumulator B		Accumulator B	
		Accumulator D *	
Index Register X		Index Register X	
		Index Register Y	
Stack Pointer S		Stack Pointer S (system stack ptr)	
		Stack Pointer U (user stack ptr)	
Program Counter PC		Program Counter PC	
Condition Code CCR		Condition code CCR	
		Direct Page DP	
Interrupt Facilities			
Interrupt	IRQ	Interrupt	IRQ
		Fast Interrupt	FIRQ
Non Mask. Int.	NMI	Non Mask. Int.	NMI

* Accumulator D of the 6809 is the concatenation (joining) of Accumulator A and B, with B as the low order byte. See discussion below.

If you think the 6809 looks like an expanded 6800, you have it pretty well figured out. Now let's run through the functions of each of these registers. Each item mentioned above is an 8 bit or a 16 bit register that is internal to the processor. You could view them as special memory locations addressable by name. They have much more specific functions than general memory locations.

Accumulators - These are where the arithmetic manipulations take place. Values in accumulators can be loaded from memory or stored to memory. They can be added or subtracted. The 6809 has a MUL instruction (multiply A by B, result in D). The 6809 has the useful extension of being able to place the A and B accumulators together to form a 16 bit register. Motorola implemented a few instructions that take advantage of this feature. You can load D (LDD #$FFE7) with a specified value in the program or from a memory location (LDD $1234). You can store D in memory or transfer the contents of D to one of the other registers such as X, U, or Y. The instruction set falls short of much more than that, however. For example you can complement or negate the contents of A or B, but there is no NEGD or COMD instruction. You can shift a value in A or B but not in D. There are other differences, but we will get into those when we talk about instruction sets later. Just remember for now that the Accumulators are used to do arithmetic. Using a second accumulator (B is not quite as talented as A with regard to the instructions it can do) avoids a lot of temporary storage of values. It can make a program considerably more efficient.

Index Registers - An index register is an "indirect address" feature. You load X with the address of a variable and you can then access it as in LDA 0,X. The 6809 is more useful in that you can add an offset to the address in X. The 6800 allows you to increment or decrement the address in X with the INX and DEX instructions. The 6809 allows you to add a value to the address in X. LEAX 27,X will add 27 to the value in X. The 6800 allows you to add a value in B to X with the ABX instruction. You can write programs that are much more efficient with the 6809 instruction set. The 6809 in addition has the second index register Y. Having two index registers is a great advantage, again as we will see later when we discuss instruction sets. The X and Y index registers are both 16 bit registers so they can hold full addresses.

Stack Pointers - The S register in both processors is used as the subroutine return address stack pointer. When you do a JSR or BSR instruction (Jump or Branch to a subroutine), the address of the next instruction in the program (where the program is to resume running after completing the subroutine) is "pushed" onto the stack. At the end of the subroutine when the ReTurn from Subroutine instruction is found, that address is used to resume execution of the program. That is, the return address is pulled off of the stack and placed in the program counter (PC). The stack itself exists in memory. The utility of the stack pointer is that it is faster to access memory through it than directly.

The User stack as the name implies can be used by the programmer for whatever he wants. Some assembler programmers use it to access their variables as set up in memory. We'll talk about that technique later. A stack pointer simply keeps track of items on the stack. An assembler program can initialize the stack pointer. Pushing a byte on the stack decrements the stack pointer and places the byte at that address. Pulling a byte off the stack simply reads the byte and then increments the stack pointer.

The condition code register reflects the result of an operation. If two values were compared and were equal, the Zero flag is set. If a value was subtracted from another and the result was negative, the Negative flag is set, etc. The status register can be used directly by a program, but more commonly it is used by the processor to test for a branch instruction. For example, BEQ, (Branch if Equal) will happen if the zero flag is set. BNE (Branch if Not Equal) will happen if the zero flag is NOT set. I must insert here that the Motorola assemblers and instruction descriptions use the dollar sign ($) as a hexadecimal indicator. $10 indicates hexadecimal 10 (decimal 16).

The PC register is not changed directly by a program instruction, but it is frequently changed by the execution of an instruction such as JMP (Jump) or a branch instruction. JMP $1234 causes the program counter to be loaded with the memory address $1234, and the next program instruction is taken from that location. When you write an assembler program the last item at the end is the starting execution address (commonly called the transfer address) so that when the program is loaded into memory, the processor knows where the starting point is.

Interrupts are enough of a subject to fill a whole column. Let me just say that an interrupt usually comes from an external device. You are working along editing a file and simultaneously printing another file to your printer. The printer runs out of text in it's buffer and says INTERRUPT -- Hey! give me more data. Your edit session is interrupted to send another character (or a line) to the printer and you can resume editing. If the processor is fast enough, you don't notice the time the processor was away and the computer looks as though it were doing two things at once. In my days of using the 680X I never found a suitable "print spooler" that would let me print a file and not have my editor go dead long enough to miss a couple of characters if I happened to be typing along at high speed. Interrupts are extremely handy when you are trying to measure the time between two external events that can signal when they occur.

The 6800 has a special short addressing mode for the first 256 bytes of memory. That is, memory from $0000 to $00FF. Addresses in this range can be represented by a single byte. Special instructions allow using single byte address values for these locations, thus making a program smaller. The 6809 allows you to treat any 256 byte "page" of memory as the direct addressing page. The value in the DP register is the high order byte of the address of the direct page. Thus if you set DP=$02, addresses $0200 to $02FF are the direct page. Further, in an assembler program you can set the DP register as often as you wish, and treat different areas of memory as addressable with a short address.

Since programs have grown so much, most of them use much more than 256 bytes for variables. Most higher level language compilers simply ignore this feature of the 6809 and use "extended addressing" mode for all memory addresses. It was a neat trick when a large memory was 16K. SWTPc sold their first computers with a 4K memory board but it could be bought with enough memory chips for 2K. You had to buy a 2K upgrade kit to make a 4K out of it.

While the 6809 may appear to be a minor step forward from the 6800, the greater hardware facilities (registers) and the more powerful instruction set can make quite a difference in code in assembler to do the same function. I can't resist one example so here is code to move $100 (decimal 256) bytes of data from locations starting at $3000 in memory to locations starting at $1000 in memory. I'll comment the code. We'll get into much more detail regarding the instruction sets next time.

```
        ldx #$1000      use x register as a pointer for the move
        stx xdest       save destination address in temp location
        ldx #$3000      source address
loop    lda 0,x         get a byte from source location
        inx             increment the pointer
        stx xsrce       store it temporarily
        ldx xdest       get dest in x
        sta 0,x         store the byte at dest
        inx             increment dest pointer
        stx xdest       store it
        ldx xsrce       load source ptr again
        cmpx #$3100     have we done $100 bytes yet?
        ble loop        if not, go around the loop again
```

When we break out of this loop we have moved 100 bytes of data from memory location $3000 to $3100 to locations $1000 to $1100. Let's see how the 6809 could do the same thing:

```
        leax #$1000     slightly different syntax dest address
        leay #$3000     use Y for source
loop    lda ,y+         get a byte and increment y
        sta ,x+         store a byte and increment x
        cpx #$1100      have we done $100 bytes yet?
        ble loop        if not go around loop again
```

To be sure, there are other ways of coding either of these, but it ought to be obvious that the 6809 has a more "powerful" instruction set and better hardware facilities. Having two index registers removes the need to swap the value in the X register twice per pass through the loop and having the post increment instruction (,x+) eliminates the INX instruction. Note particularly that there are many less instructions included in the loop.

With the 6809 it wouldn't be hard to speed up the operation by using the D register and moving two bytes at a time:

```
        leax #$1000     slightly different syntax dest address
        leay #$3000     use Y for source
loop    ldd ,y          get two bytes and increment y
        leay 2,y        add 2 to value in y
        std ,x          store two bytes and increment x
        leax 2,x        increment x by 2
        cpx #$1100      have we done $100 bytes yet?
        ble loop        if not go around loop again
```

This works fine for an even number of bytes, but the autoincrement can't be used to add 2 to the index register value. Instead, we have to use the more general increment X by a constant instruction. leay can increment the value by up to 127. The loop got longer by two instructions but we only have to execute it half as many times since we are moving two bytes at a time.

Ron Anderson
3540 Sturbridge Ct.
Ann Arbor, MI 48105

Do you need
***Micro Cornucopia* Disks?**
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
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
Spellbinder v5.3H	$60.00

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

TCJ Center Fold

Special Feature

All Users

SS-50/SS-30 BUS

The centerfold is the SS-50 and SS-30 BUS. This bus is an 6800 based bus system. The documentation is based mostly on the GIMIX GHOST MOTHER BOARD. It provided 15 slots for full size (SS-50) boards and 8 slots for I/O-sized (SS-30) boards. A special 10 pin slot also provides Baud Rate Generation. Some of the features are:

1) Fully compatible with the SS-50 (6800) and SS-50C (6809) busses.
2) Gold plated PIN and Socket type connections.
3) 4, 8, or 16 decoded addresses per I/O slot.
4) Extended address decoding for the I/O section.
5) The I/O block is DIP-Switch addressable to any 32, 64, or 128 byte boundary.
6) Baud rate generator for 75 to 38,400 baud.
7) All data, address, and control lines are terminated and separated by noise reducing ground lines.

The following information on the SS-50 Bus is gathered from an 1982 Sam's book, #21810, "The S-100 & Other Micro Buses", by Elmer C. Poe and James C. Goodwin (ISBN: 0-672-21810-0). This book covered all micro based buses in use at the time. A couple of chapters are devoted to interfacing the S-100 to Benton Harbor Bus, TRS80 Model I bus, and 6502/6800 system (KIM).

The SS-50 bus was introduced in 1975 by Southwest Technical Products Corporation (SwTPC) for their SwTPC 6800 microcomputer system. Since then, use of the SS-50 bus grew and at least a dozen manufacturers produced components that ranged from disk controllers to digital video boards. All manufacturers of the SS-50 conform to the original bus definitions laid down by SwTPC. The compatibility problems that plague the S-100 bus do not exist for the SS-50 bus.

Since it was based on the SwTPC 6800 microcomputer system, the SS-50 bus easily supports the 6800 and 6502 processors. Unlike the 8080, these processors do not multiplex control signals on the data bus. All control signals are derived on the processor. Parallel and serial I/O ports and memory are all treated alike by the CPU. Each is addressable and must respond to a simple R/W line and the address bus.

Most of the signal on the SS-50 bus come directly from the 6800 processor. The SS-50 data bus is bidirectional; it is not split into data-in and data-out lines.

The SwTPC motherboard accepts two different size boards - an SS-50 and a peripheral board called the SS-30. The SS-30 board uses a subset of the SS-50 signals.

The SS-50 boards are 9 by 5 1/2 inches (22.8 by 13.9 cm) in size. They plug into the motherboard through 50 pin Molex connectors. The pins are numbered from right to left. Positive indexing is provided by plugging pin 33 on all boards.

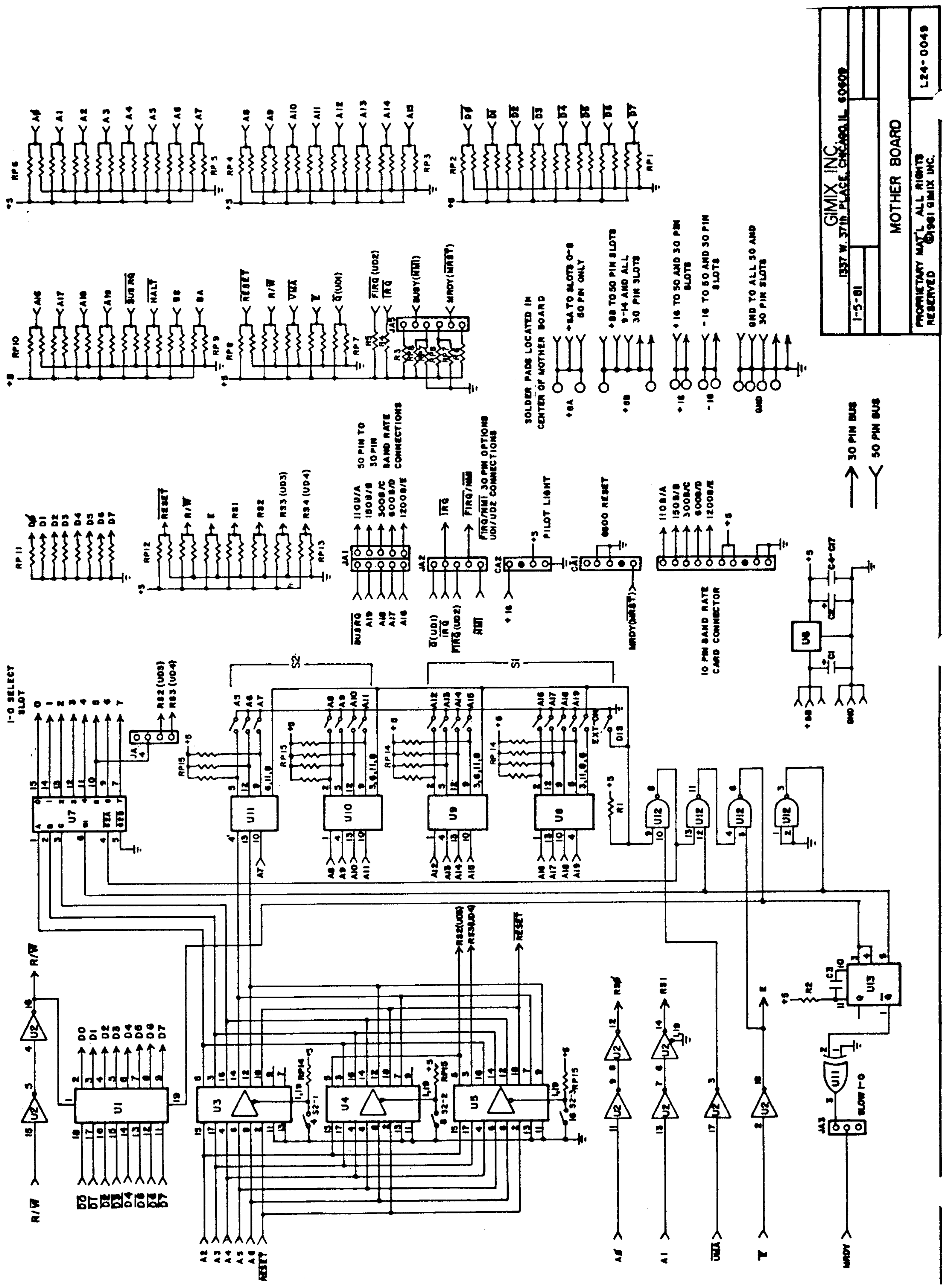
The SS-30 is designed to facilitate use of both serial and parallel interface devices. The most obvious difference between the two is the absence of the address bus on the SS-30. It is replaced by board select lines which are derived from the SS-50 address bus. Each board has been assigned a block of four contiguous addresses in memory. Address decoding circuitry on the motherboard generates a board select signal (using a 1-of-8 decoder) when the base address assigned a board is put on the address bus. The board select signal is fed only to the proper board, which must then respond to the register select lines, RS0 and RS1. The RS0 and RS1 lines are connected to all peripheral boards and represent address bus lines A0 and A1. Using the RS0 and RS1 lines, a board can determine which of its four locations is being addressed. One of the peripheral boards is usually dedicated to a serial terminal.

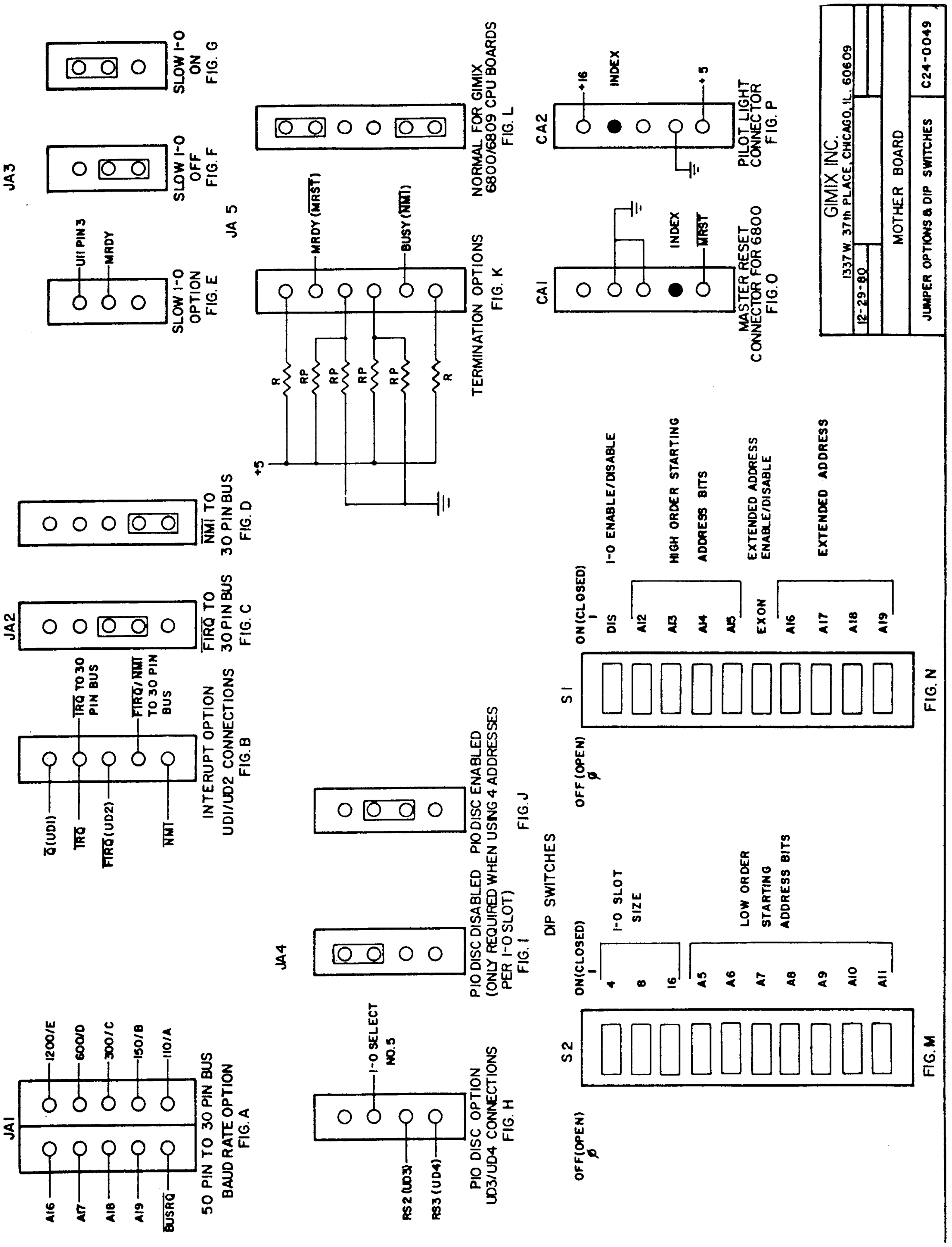
SS-30 boards are 5 by 4 inches (12.7 by 10.1 cm) and plug in through 30 pin Molex connectors. Positive indexing is provided by plugging pin 7.

Implementation of Motorola's 6809 microprocessor on the SS-50 required redefinition of several lines. Most of the changes resulted from differences in control lines between the 6800 and the 6809. This changed bus was know as the SS-50C. Some changes were also prompted in the SS-30, producing the SS-30C.

Board Select Address Definitions for SS-30 Select Line:

Select #0	$8000
Select #1	$8004
Select #2	$8008
Select #3	$800C
Select #4	$8010
Select #5	$8014
Select #6	$8018
Select #7	$801C


GIMIX INC.
1337 W. 37th PLACE, CHICAGO IL 60609
1-5-81
MOTHER BOARD
L24-0049
PROPRIETARY MAT'L ALL RIGHTS RESERVED ©1981 GIMIX INC.
SOLDER PADS LOCATED IN CENTER OF MOTHER BOARD
+8A TO SLOTS 0-8 50 PIN ONLY
+8B TO 50 PIN SLOTS 9-14 AND ALL 30 PIN SLOTS
+16 TO 50 AND 30 PIN SLOTS
-16 TO 50 AND 30 PIN SLOTS
GND TO ALL 50 AND 30 PIN SLOTS
30 PIN BUS
50 PIN BUS
50 PIN TO 30 PIN BAUD RATE CONNECTIONS
30 PIN OPTIONS UD1/UD2 CONNECTIONS
PILOT LIGHT
6800 RESET
10 PIN BAUD RATE CARD CONNECTOR
I-O SELECT SLOT
SLOW I-O
MOTHER BOARD


GIMIX INC.
1337 W. 37th PLACE, CHICAGO, IL. 60609
12-29-80
MOTHER BOARD
JUMPER OPTIONS & DIP SWITCHES
C24-0049
JA1
A16
A17
A18
A19
BUSRQ
1200/E
600/D
300/C
150/B
110/A
50 PIN TO 30 PIN BUS
BAUD RATE OPTION
FIG. A
Q (UD1)
IRQ
FIRQ (UD2)
NMI
IRQ TO 30 PIN BUS
FIRQ/NMI TO 30 PIN BUS
INTERUPT OPTION
UD1/UD2 CONNECTIONS
FIG. B
JA2
FIRQ TO 30 PIN BUS
FIG. C
NMI TO 30 PIN BUS
FIG. D
U11 PIN 3
MRDY
SLOW I-O OPTION
FIG. E
JA3
SLOW I-O OFF
FIG. F
SLOW I-O ON
FIG. G
RS2 (UD3)
RS3 (UD4)
I-O SELECT NO.5
PIO DISC OPTION
UD3/UD4 CONNECTIONS
FIG. H
JA4
PIO DISC DISABLED
(ONLY REQUIRED WHEN USING 4 ADDRESSES PER I-O SLOT)
FIG. I
PIO DISC ENABLED
FIG. J
+5
R
RP
RP
RP
RP
R
MRDY (MRST)
BUSY (NMI)
TERMINATION OPTIONS
FIG. K
JA 5
NORMAL FOR GIMIX 6800/6809 CPU BOARDS
FIG. L
CA1
INDEX
MRST
MASTER RESET CONNECTOR FOR 6800
FIG. O
CA2
+16
INDEX
+5
PILOT LIGHT CONNECTOR
FIG. P
DIP SWITCHES
OFF (OPEN) ø
ON (CLOSED) 1
S2
4
8
16
A5
A6
A7
A8
A9
A10
A11
I-O SLOT SIZE
LOW ORDER STARTING ADDRESS BITS
FIG. M
S1
DIS
A12
A13
A14
A15
EXON
A16
A17
A18
A19
I-O ENABLE/DISABLE
HIGH ORDER STARTING ADDRESS BITS
EXTENDED ADDRESS ENABLE/DISABLE
EXTENDED ADDRESS
FIG. N

SS-5Ø BUS DESIGNATIONS

SS-5Ø	GIMIX	SS-5ØC
DØ	DØ	DØ
D1	D1	D1
D2	D2	D2
D3	D3	D3
D4	D4	D4
D5	D5	D5
D6	D6	D6
D7	D7	D7
A15	A15	A15
A14	A14	A14
A13	A13	A13
A12	A12	A12
A11	A11	A11
A1Ø	A1Ø	A1Ø
A9	A9	A9
A8	A8	A8
A7	A7	A7
A6	A6	A6
A5	A5	A5
A4	A4	A4
A3	A3	A3
A2	A2	A2
A1	A1	A1
AØ	AØ	AØ
GND	GND	GND
GND	GND	GND
GND	GND	GND
+8V	+8V	+8V
+8V	+8V	+8V
+8V	+8V	+8V
-12	-16	-16
+12	+16	+16
INDEX	INDEX	INDEX
MRST	MRDY	MRDY
NMI	NMI/BUSY	BUSY
IRQ	IRQ	IRQ
UD2	FIRQ	FIRQ
UD1	Q	Q
Ø2	E	E
VMA	VMA	VMA
R/W	R/W	R/W
RESET	RESET	RESET
BA	BA	BA
Ø1	BS	BS
HALT	HALT	HALT
11Øb	BUSRQ	BUSRQ or 11Øb
15Øb	S3/A19	96ØØb or S3
3ØØb	S2/A18	3ØØb or S2
6ØØb	S1/A17	48ØØb or S1
12ØØb	SØ/A16	12ØØb or SØ

SS-3Ø	GIMIX	SS-3ØC
UD3	RS2	RS2
UD4	RS3	RS3
-12	-16	-16
+12	+16	+16
GND	GND	GND
GND	GND	GND
INDEX	INDEX	INDEX
NMI	FIRQ/NMI	FIRQ
IRQ	IRQ	IRQ
RSØ	RSØ	RSØ
RS1	RS1	RS1
DØ	DØ	DØ
D1	D1	D1
D2	D2	D2
D3	D3	D3
D4	D4	D4
D5	D5	D5
D6	D6	D6
D7	D7	D7
Ø2	E	E
R/W	R/W	R/W
+8V	+8V	+8V
+8V	+8V	+8V
12ØØb	12ØØb/E	12ØØb
6ØØb	6ØØb/D	48ØØb
3ØØb	3ØØb/C	3ØØb
15Øb	15Øb/B	96ØØb
11Øb	11Øb/A	11Øb
RESET	RESET	RESET
I/O SEL	CS	I/O SEL

NOTE: THIS CHART DOES NOT INDICATE THE POLARITY OF THE SIGNALS. IT IS ONLY A COMPARISON OF THEIR NAMES.

THE NAMES IN THE "GIMIX" COLUMN REFLECT THE DESIGNATIONS THAT APPEAR ON THE MOTHER BOARD ITSELF AND IN THE DOCUMENTATION. THE ACTUAL SIGNALS AT SOME OF THE PINS DEPENDS ON THE JUMPER CONFIGURATION OF THE BOARD AND THE PARTICULAR CPU CARD INSTALLED.

Serial Interrupts for Kaypro II

by Walter J. Rottenkolber

Special Feature

Intermediate Users

Forth Interrupt Handler

Using Forth For Serial Port Interrupts In The Kaypro II

I suppose you have heard of interrupts, but what do you really know about them, how they work, and how to program them? Are they really as mysterious and difficult as you may have heard? This article describes Forth code I wrote to set up an interrupt driven serial port (the modem port) in a Kaypro II.

Interrupts can be tricky beasts, however. You turn loose a chip with a mind of its own and hope it behaves itself. Although this code works on my Kaypro II, similar code crashed a Kaypro 10 with the Advent ROM. As a result, I make no claims or warranties regarding the information or code in this article, or their suitability for your system. I will not assume any responsibility or liability for any damage or loss to hardware, disk drives or data. If you implement the code or information in this article you accept full responsibility for their use.

Gathering data from serial ports takes two forms: Polling and Interrupts.

In Polling, the program repeatedly checks the SIO port for data. Though Polling wastes processor time, it is simple to implement, and to synchronize with the program.

With Interrupts, the hardware generates a line signal to which the processor responds. Interrupts allow for immediate response to external activity, or efficient handling of infrequent and unpredictable signals.

Interrupts can deal with the problem of data input occurring faster than the program or operating system can process. This causes a problem familiar to owners of some later CP/M Kaypros, namely, characters dropped at the beginning of new lines when using modems 1200 baud or above.

Zilog's Z80 CPU has four hardware interrupts: Reset, Bus-Request, Non-maskable, and Maskable.

Reset puts the CPU and hardware into a startup mode. We usually think of it as the 'red button', but it is also activated on power up or by too low a line voltage.

Bus-Request is used when peripheral hardware needs to take over the address and data busses, usually to do direct memory access (DMA).

Non-maskable interrupts (NMI) causes the Z80 CPU to stop its activity and execute a CALL to memory location 102 (66H).

Maskable interrupts (MI) causes the Z80 CPU to stop its activity and execute other code. It differs from the other interrupts in that the software can turn the Z80 CPU response on or off by issuing either an Enable Interrupt (EI) or Disable Interrupt (DI) opcode. The Z80 has three maskable interrupt modes.

Mode 0 is the default mode enabled on power up or reset. On interrupt, the external hardware must place a single byte on the data lines. The Z80 CPU reads the data lines and interprets the data as a one byte opcode. Usually, this is an RST, which is a CALL to one of eight preassigned memory locations in page zero. But it could just as well be an INR which would increment a Z80 register to count the external event.

Mode 1 executes a CALL to memory address 56 (38H), to run the code there.

Mode 2 is the most powerful and flexible. It requires the peripheral chips to be Zilog's SIO or PIO, as they need to interact with the Z80 CPU. The Z80 CPU is programmed with the most significant byte (MSB) of a 16-bit base address, and the I/O chip gets the least significant byte (LSB). The peripheral chip can modify the LSB to address up to eight routines in an interrupt table. The interrupt recombines the data within the Z80 CPU and peripheral chip into a 16-bit interrupt address, to which a CALL is then executed. This address can be to nearly any location in the 64K RAM.

Non-maskable and maskable interrupts generally run code routines independent of the main program. However, the Z80 has a few block move opcodes, such as OTDR (which transfers a block of data to a port), that use interrupts to time a loop. The Kaypro uses the NMI just this way for disk read/write.

During activity involving the keyboard or video display, the Kaypro bank switches the lower 16K bytes of main memory to ROM and video memory. Any interrupts attempting to access code located here during a bank switch will crash. To prevent this, all interrupt code must be located at or above memory location 16,384. As a result, we must use Mode 2 to implement the interrupt program. It is the only mode that can address memory above page zero (memory 0 to 255).

This problem is not unique to the Kaypro, although other computer systems may use different blocks of memory. It's important to locate interrupt code in stable memory.

I use the Laxen and Perry F83-Forth for the Kaypro, modified with Van Duinen's PDE super editor. This Forth is based on Intel 8080 code and its assembler is limited to 8080 opcodes.

With that in mind, let's walk through the Forth source code. As is obvious, the code required to implement polling (Scr 2 & 15), is much simpler than for interrupts (Scr 2-12).

First come the constants for the assorted ports and control codes for the Kaypro II serial chip, Zilog's MK-3884 (ZSIO). Note that SIO channel-B will be needed even though I plan to set up interrupts only on channel-A. I excluded the codes for establishing the serial port parameters (baud, word-size, etc.), but have listed some code for simple settings (Scr 17-18) if you haven't got your own.

Next are definitions for the Z80 opcodes (RETI, 2IM, & AI-MOV) required to setup Mode 2 interrupts (Scr 3). The names chosen follow the Extended Intel notation rather than Zilog's. RETI is a special RETurn the ZSIO also interprets to reset the internal priority latch. 2IM sets Interrupt Mode 2. AI-MOV moves the MSB of the interrupt address from Reg-A into the interrupt Reg-I in the Z80. This is a special move opcode that only works with Reg-A. Unlike the 8080 opcodes, these are two bytes long, as are most of the new Z80 superset of opcodes. To be consistent, I decided to extend the 8080 assembler defining words with 6MI. This Word can create and compile the two byte opcodes. They are used the same as the 8080 codes by just naming them.

The ZSIO has two channels, A and B, which can be programmed independently. In the Kaypro II, channel-A handles the modem port, and channel-B, the keyboard. Each channel has two port addresses, data and control. These ports do double duty. You transmit data by writing to the data port, and receive data by reading from it. Likewise, you set control codes by writing to the control port, and get status information by reading from it.

Because one byte would not allow for enough data to control the ZSIO, the control port has eight write and three read registers internally. Registers are numbered from zero, and they default to zero after completing a function. Therefore, only registers above zero require setting before access. You do this by writing the register number to the control port. Register #2, both read and write, is missing from channel-A. This register, accessed only in channel-B, handles the LSB of the interrupt vector for both channels.

The SET and GET Words do writes and reads to the ZSIO's control port. Both require the register number even for zero register access. Compare this to the polling primitives (Scr 15), which only access register zero.

Describing all these registers would be an article in itself. Fortunately, there is some method in the madness. Write registers #3, #4, and #5 primarily set up the port parameters (baudrate, parity, etc>, as well as some synchronous mode parameters. Write registers #6 and #7 set up sync characters for the Monosych mode. So I'll limit the description to those registers involved with programming interrupts, namely, write and read registers #0, #1, and #2.

Look over the description of the registers in the side bar. At first, programming the ZSIO appears daunting because of all the options. The trick is to mark out the functions you don't need, so you can focus on the rest.

I wanted this code to be useful as well as instructive, so I implemented all four Mode 2 interrupts for Ch-A: Receive and Transmit data (with buffers), Receive error, and Ext/status.

The buffers are circular queues with a defined Base and Top. An Inpointer and an Outpointer locate the ends of the data in the queue. A Count variable tracks the amount of data in the queue and lets you detect whether it is empty or full.

The assembly code for the interrupt routines (Scr 6 to 8) use LABEL and not CODE. Label returns just the starting address of the assembly code. The Calls generated by the interrupts must see only machine code.

CODE is used when assembly code must be accessed by Forth, and return to Forth through Next. You could get the address of the code in a CODE word by using Tick (') and >BODY, but this would be unnecessarily complicated.

The receive data routine, >IRXBUF, gets the data from the receive data register and moves it to the receive buffer. The first step is to save all the registers used by the interrupt, just as the last is to restore the registers via SIORET. This is necessary to prevent crashing the interrupted program.

Note that with Forth, you can still factor out code subroutines and fragments which can then be Called or Jumped to. Note too, that while SIORET has an Enable Interrupt (EI) opcode, >IRXBUF does not have a Disable Interrupt (DI). That's because the ZSIO itself does a DI as part of the interrupt process. Also, on returning from an interrupt, you must use RETI, as this opcode is recognized by the ZSIO and causes an internal priority interrupt latch to be reset.

The next step in >IRXBUF is to input the data and store it temporarily in Reg-C. Then the value of IRXSIZE is maneuvered into Reg-DE while that of IRXCNT is placed in Reg-HL.

The subroutine HL>DE? is Called. It subtracts the value in Reg-HL from that in Reg-DE. If the value in Reg-HL is larger than that in Reg-DE, a borrow occurs, and the Carry/Borrow Flag is Set. Note the use of SUB and SBB. The first ignores the borrow in doing the subtraction, but the second includes it. Since the count is incremented after the data is saved, a count larger than the buffer size means that the buffer is full.

The use of Forth macros in writing assembly code is a nice touch that makes the code more readable. Here, C0= tests for Carry Not Set, and if True, we do the subroutine (>IRXBUF). This moves the data into the buffer, and updates the pointers and count. Otherwise, the subroutine is bypassed and the data is dumped. This is my choice of action. You could handle a buffer overrun differently.

The subroutine (>IRXBUF) first increments and saves the count. It then moves the value of Top into Reg-DE, and the Inpointer to Reg- HL. The data is moved to the buffer address pointed to in Reg-HL. The Inpointer is incremented, and compared to Top. If it has gone over Top, the Carry is Set (CS), and the Inpointer is reset to the value in Base before being saved. In that way the Inpointer continues its circular movement in the queue.

With a few changes, the assembly code for the transmit buffer routine follows that for receive. The main routine, TXBUF>, moves data from the buffer to the SIO transmit data register. The first step, after the registers are saved, is to test if ITXCNT is zero, ie. the buffer is empty. Subroutine HL0? does this by OR'ing the H and L registers, and, if both are zero, to set the Zero Flag. If the flag is Not Zero (0<>), a data byte is sent by (ITXBUF>), otherwise ITXRESET is done.

Subroutine (ITXBUF>) decrements and saves the count. The value of Top is placed in Reg-DE, and the Outpointer into Reg-HL. After moving the data from memory to the Tx-register, the Outpointer is adjusted as in (IRXBUF>).

Subroutine ITXRESET sets the Data Transmit Int. Bit to ready the ZSIO for future transmissions. When you return from the interrupt without sending data, you must set this bit or the next data transmission will not cause an interrupt. Flag 'OIE? is also set. This flag does double duty by indicating Tx-buffer empty and transmit ready.

Factoring code into subroutines and fragments makes testing and debugging much, much easier.

The Receive error and Ext/status routines are similar. Both clean out the receive data register as it probably contains junk. The appropriate status byte is read and stored into a variable. A flag variable is also set. The proper interrupt latch is reset, and then return. These two routines, as written, are designed primarily for studying interrupt behavior, since the status byte of a new interrupt will simply overwrite the old one.

It's important to save the starting status values. Status bytes do not startup as zero, and the information you seek may lie in the difference between the old status value and the new. Also, hardware design may invert the expected status bit values.

The Interrupt Table is the heart of Mode 2. It consists of eight two byte addresses, four for each channel, 16 bytes in all. You can have up to 16 tables located in a memory page defined by the MSB in the Z80 CPU. This allows for up to 32 interrupt driven channels.

When an interrupt occurs, a Call is made to one of these eight memory locations, depending on the channel and interrupt type.

```
Base: TxData  Ext/Stat  RxData  RxError  \ Channel-B
      TxData  Ext/Stat  RxData  RxError  \ Channel-A
```

Just as you can have interrupt priorities among chips, there is also a priority within the chip, as follows:

```
Lowest: Ext/Stat  TxData  RxData  RxError  \ Channel-B
        Ext/Stat  TxData  RxData  RxError  \ Channel-A
```

In both, RxError for Ch-A has the highest location and priority.

Let's take these interrupts from the top down.

RxError -- Interrupt generated when received data causes a parity, receiver overrun, framing, or end-of-frame (SDLC) error. Parity errors may be excluded from the list. All except the framing error cause the relevant error bit in RR1 to be latched so that subsequent data input does not change it. This allows for a block of data to be received and then checked for an error. The latched bits must be reset to detect new errors. If the corrupt data is not removed from the receive register at this time, it will later cause a normal RxData interrupt and be saved.

RxData -- There are two receive data registers in the ZSIO and any data in them will trigger this interrupt. If these registers are full and a third character received, it is discarded and an overrun error will occur.

Ext/Status -- Interrupt triggered when a change occurs in the Data Carrier Detect (DCD), Clear to Send (CTS), or SYNC lines, or a Break received. A Break is a string of nulls long enough to cause a framing error. Though a type of receive error, it causes the interrupt here.

TxData -- Interrupt triggered when the transmit data register is emptied, and ready to accept another character. If there is no more data to be transmitted, the Transmit Enable Bit must be set. Otherwise, this interrupt will not occur with the next data transmission.

The base address of the table takes the form xxxx000y. The four 'x' bits can be any value, and will determine in which of 16 'paragraphs' the table will be located within the 256 byte page memory. Bit 'y' can be 0 or 1. Its choice should be consistent or you can inadvertently overlap two tables by one byte. It's best, however, to set 'y' to 0. Not only are memory calculations simpler, but it guarantees that none of the interrupt addresses will extend into the next memory page.

The routine -- HERE 16 + 15 NOT AND -- calculates the next paragraph base address. This is saved in INTBL, and then used to set the Dictionary Pointer (DP) so that the interrupt table is located correctly as inline code.

I wrote the NOINT function both as a place holder in the table and as insurance from an aberrant interrupt. Since Ch-B interrupts are not enabled, it could be left off there. But the ITXBUF> address must still begin at TxData, the ninth byte of the table.

Ext/Stat and TxData can be enabled independently, but RxError tracks RxData. If RxData interrupts are enabled, you must provide for the associated RxError interrupt. You can use NOINT to force a RETI, and just ignore the data and status bytes. However, the corrupt data still in the receive register will then cause a RxData interrupt and be saved to the buffer. Using IRXERR instead would dump the corrupt data, and help eliminate screen garbage.

The Receive Error Bits can be used for byte-wise error detection in data transmission. This was popular in the middle 1970's, the dawn of microcomputing, but became obsolete after X-Modem came on the scene.

Next to setup are the buffers. You could locate them in free memory, but for now it's simpler to place them inline with ALLOT. Since all of the buffer pointers and count are variables, they must be initialized, and INITBUF does this. A program can also use it to dump the buffer contents. CLRBUF fills the buffers with nulls. This isn't absolutely necessary, but it is good practice to set all variables and buffers to a known starting state. This can be very helpful when testing and debugging.

Programs will access the ZSIO interrupt system via IMKEY? and IMKEY for input, and IMEMIT for output. The names reflect my using the ZSIO mostly for modem connection.

First to define are DII and EII, to block interrupts. Code that accesses or changes variables and pointers also affected by interrupt routines must complete their task without interruption. Otherwise you get a conflict problem often encountered in multitasking and networking.

The Words in angle brackets are the primitives. These Words mirror, in high level Forth, the behavior of the assembly routines in dealing with buffer counts and addresses. Note the choice of unsigned comparison operators, eg. U<. It's easy to forget that counts and addresses are unsigned integers, and to use signed integer operators. That is, until the values go above 32,767, and strange bugs appear.

I dreamed up the Word <IMEMIT#?> to use in block data transmission. It checks for a minimum buffer count before the routine fills up the buffer again. Otherwise, the buffer is constantly topped off a byte at a time, and data transmission slows. Waiting until the buffer is empty results in a jerky transmission pattern.

<IMEMIT> is designed to send data direct to the data port if the buffer is empty and to return through the interrupt. If data is being transmitted, then new data is added to the buffer. I originally had the routine add all data to the buffer, and then, if it were not transmitting, to access (ITXBUF>) to send it out. This seemed unnecessarily complicated, but shows that alternate routines are workable.

We initialize the interrupt vectors with INITVEC. First the LSB of the INTBL address is fetched and stored in Write Register #2 of Ch-B. Then the MSB is fetched and stored by CPUVEC into the Z80 CPU. This is where we use the rest of the Z80 opcodes. After the MSB on the stack is POPped into Reg-DE, and MOVed into Reg-A, AI-MOV transfers it to internal register-I. 2IM then sets interrupts to Mode 2.

ITBL-ON enables the interrupt table by setting the status affect vector bit (Ch-B only). Words SIOA-IOFF and SIOA-ION turn the interrupts Off and On in the ZSIO. Don't confuse this with DI and EI which disable and enable the ZCPU's response to maskable interrupts. Turning interrupts Off doesn't shut the ZSIO down. It still can Send and Receive data by polling. Also, the interrupts can be selectively turned On. You may find it more practical, for example, to use interrupts for Receive and polling for Transmit.

INIT-SIOA-INT starts the machinery going by initializing and clearing the buffers, initializing Mode 2 vectors and enabling the vector table, turning the interrupts On, and then initializing the Status variables and flags.

Lastly, comes a redefined BYE for exiting the program. It turns off the Ch-A interrupts with RESIOA or SIOA-IOFF. If you neglect this step, the interrupts may be forgotten, but, trust me, they are not dead. An interrupt will seek out the now nonexistent vector table, jump routine, and program, and your system will crash. Just turning off the modem can do it.

KILLSIO and KICKSIO are a crude, first pass attempt to reconcile a ZSIO on interrupts and the Kaypro II disk system. I had an ASCII file transmit system that also captured the echoed text into a file. At the time only the data interrupts were implemented. When the program did disk I/O, it came back to a dead ZSIO even though the Kaypro disk routine disables the maskable interrupts. A variety of error flags were set so apparently enough ZSIO activity occurred to crash the chip. I found that turning the Receive and Transmit enable bit Off required resetting the entire ZSIO system to turn back On. So I chose to simply turn the interrupts Off with KILLSIO, and let the chip crash. Later, KICKSIO resets all the status latches, cleans up the system, and restarts transmitting any data in the buffer. There has to be a better way.

Part of the challenge integrating these routines into your program is dealing with the buffers. Polling usually has a solid connection to your program so that changes in I/O occur instantly. Buffers add a certain springiness to I/O response that your program must allow for and control. To send a Cancel command, for instance, you will need to dump the transmit buffer first. If the command were just placed at the end of the buffer, it could take several seconds before being sent. And a closing disk save must account for a partially filled receive buffer.

Be certain that at least the variables, assembly code, and interrupt table are located above the 16KB level in RAM to avoid being bank switched out. The simplest way is to test HERE before the variables are loaded, and set the dictionary pointer (DP) at or above 16,384. The statement -- HERE 16390 MAX DP! -- placed just before the variables should do it.

A rough timing estimate on my 5 MHz Kaypro shows that the assembly I/O routines can process about 12,700 bytes/sec., and the Forth I/O code, 5,200 bytes/sec. Since what goes in the buffer has to come out, the average data rate, about 3700 bytes/sec., is the actual throughput. In theory, that's fast enough to handle a 14,400 baud modem if the rest of the program (or system) doesn't add too much overhead.

Well, that's all there is to implementing a fancy interrupt driven serial I/O on the Kaypro II. You may find it confusing at first because there are so many parts to consider and program. But all is revealed once you piece them together and complete the puzzle. Best of all, the lessons learned here can be applied to other interrupt systems.

Description Of Zilog SIO Registers Used In Interrupts

Write register #0 (WR0):

Bits 0 to 2 -- When set to a number from 0 to 7 causes the next access, whether read or write to go to that register (if it exists). After the access, the register resets to #0.

Bits 3 to 5:
0 0 0 Null code -- no action is taken. Allows for other bit commands without affecting these bits.
0 0 1 Send Abort -- in SDLC mode only.
0 1 0 Reset Ext/Status Int. -- After an interrupt generated by a Break or a change in the DTR, CTS, or DCR, the status bits in RR0 are latched so they can be read. This command resets the latch so that interrupts can occur again.
0 1 1 Channel Reset -- Like the hardware reset but only effects channel-A.
1 0 0 Enable Int. on next Rx character -- Used with the On First Receive Character mode in which the int. must be reset after each Rx int.
1 0 1 Reset Tx Int. Pending -- If the Tx data register must be left empty (no more data), the Tx Int. must be reset.
1 1 0 Rx Error Reset -- After a Parity or Overrun error, bits in RR1 are latched so they can be read later. This command resets the latch.
1 1 1 Return from Interrupt -- This command is present only in channel-A. It does the same as the RETI opcode in resetting the highest priority latch so that lower priority devices in a daisy chain are enabled.

Bits 6 and 7 -- Involved with SDLC mode. Normally set to null in async. mode.

Write Register #1 (WR1):

Bit #0 -- Enables the Ext/Status Interrupt which causes interrupt on changes in the DCD, CTS or SYNC lines, or from a Break/Abort transmission.

Bit #1 -- Transmit Interrupt Enable

Bit #2 -- Status Affects Vector (Ch. B only). This enables the interrupt vector table. Otherwise only a single address would be generated for all interrupts.

Bits #3 and #4:
0 0 Receive Interrupt Disable -- Self explanatory.
0 1 Rx Int. on First Character -- Requires Rx int. to be reset after each character (see WR0)
1 0 Int. on all Rx characters (include parity in Rx error int.)
1 1 Int. on all Rx characters (exclude Parity from Rx error int.)

Bits #5 to #7 -- Wait/Ready Function Selector. Used to control the ZSIO's response to the WAIT and READY lines during block reads or writes with a direct memory access (DMA) controller.

Write Register #2 (Ch. B only) (WR2):

Holds the least significant byte of the interrupt vector. During an interrupt, all the bit values as written are returned if the Status Effects Vector control bit is 0. If this bit is 1, then only bits #1, and #5 to #7 are returned as written, and bits #2 to #4 are modified depending on the particular interrupt so as to access the proper address in the interrupt table.

Read Register #0 (RR0):

Bit #0 -- Rx char available.

Bit #1 -- Interrupt pending (Ch. A only)

Bit #2 -- Tx buffer empty.

Bit #3 -- DCD (Data Carrier Detect) Shows inverted state of DCD pin. If interrupt enabled for Ext/Stat, a change in any of the bits (DCD, /CTS, Sync/Hunt, Break/Abort, or Tx-underrun/EOM) will cause the DCD to be a latched until the Reset Ext/Stat command (WR0).

Bit #4 -- Sync/Hunt. Used in Sync. mode.

Bit #5 -- CTS (Clear to Send) Shows inverted state of CTS pin. Interrupts as in DCD.

Bit #6 -- Tx Underrun/End of Message -- Used in Sync. mode.

Bit #7 -- Break/Abort. In Async. mode, a null sequence in the input stream long enough to generate a null character plus framing error will set this bit and cause an interrupt if enabled (see DCD). The data register will contain a null byte which should also be discarded as part of reset routine. In Sync. mode, an abort sequence (7 or more 1's) in the input stream sets the bit and causes an interrupt.

Read Register #1 (RR1):

Bit #0 -- All Sent. In async. mode, bit is set when Tx data register is empty. Always set in sync. mode.

Bits #1 to #3 -- Used in SDLC receive mode.

Bit #4 -- Parity Error. Set when parity does not match programmed sense (even/odd). Error is latched and must be reset with Error Reset command (WR0).

Bit #5 -- Receive Overrun Error. Set when three or more bytes enter are received without a read from the CPU. Receive has two input registers to buffer input. In an overrun, these two bytes remain, and it is the third and additional bytes that are discarded. Error is latched and must be reset with Error Reset command (WR0).

Bit #6 -- CRC/Framing Error. In async mode, bit is set for a framing error. This error occurs if the Stop Bit is not detected when expected in the data word bit sequence. In sync mode, bit is set for CRC error. This bit is not latched but is constantly updated with new data. It is reset with Error Reset command (WR0).

Bit #7 -- End of Frame. Used in SDLC mode.

Read Register #2 (RR2) (Ch. B only):

Contains the interrupt vector written into WR2. If the Status Effects Vector bit is not set, the vector is returned as written. If the bit is set, the vector is modified. After an interrupt, it is set to the address of the highest priority interrupt pending at the time of the read. If no interrupt is pending, the vector returned is modified to xxxx011x (x= bits as written). This is equivalent to the Receive Error Condition for Ch. B.

```
\ Screen 0
\ Interrupt Driven Zilog Serial I/O     WJR04MAY94

            Interrupt Driven Zilog Serial I/O
            16-Bit Pointers
            Walter J. Rottenkolber
            9 Mar 94
            Forth-83 Source Code

( For Kaypro II CP/M computer using Laxen & Perry's F83.COM )

\ Screen 1
\ Interrupt Driven Zilog Serial I/O     WJR04MAY94
ONLY  FORTH ALSO  FORTH DEFINITIONS  DECIMAL
 2  3 THRU   \ Constants, ASM Words and Port routines.
19 21 THRU   \ ZSIO set parameter routines.
 4 13 THRU   \ ZSIO interrupt code.
  14 LOAD   \ Int. start/stop code.
  15 LOAD   \ Polling I/O.
  16 LOAD   \ Simple terminal routines.

\ Screen 2
\ ISIO -- Z80-SIO  Constants          WJR30MAR94
  0  CONSTANT  BAUDPORTA   \ Baudport for SIO Port-A
  6  CONSTANT  SIOACTL          \ SIO Control Port-A
  4  CONSTANT  SIOADATA    \ SIO Data Port-A
  7  CONSTANT  SIOBCTL          \ SIO Control Port-B
  5  CONSTANT  SIOBDATA    \ SIO Data Port-B
 40  CONSTANT  TXRES            \ Reset Tx Data Empty Int.
 48 CONSTANT  RXERRES        \ Reset Rx Data Error
 16 CONSTANT  EXTSTATRES \ Reset External Status Line Int.
 56 CONSTANT  INTRET         \ Does same as RETI opcode
 13 CONSTANT  CRR

\ Screen 3
\ ISIO -- Z80 Opcodes for Mode 2, SIO-A Ctl/Status    WJR19MAR94
HEX
: 6MI  ( b b) (S opcode)  CREATE SWAP C, C, DOES> @ , ;

 ED 4D 6MI RETI         \ RETI    = return from mode 2 int.
 ED 5E 6MI 2IM          \ IM 2    = set z80 to mode 2 int.
 ED 47 6MI AI-MOV   \ MOV I,A = set page addr. of int. table
DECIMAL
\ Words to Set SIO-A control bytes & to Get status bytes
: SETBAUDA  ( ctl)        BAUDPORTA PC! ;
: SETSIOA   ( ctl reg#)  SIOACTL PC!  SIOACTL PC! ;
: SETSIOB   ( ctl reg#)  SIOBCTL PC!  SIOBCTL PC! ;
: GETSIOA   ( reg# - b)  SIOACTL PC!  SIOACTL PC@ ;
: GETSIOB   ( reg# - b)  SIOBCTL PC!  SIOBCTL PC@ ;

\ Screen 4
\ ISIO -- IRx Buffer Variables          WJR04MAY94
VARIABLE  IRXSIZE
  128  IRXSIZE !  \ Change Buffer size as needed.
VARIABLE  IRXTOP
VARIABLE  IRXBASE
VARIABLE  IRXIPTR
VARIABLE  IRXOPTR
VARIABLE  IRXCNT
VARIABLE  IRXERRVAL
VARIABLE  'IRXERR?

\ Screen 5
\ ISIO -- ITx Buffer Variables          WJR04MAY94
VARIABLE  ITXSIZE
  128  ITXSIZE !  \ Change buffer size as needed.
VARIABLE  ITXMIN#
VARIABLE  ITXTOP
VARIABLE  ITXBASE
VARIABLE  ITXIPTR
VARIABLE  ITXOPTR
VARIABLE  ITXCNT
VARIABLE  EXT/STATVAL
VARIABLE  'EXT/STAT?
VARIABLE  'OIE?   'OIE? ON   \ true= Tx buffer empty

\ Screen 6
\ ISIO -- IRx & ITx Buffer I/O ASM Routines         WJR19MAR94
LABEL  HL>DE?  \ Carry flag set (CS) if HL>DE,  C0= if HL<=DE
  A ORA   E A MOV   L SUB  D A MOV   H SBB   RET
LABEL  HL0?   \ Zero flag set if HL= 0
  L A MOV   H ORA   RET
LABEL  SIORET  \ Common return path for >IRXBUF & ITXBUF>
  PSW POP   B POP   D POP   H POP   EI  RETI
LABEL  (>IRXBUF)  \ data in reg-C, IRXCNT in reg-HL
  H INX   IRXCNT SHLD   IRXTOP LHLD
  XCHG   IRXIPTR LHLD   C M MOV   H INX
  HL>DE?  CALL CS  IF  IRXBASE LHLD  THEN  IRXIPTR SHLD  RET
LABEL  >IRXBUF  \ Moves byte from SIO to rxbuffer
  H PUSH   D PUSH   B PUSH   PSW PUSH
  SIOADATA IN   A C MOV   IRXSIZE LHLD   XCHG  IRXCNT LHLD
  HL>DE?  CALL  C0= IF  (>IRXBUF)  CALL  THEN  SIORET  JMP

\ Screen 7
\ ISIO -- IRx & ITx Buffer I/O ASM Routines         WJR16MAR94
LABEL  (ITXBUF>)  \ itxcnt value in HL
  H DCX   ITXCNT SHLD   ITXTOP LHLD   XCHG
  ITXOPTR LHLD   M A MOV   SIOADATA OUT   H INX
  HL>DE? CALL CS IF  ITXBASE LHLD  THEN
  ITXOPTR SHLD RET
LABEL  ITXRESET
  TXRES A MVI  SIOACTL OUT
  TRUE H LXI   'OIE? SHLD  RET
LABEL  ITXBUF>    \ Sets var 'OIE? = true when buff empty
  H PUSH   D PUSH   B PUSH   PSW PUSH
  ITXCNT LHLD   HL0? CALL   0<>
  IF  (ITXBUF>) CALL  ELSE  ITXRESET CALL  THEN
  SIORET JMP

\ Screen 8
\ ISIO -- IRxErr & Ext/Stat ASM Routines          WJR30MAR94
LABEL  IRXERR  \ Err in parity, overrun, framing, & frame end
  H PUSH   D PUSH   B PUSH   PSW PUSH
  SIOADATA IN ( dump data)   1 A MVI   SIOACTL OUT
  SIOACTL IN   0 H MVI   A L MOV   IRXERRVAL  SHLD
  TRUE H LXI   'IRXERR? SHLD
  RXERRES A MVI   SIOACTL OUT   SIORET JMP
LABEL  EXT/STAT   \ Chg. in DCD, CTS, SYNCH, & Break
  H PUSH   D PUSH   B PUSH   PSW PUSH
  SIOADATA IN  ( dump data) "
  SIOACTL IN  0 H MVI   A L MOV   EXT/STATVAL SHLD
  TRUE H LXI   'EXT/STAT?  SHLD
  EXTSTATRES A MVI   SIOACTL OUT   SIORET JMP
LABEL  NOINT    EI  RETI   FORTH   \ Interrupt NOOP
```

```
\ Screen 9
\ ISIO -- Set-up Interrupt Table        WJR23MAR94
 VARIABLE  INTBL
 HERE 16 + 15 NOT AND INTBL !  \ Base of interrupt table
 INTBL @  DP !  \  Interrupt Table
           NOINT ,   NOINT ,   NOINT ,   NOINT ,   \ Channel B
           ITXBUF> ,  EXT/STAT , >IRXBUF ,  IRXERR ,  \ Channel A

\ Screen 10
\ ISIO -- IRx & ITx Buffer Initialize     WJR16MAR94
CREATE (IRXBUF)  IRXSIZE @ ALLOT
CREATE (ITXBUF)  ITXSIZE @ ALLOT
: INITRXBUF
  (IRXBUF) DUP IRXBASE !  DUP IRXIPTR !
  DUP IRXOPTR !  IRXSIZE @ + IRXTOP !  IRXCNT OFF ;
: INITTXBUF
  (ITXBUF) DUP ITXBASE !  DUP ITXIPTR !
  DUP ITXOPTR !  ITXSIZE @ + ITXTOP !  ITXCNT OFF
  'OIE? ON ;
: INITBUF   INITRXBUF  INITTXBUF ;
: CLRBUF
  IRXBASE @ IRXSIZE @ ERASE
  ITXBASE @ ITXSIZE @ ERASE ;

\ Screen 11
\ ISIO -- IRx & ITx Buffer I/O          WJR30MAR94
CODE  DII   DI  NEXT  C;
CODE  EII   EI  NEXT  C;
: <IMEMIT?>   ( - f)  \ f= true if itx-buf not full.
  DII ITXCNT @  ITXSIZE @ U< EII ;
: <IMEMIT#?>  ( - f)  \ f= true if itx-buf <= itxmin#.
  DII ITXCNT @  ITXMIN# @ U> NOT EII ;
: >ITXBUF   ( b)
  DII  ITXIPTR @ C!  ITXIPTR @  1+ DUP  ITXTOP @ > IF
  DROP  ITXBASE @  THEN  ITXIPTR !  1 ITXCNT +!  EII ;
: <IMEMIT>  ( b)
  DII  'OIE? @ IF 'OIE? OFF  SIOADATA PC!
  ELSE >ITXBUF THEN  EII ;

\ Screen 12
\ ISIO -- IRx & ITx I/O     WJR04MAY94
: <IMKEY?>  ( - f) \ f= true if char in rx-buf
  IRXCNT @ 0<> ;
: <IMKEY>   ( - b)  \ Equiv. to IRXBUF>
  DII  IRXOPTR @  C@ IRXOPTR @  1+ DUP  IRXTOP @ U> IF
  DROP  IRXBASE @  THEN  IRXOPTR !  -1 IRXCNT +!  EII ;
: IMKEY?  ( - f)  <IMKEY?> ;
: IMKEY   ( - b)  BEGIN PAUSE <IMKEY?>  UNTIL <IMKEY> ;
: IMEMIT  ( b)   BEGIN PAUSE <IMEMIT?> UNTIL <IMEMIT> ;

\ Screen 13
\ ISIO -- Init. Interrupt Vectors         WJR28MAR94
CODE  CPUVEC  ( hi-ivec-adr)
  D POP  E A MOV  AI-MOV  2IM  NEXT  C;
: SIOVEC  ( low-ivec-adr)  2 SETSIOB ;
: SIOA-VECON    4 1 SETSIOB ;
: INITVEC    \ Set up Interr. vectors for Channel-A
  INTBL DUP C@ ( sio ) SIOVEC  1+ C@ ( cpu ) CPUVEC ;
: SIOA-IOFF   0 1 ( ctl reg#)  SETSIOA ; \ iRx & iTx off
\ : SIOA-ION   24 1 ( ctl reg#)  SETSIOA ; \ iRx only
\ : SIOA-ION   26 1 ( ctl reg#)  SETSIOA ; \ iTx & iRx on.
: SIOA-ION  27 1 ( ctl reg#)  SETSIOA ; \ iTx, ext/stat, iRx on
: SET-ERRVAL    \ Store base status values.
  0 GETSIOA  EXT/STATVAL !   1 GETSIOA  IRXERRVAL !
  'EXT/STAT? OFF   'IRXERR? OFF ;
: INIT-SIOA-INT          \ Initialize Channel-A SIO interrupts
  INITBUF CLRBUF INITVEC SIOA-VECON SIOA-ION SET-ERRVAL ;

\ Screen 14
\ ISIO -- IRx & ITx  Stop and Restart WJR04MAY94
: KILLSIO   SIOA-IOFF  DII 50 MS ;
: <MKEY?>  ( - f)  SIOACTL   PC@  1 AND ;
: <MKEY>   ( - b)  SIOADATA  PC@ ;
: GOBBLE    BEGIN <MKEY?> WHILE <MKEY> DROP REPEAT ;
CODE (ITX>)  ( n)  \ n= itxcnt val
  H POP  (ITXBUF>) CALL  NEXT C;
: ITX>    ITXCNT @ (ITX>) ;
: KICKTX    ITXCNT @ 0= IF 'OIE? ON ELSE 'OIE? OFF ITX> THEN ;
: KICKSIO   \ Kickstart the SIO after KILLSIO
  DII  TXRES 0 SETSIOA            \ reset iTx
  EXTSTATRES 0 SETSIOA         \ reset ext/stat
  GOBBLE  RXERRES 0 SETSIOA                \ reset iRx
  0 GETSIOA 2 AND IF INTRET 0 SETSIOA THEN \ reset iRET
  SIOA-ION  KICKTX  EII ; \ turn on interrupts & restart iTx

\ Screen 15
\ Polling SIO -- KayPro2WJR16MAR94
: <MKEY?>  ( - f)  SIOACTL   PC@  1 AND ;
: <MEMIT?> ( - f)  SIOACTL   PC@  4 AND ;
: <MKEY>   ( - b)  SIOADATA  PC@ ;
: <MEMIT>  ( b)   SIOADATA  PC! ;
: MKEY?    ( - f)  <MKEY?> ;
: MKEY      ( - b)  BEGIN  PAUSE  <MKEY?>   UNTIL  <MKEY> ;
: MEMIT    ( b)   BEGIN  PAUSE  <MEMIT?>  UNTIL  <MEMIT> ;

\ Screen 16
```

TCJ CLASSIFIED

Needed: Amiga 1000 schematic (revision A). KB6ZBD, RR2, Box 195, Woonsocket, SD 57385.

Needed: CP/M Kaypro 10 Software, Excalibur "work in progress", "payroll", "order entry", "standard billings." Need these separate modules of the group of 11. No Longer supported. Have others in group intended for repair shop billing. Contact Jim's Repair Service, 8633 Wicker Ave., St. John IN 46373-9741, (219) 365-5555.

For Sale: 500 computer and technical books from 50's thru 90's. 300-400 in mint condition, many from 70's and covering micros and minis, software and hardware. Asking $1500. Richard Hawkins, (216) 371-5935.

KayPro printer (Juki 6100 daisywheel) and tractor feed, with cables. Extra printwheels. Works fine! Close out $45.

Bootable Z-System floppies and ZCOM for Kaypro 2x or 4. Yes you can run the Z-System from bootable floppies (on stock ROM machines). Here they are with installer disk to make bootable disks in three TPA's. Basic utilities, ZCOM (builds itself), ZRDOS and Z-system manual. All for $20. **"Getting the most from WordStar and Mailmerge: Things MicroPro Never Told You"** book by David Stone. For version 3.x and some Ver. 4 tips. Only $5.00. Call at (916) 371-2964.

For Sale: Make offer on SB180 Dual floppy system with COMM180 & ETS 180 boards. Loads of Software - Z-System utilities, programming tools, WordStar 4. All manuals, CP/M books, TCJ back Issues #22 to 66. Contact: Ian Partridge, 22 Laing Gardens, Broxburn, EH52 6XT, Scotland. Tel: +44 506 858038; Fax +44 331 7709.

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

```
\ Simple terminal        WJR04MAY94
: ESC?  ( b)  \ Escape from terminal loop
   27 ( ESC) = ABORT" ++DONE++" ;
: T   \ Simple polled terminal loop.
   BEGIN  KEY? IF KEY DUP ESC? MEMIT THEN
   MKEY? IF MKEY EMIT THEN  AGAIN ;
: IMDM    \ Initializes interrupt SIO
   1200BBS  INIT-SIOA-INT ;
: IT    \ Simple interrupt terminal loop.
   BEGIN  KEY? IF KEY DUP ESC? IMEMIT THEN
   IMKEY? IF IMKEY EMIT THEN  AGAIN ;
: .ERRVAL    \ displays int. error values.
   IRXERRVAL ?  EXT/STATVAL ? ;

\ Screen 17
\  Data to Set Zilog SIO Baud Rate  WJR18OCT93
  Baud   Ctl-Value
   300      5
  1200      7
  2400      0A
 4800       0C
  9600      0E
 19200      0F

\ Screen 18
\ Data to Set Zilog SIO Word Bits, Stop Bit, & Parity
 rx-reg# = 3 tx-reg# = 5
 tx-ctl        rx-ctl        bits
  88           1             5
  A8           81            6
  C8           41            7
  E8           C1            8
stop           ctl
  1            44            stop-bit and parity use reg# 4
 1.5           58            must be OR'd together.
  2            4C
 parity        ctl
 one           40
 odd           41
 even          43

\ Screen 19
\  Init. SIO Routine       WJR04MAY94
HEX
   VARIABLE  TXCTL  \ save tx ctl byte for hangup & mbreak
: RESIOA    18 0  SETSIOA ;  \ Reset SIO Channel-A
: SETBBS    \ Standard BBS setup - 8N1
   C1 ( rx) 3 SETSIOA  E8 ( tx 8-bits) DUP TXCTL ! 5 SETSIOA
   44 ( 1-stop )  40 ( no parity )  OR  4 SETSIOA ;
: SETPAK    \ Setup for some commercial systems - 7E1
   41 ( rx) 3 SETSIOA  C8 ( tx 7-bits) DUP TXCTL ! 5 SETSIOA
   44 ( 1-stop)  43 ( even parity) OR  4 SETSIOA ;
DECIMAL \S

\ Screen 20
\  Set SIO Parameters   WJR04MAY94
: 1200BAUD    RESIOA  7  SETBAUDA ;
:  300BAUD    RESIOA  5  SETBAUDA ;
\ Words to set SIO for terminal routines
: 1200BBS  1200BAUD  SETBBS ;
:  300BBS   300BAUD  SETBBS ;
: 1200PAK  1200BAUD  SETPAK ;
:  300PAK   300BAUD  SETPAK ;

\ Screen 21
\  TOG-TXBIT  HANGUP  MBREAK WJR04MAY94
: TOG-TXBIT  ( time-ms mask)
   TXCTL @  TUCK XOR 5 SETSIOA
   SWAP ( time) MS  5 SETSIOA ;
: HANGUP    1200 128 TOG-TXBIT ; \ drop DTR to hangup modem.
: MBREAK  100  16 TOG-TXBIT ; \ Send Break to modem.
HEX  : DRV-OFF    1C PC@ 40 OR 1C PC! ;  DECIMAL
\ Turns off Kaypro II floppy drive motors.

               ====  End Code  ====
```

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

32-Bit Systems

All Readers

TCP/IP & WIN

Real Computing

By Rick Rodman

Tiny-TCP

I've fixed a number of minor bugs in the FTP client code of Tiny-TCP, and now it can retrieve files from a server just fine. The next step is writing the FTP server side code. I don't expect too much difficulty, so this code should be on the BBS when you're reading this. Also, I can E-mail it to you if you send me E-mail, or mail you a diskette if you send it to me formatted with return postage.

FTP is a very simple package which uses ASCII messages to transfer commands and results to and from a "control port", port 21, while the data is transferred through another port. These "ports" are simple numbers used by the TCP layer to identify endpoints within a specific machine. Tiny-TCP performs a sort of multitasking by means of a polling loop in which it calls a function called "application".

There are two areas where porting the code could be a problem: first, in the accessing of files; second, in the serial I/O logic. The first area is mostly a filename syntax and directory format issue. For example, if you retrieve a file "/usr/rickr/photocd/photocd-1.24.doc" from a Sun, and want to save it on a CP/M system, the file saving logic should have additional logic to remove the directory path and truncate the filename. There are also some variations in C compiler file I/O function call syntax you may have to deal with.

As far as the serial I/O logic is concerned, it's unlikely (unless you're running CP/M Plus or MP/M) that your serial driver has any interrupt-driven buffering or a standard interface method. Tilmann Reh has suggested using a jump table so that "code overlays" can be added to the program for this logic, much as was done with Modem7. This seems like a good idea. Speed-wise, in a point-to-point configuration each link could be a different speed, but I recommend using 9600 baud for everything. If you don't have interrupts on the serial I/O, you probably won't be able to go that fast without losing characters; you might be able to run 1200 baud.

I've already discussed the real-time clock. This clock is used for timeouts. These timeouts are not very critical, and can be implemented by incrementing a subcounter which, when it reaches an empirically-determined value, increments the seconds value. If your machine has a real-time clock, of course, by all means use it. Because of the fact that the software uses a big polling loop, do *not* do anything that will take a substantial amount of time.

One of my ports will be to the Amiga. I'm aware that Matt Dillon has a fine package on the Amiga, called DNET, and that KA9Q has also been ported. However, the machine I have is an old A1000 with no hard drive, and I'm sure that most of the Amigas out there are a similar configuration, so mine, and those, have no way of using those larger packages. There is a nice C compiler, also from Matt, called DICE, which can be run on a floppy-only Amiga, and this is what I plan to use.

I'm also planning to port to the DEC Rainbow. The Rainbow is an unusual computer with two CPUs running simultaneously. It appears from the schematics that only the Z-80 can talk directly to the serial ports, so Tiny-TCP will need to be running on that processor. By the way, Rainbows and their similar-looking kin, the PDP-8-based Decmate III and the LSI-11-based Professional 350, are common in the used market at giveaway prices. Rather ironic, considering their original prices!

Another Bus Bites the Dust

The time has apparently come that even IBM has had to recognize the imminent death of Micro Channel. In some ways, Micro Channel was ahead of its time; but in other ways, it was very annoying. Yes, you could configure your boards through software. But the way you had to do it, booting and re-booting the Reference Diskette, over and over, sometimes five or six times, was intolerably cumbersome. And Micro Channel boards were very expensive, costing twice or more as much as the equivalent ISA board.

Now that IBM has admitted its willingness to let MCA fade away, we're probably going to see lots of PS/2s at flea markets and hamfests. If you're tempted by a pretty one, try to think of it as a generic DOS box - fine for word processing, but don't plan to put any boards in it.

Conventional wisdom regarding Windows programming

Those of you who, like me, have to make a buck by programming for Windows, have always been told to stay away from Large Model. This is because Windows can't move the segments once they are loaded. If you don't know what I'm talking about, skip to the next section.

I'm here to tell you, it ain't necessarily so. The bad thing about small-model EXEs and DLLs is that the whole module gets loaded in one fell swoop. This means that your memory requirements are actually *maximized*, not minimized, by using small model. This can lead to memory problems, especially low-memory problems. Windows will not be able to unload *any* of your EXE or DLL. Also, your loading time is maximized - everything must be loaded at once.

You read it here, and you'll probably never see it anywhere else: The way to do Windows is to use *large model!* But not just that. You also have to use *named code segments* by using the "-NT" option of the compile step, and give each module a distinct name. Then, you have to list each segment in a SEGMENTS section of your DEF file, and give it the attributes "MOVEABLE [sic] LOADONCALL DISCARDABLE". This will cause each to become a separate segment in the EXE or DLL file. Finally, you have to *disable* the segment optimization by adding the "-K" option on the RC line. RC will complain about having to convert some segments to PRELOAD. By moving things around, you should try to minimize the number of segments marked as PRELOAD, but some will almost always get marked that way.

I realize that all of this is the *exact opposite* of what everyone else is telling you, but it really works. Send E-mail or letters if you need more help. Don't be ashamed to ask, either. Microsoft Windows has got to be the most bizarre, idiosyncratic target environment ever developed - even 1802 machine code on the COSMAC VIP would be less byzantine.

News Items

The latest thing in the Minix world is that Minix has been ported to the Transputer chip by some folks in Germany. Remember the Transputer? It was Inmos' attempt to put a Lamborghini in every driveway.

The AT&T-clean version of Unix called "NetBSD" has been ported to the PC-532, and minor cleanup and debugging is underway. This is a full Unix operating system which will require 8 megabytes of RAM, which means that a patch is necessary to the ROM monitor. Distribution may be done by tape.

Can you believe the software price wars? FoxPro for $99! Quattro Pro for $49! OS/2 for Windows for $29! (By the way, OS/2 for Windows is a great package. Instead of Win-OS/2, you run real Windows - including our old friend Dr. Watson.) But the low point of the price war to date is Simply Tax for free. I did my taxes with this package - it's very good. Now a couple of other software packages are following this lead, for example, File Saver. My guess is that everyone hopes to build market share and then sell upgrades to a substantial percentage of their user base.

Rumors are abounding about Commodore, Philips, and Hewlett-Packard. Picture an Amiga 5000 with a PA-RISC and a built-in Video Toaster - running Windows NT. Don't be in a big hurry to plunk down plastic for a Power Mac.

It may be asked, what is the Real Computing view of these various RISC processors? As *TCJ* readers, we have to reserve judgement until we see the assembly language instruction set of each processor. Otherwise, what else is there to go on besides marketing hype? The only one I've seen so far is the Motorola 88000. Manufacturers are invited to submit instruction set listings to Real Computing c/o *TCJ*.

I've moved the BBS to an IBM PS/2 running The Major BBS. I'm sorry this isn't a 32-bit platform, but I have other projects underway. At some point I'd like to move it to a multitasking platform and integrate it with document management and text retrieval systems.

Next time

I didn't get to the Linux TCP/IP owing to pressing priorities in other areas. However, TCP/IP is becoming a high priority in a surprising number of areas of my life, both professional and personal. It seems that TCP/IP has become the de facto protocol for *internetworking* - connecting smaller LANs together.

Here in the Nation's Capital, the National Information Infostructure (NII) is a popular topic of conversation. Actually it's not that nobody knows what it is, it's that everybody conceives of it as their own pet system. Everyone wants to provide it - the cable TV companies, the phone companies, the computer companies, the satellite companies. The only thing people seem to agree on is that it will use TCP/IP, or it will use OSI (X.400 and X.500), or an admixture of both, ... or something else.

Anyway, you can expect that a lot of TCP/IP and X Window will be forthcoming in Real Computing. I'm having fun interconnecting my little machines so they can be players just like the big boys in the NII, whatever it turns out to be - if anything.

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

TINY-TCP.ZIP and FTP information continued on next two pages (38 & 39).

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

Archive: a:TINYTCP.ZIP

Name	Length	Mod Date
ARP.C	3198	13 Dec 93
MAIN.C	1654	03 Apr 94
SED.C	7102	13 Dec 93
SEDSLIP.C	6515	13 Feb 94
TINYFTP.C	9105	03 Apr 94
TINYTCP.C	24539	03 Apr 94
CASYNCMS.H	270	14 Dec 93
PROTO.H	2098	13 Feb 94
SED.H	2979	13 Feb 94
TINYTCP.H	6642	13 Dec 93
BU.BAT	126	13 Feb 94
D.BAT	54	13 Dec 93
GRAPH.BAT	181	08 Dec 93
X.BAT	20	03 Apr 94
DOS.MAK	1387	14 Dec 93
README.TXT	2429	13 Feb 94
CASYNCMS.ASM	6737	14 Dec 93
*total 17	75036	

TinyTcp Public Domain Release

The files in this release contain a simple implementation of TCP & FTP, suitable for burning into ROM. It is, in effect, a big hack put together in two or three days. It works for us, though, and you might like it, too. Warning: the code was intended for a 68000, and doesn't have any byte swapping support in it. Shouldn't be too hard to add, though.

- Geof Cooper
Imagen Corporation
[imagen!geof@decwrl.dec.com]
April 16, 1986

The package requires some system support:

clock_ValueRough() - should be a procedure that returns the current value of a millisecond clock. The procedure is called frequently, so that interrupts are not needed to service the clock. Our implementation polls the real time timer and assumes that it is called frequently enough so that it doesn't miss clock ticks (Since the timer is only used for network timeouts, it doesn't really matter if it does miss clock ticks, of course). Systems without a clock could probably get by with a procedure that increments a static variable and returns it, by adjusting the timeout constants in the program.

Network driver - some network interface driver is needed. A driver for a 3Com multibus (ethernet) board is included; this board isn't made anymore (at least not by 3Com), so you'll probably need to write a driver for the board in your system.

Guide to source files:

sed.c - Simple Ethernet Driver - Driver for 3Com multibus card. If you have another type of Ethernet board, you can use this driver as a template.

sed.h - header file for the above.

arp.c - Implementation of Address Resolution Protocol. Note that there is no arp "mapping" per se. The higher level code (tcp, in this case) is required to keep track of internet and ethernet addresses.

tinytcp.c - Implementation of TCP.

tinytcp.h - Header file for above, and for everything else.

tinyftp.c - Implementation of FTP, only allows files to be retrieved, not sent.

Notes from R. Rodman:

While the above says 'public domain release', all of the files contain copyright notices.

940213 The TCP layer appears to be working now, after correction of a couple of minor errors. However, the FTP server does not respond when I send a message (e.g. HELP), and the FTP layer here sits waiting for a response.

****** FTP.TXT

Network Working Group
Request for Comments: 959
J. Postel
J. Reynolds
ISI
Obsoletes RFC: 765 (IEN 149)
October 1985

FILE TRANSFER PROTOCOL (FTP)

Status of this Memo

This memo is the official specification of the File Transfer Protocol (FTP). Distribution of this memo is unlimited.

The following new optional commands are included in this edition of the specification:

CDUP (Change to Parent Directory), SMNT (Structure Mount), STOU (Store Unique), RMD (Remove Directory), MKD (Make Directory), PWD (Print Directory), and SYST (System).

Note that this specification is compatible with the previous edition.

1. INTRODUCTION

The objectives of FTP are 1) to promote sharing of files (computer programs and/or data), 2) to encourage indirect or implicit (via programs) use of remote computers, 3) to shield a user from variations in file storage systems among hosts, and 4) to transfer data reliably and efficiently. FTP, though usable directly by a user at a terminal, is designed mainly for use by programs.

**** TCP.TXT
RFC: 793

TRANSMISSION CONTROL PROTOCOL
DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION
September 1981

by
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, California 90291

This document describes the DoD Standard Transmission Control Protocol (TCP). There have been nine earlier editions of the ARPA TCP specification on which this standard is based, and the present text draws heavily from them. There have been many contributors to this work both in terms of concepts and in terms of text. This edition clarifies several details and removes the end-of-letter buffer-size adjustments, and redescribes the letter mechanism as a push function.
Jon Postel
Editor

RFC: 793
Replaces: RFC 761
IENs: 129, 124, 112, 81,
55, 44, 40, 27, 21, 5

PROTOCOL SPECIFICATION
1. INTRODUCTION

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks.

This document describes the functions to be performed by the Transmission Control Protocol, the program that implements it, and its interface to programs or users that require its services.

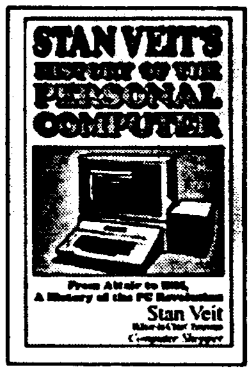
TCP is a connection-oriented, end-to-end reliable protocol designed to fit into a layered hierarchy of protocols which support multi-network applications. The TCP provides for reliable inter-process communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks.

The TCP fits into a layered protocol architecture just above a basic Internet Protocol [2] which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram "envelopes". The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentation or reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting gateways. The internet protocol also carries information on the precedence, security classification and compartmentation of the TCP segments, so this information can be communicated end-to-end across multiple networks.

```
          Protocol Layering
+------------------------------+
|         higher-level         |
+------------------------------+
|             TCP              |
+------------------------------+
|       internet protocol      |
+------------------------------+
|     communication network    |
+------------------------------+
```

NOW AVAILABLE!

"...a must addition to the library of every computer enthusiast. Highly recommended."
—John C. Dvorak, PC Magazine



Stan Veit's HISTORY OF THE PERSONAL COMPUTER

Order from: WORLDCOMM® 1-800-472-0438
65 Macedonia Road, Alexander NC 28701
(Visa and MasterCard accepted)
N.C. Residents Add 6% Sales Tax

Please Send:

____**Hardback Copies of *History***	**@ $27.95 plus $3.00 S&H**
____**Paperback Copies of *History***	**@ $19.95 plus $3.00 S&H**

Name ______________________
Address ______________________
City __________ State____ Zip __________

Special Feature

Intermediate Users

Part 5: The Z80 Primitives

MOVING FORTH

by Brad Rodriguez

THE CODE I PROMISED

At long last, I am ready to present the complete source code for an (I hope) ANSI compliant Forth, CamelForth[1]. As an intellectual exercise – and to ensure a clear copyright -- I've written this code entirely from scratch. (Do you know how hard it is to *not* look at excellent code examples?) Of course, my experience with various Forths has no doubt influenced some design decisions.

Due to space limitations, the source code will be presented in four installments (if you can't wait, complete files will be on GEnie):

1. Z80 Forth "primitives," in assembler source
2. 8051 Forth "primitives," likewise
3. Z80/8051 high-level kernel, likewise
4. complete 6809 kernel, in metacompiler source

For CamelForth I'm trying to use exclusively public-domain tools: for the Z80, the Z80MR assembler under CP/M [3]; for the 8051, the A51 cross-assembler on an IBM PC [4], and for the 6809, my own metacompiler under F83 for CP/M, IBM PC, or Atari ST.

By "kernel" I mean the set of words that comprises a basic Forth system, including compiler and interpreter. For CamelForth this is the ANSI Forth Core word set, plus any non-ANSI words necessary to implement the Core word set. A Forth kernel is usually written partly in machine code (as CODE words), and partly in high-level Forth. The words which are written in machine code are called the "primitives," since, in the final analysis, the entire Forth system is defined in terms of just these words.

Exactly *which* words should be written in machine code? The selection of the optimal set of primitives is an interesting debate. A smaller set of primitives makes for easier porting, but poorer performance. I've been told that a set of 13 primitives is sufficient to define all of Forth -- a *very slow* Forth. eForth [2], designed for easy porting, had a more generous set of 31 primitives. My rules are these:

1. Fundamental arithmetic, logic, and memory operators are CODE.
2. If a word *can't* be easily or efficiently written (or written at all) in terms of other Forth words, it should be CODE (e.g., U<, RSHIFT).
3. If a simple word is used frequently, CODE may be worthwhile (e.g., NIP, TUCK).
4. If a word requires fewer bytes when written in CODE, do so (a rule I learned from Charles Curley).
5. If the processor includes instruction support for a word's function, put it in CODE (e.g. CMOVE or SCAN on a Z80 or 8086).
6. If a word juggles many parameters on the stack, but has relatively simple logic, it may be better in CODE, where the parameters can be kept in registers.
7. If the logic or control flow of a word is complex, it's probably better in high-level Forth.

For Z80 CamelForth I have a set of about 70 primitives. (See Table 1.) Having already decided on the Forth model and CPU usage (see my previous TCJ articles), I followed this development procedure:

1. Select the subset of the ANSI Core word set which will be primitives. (Subject to revision, of course.)
2. From the ANSI descriptions, write assembler definitions of these words, plus the processor initialization code.
3. Run this through the assembler, fixing source code errors.
4. Test that you can produce working machine code. I usually add a few lines of assembler code to output a character once the initialization is complete. This seemingly trivial test is crucial! It ensures that your hardware, assembler, "downloader" (EPROM emulator or whatever), and serial communications are all working!
5. (Embedded systems only.) Add another assembler code fragment to read the serial port and echo it back...thus testing *both* directions of communications.
6. Write a *high-level* Forth fragment to output a character, using *only* Forth primitives. (Usually something like "LIT,33h,EMIT,BYE".) This tests the Forth register initialization, the stacks, and the threading mechanism. Problems at this stage can usually be traced to logic errors in NEXT or in the initialization, or data stack goofs (e.g. stack in ROM).
7. Write a colon definition to output a character, and include it in the high-level fragment from step 6. (E.g., define BLIP as "LIT,34h,EMIT,EXIT" and then test the fragment "LIT,33h,EMIT, BLIP,BYE".) Problems at this stage are usually with DOCOLON or EXIT logic, or return stack goofs.

8. At this point you can write some tools to help you with debugging, such as words to display in hex a number on the stack. Listing 1 shows a simple test routine to do a never-ending memory dump (useful even if your keyboard doesn't work). This tests the primitives DUP, EMIT, EXIT, C@, ><, LIT, 1+, and BRANCH, as well as several levels of nesting. Plus, it doesn't use DO..LOOP, which are often difficult to get working. When this code works, you have some confidence that your basic Forth model is valid.

9. From here on it's just testing the remaining primitives -- DO..LOOP, UM/MOD, UM*, and DODOES are particularly tricky -- and adding high-level definitions. I like to get a rudimentary interpreter going next, so that I can test words interactively.

With this set of primitives you can begin writing Forth code. Sure, you have to use an assembler instead of a Forth compiler, but -- as Listing 1 suggests -- you can use high-level control flow and nesting to write useful code that would be more difficult to write in assembler.

READ THE CODE!

I've run out of abstractions for today. If you want to learn more about how a Forth kernel works and is written, study Listing 1. It follows the Forth convention for documentation:

WORD-NAME   stack in -- stack out   description

WORD-NAME is the name by which *Forth* knows the word. Often these names include peculiar ASCII characters, so an approximation must be used when defining assembler labels (such as ONEPLUS for the Forth word 1+).

stack in are the arguments this word expects to see on the stack, with the topmost stack item always on the right. stack out are the arguments this word will leave on the stack, likewise.

If the word has a return stack effect (other than nesting, that is), an additional return stack comment will be added after "R:"

stack in -- stack out   R: stack in -- stack out

ANSI Forth defines a number of useful abbreviations for stack arguments, such as "n" for a signed single-cell number, "u" for an unsigned single-cell number, "c" for a character, and so on. See Table 1.

REFERENCES

[1] Definition of a camel: a horse designed by committee.

[2] Ting, C. H., eForth Implementation Guide, July 1990, available from Offete Enterprises, 1306 South B Stret, San Mateo, CA 94402 USA.

[3] Z80MR, a Z80 Macro Assembler by Mike Rubenstein, is public-domain, available on the GEnie CP/M Roundtable as file Z80MR-A.LBR. Warning: do *not* use the supplied Z1.COM program, use only Z80MR and LOAD. Z1 has a problem with conditional jumps.

[4] A51, PseudoCorp's freeware Level 1 cross-assembler for the 8051, is available from the Realtime and Control Forth Board, (303) 278-0364, or on the GEnie Forth Roundtable as file A51.ZIP. PseudoCorp's commercial products are advertised here in TCJ.

Source code for Z80 CamelForth is available on GEnie's Forth Roundtable as file CAMEL80.ARC.

TABLE 1. GLOSSARY OF WORDS IN CAMEL80.AZM
Words which are (usually) written in CODE.

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 ANSI Forth document for more details.

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

NAME	stack in -- stack out	description
!	x a-addr --	store cell in memory
+	n1/u1 n2/u2 -- n3/u3	add n1+n2
+!	n/u a-addr --	add cell to memory
-	n1/u1 n2/u2 -- n3/u3	subtract n1-n2
<	n1 n2 -- flag	test n1<n2, signed
=	x1 x2 -- flag	test x1=x2
>	n1 n2 -- flag	test n1>n2, signed
>R	x -- R: -- x	push to return stack
?DUP	x -- 0 | x x	DUP if nonzero
@	a-addr -- x	fetch cell from memory
0<	n -- flag	true if TOS negative
0=	n/u -- flag	return true if TOS=0
1+	n1/u1 -- n2/u2	add 1 to TOS
1-	n1/u1 -- n2/u2	subtract 1 from TOS
2*	x1 -- x2	arithmetic left shift
2/	x1 -- x2	arithmetic right shift
AND	x1 x2 -- x3	logical AND
CONSTANT	n --	define a Forth constant
C!	c c-addr --	store char in memory
C@	c-addr -- c	fetch char from memory
DROP	x --	drop top of stack
DUP	x -- x x	duplicate top of stack
EMIT	c --	output character to console
EXECUTE	i*x xt -- j*x	execute Forth word 'xt'
EXIT	--	exit a colon definition
FILL	c-addr u c --	fill memory with char
I	-- n R: sys1 sys2 -- sys1 sys2	get the innermost loop index
INVERT	x1 -- x2	bitwise inversion
J	-- n R: 4*sys -- 4*sys	get the second loop index
KEY	-- c	get character from keyboard
LSHIFT	x1 u -- x2	logical L shift u places
NEGATE	x1 -- x2	two's complement
OR	x1 x2 -- x3	logical OR
OVER	x1 x2 -- x1 x2 x1	per stack diagram
ROT	x1 x2 x3 -- x2 x3 x1	per stack diagram
RSHIFT	x1 u -- x2	logical R shift u places
R>	-- x R: x --	pop from return stack
R@	-- x R: x -- x	fetch from rtn stk
SWAP	x1 x2 -- x2 x1	swap top two items
UM*	u1 u2 -- ud	unsigned 16x16->32 mult.
UM/MOD	ud u1 -- u2 u3	unsigned 32/16->16 div.
UNLOOP	-- R: sys1 sys2 --	drop loop parms
U<	u1 u2 -- flag	test u1<n2, unsigned
VARIABLE	--	define a Forth variable
XOR	x1 x2 -- x3	logical XOR

ANSI Forth Extensions

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

<>	x1 x2 -- flag	test not equal
BYE	i*x --	return to CP/M
CMOVE	c-addr1 c-addr2 u --	move from bottom
CMOVE>	c-addr1 c-addr2 u --	move from top
KEY?	-- flag	return true if char waiting
M+	d1 n -- d2	add single to double
NIP	x1 x2 -- x2	per stack diagram
TUCK	x1 x2 -- x2 x1 x2	per stack diagram
U>	u1 u2 -- flag	test u1>u2, unsigned

Private Extensions

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

(do)	n1|u1 n2|u2 -- R: -- sys1 sys2	run-time code for DO
(loop)	R: sys1 sys2 -- | sys1 sys2	run-time code for LOOP
(+loop)	n -- R: sys1 sys2 -- | sys1 sys2	run-time code for +LOOP
><	x1 -- x2	swap bytes
?branch	x --	branch if TOS zero
BDOS	DE C -- A	call CP/M BDOS
branch	--	branch always
lit	-- x	fetch inline literal to stack
PC!	c p-addr --	output char to port
PC@	p-addr -- c	input char from port
RP!	a-addr --	set return stack pointer
RP@	-- a-addr	get return stack pointer
SCAN	c-addr1 u1 c -- c-addr2 u2	find matching char
SKIP	c-addr1 u1 c -- c-addr2 u2	s kip matching chars
SP!	a-addr --	set data stack pointer
SP@	-- a-addr	get data stack pointer
S=	c-addr1 c-addr2 u -- n	string compare n<0: s1<s2, n=0: s1=s2, n>0: s1>s2
USER	n --	define user variable 'n'

```
; Listing 1.
;
===============================================
; CamelForth for the Zilog Z80
; Primitive testing code
;
; This is the "minimal" test of the CamelForth
; kernel.  It verifies the threading and nesting
; mechanisms, the stacks, and the primitives
;   DUP EMIT EXIT lit branch ONEPLUS.
; It is particularly useful because it does not
; use the DO..LOOP, multiply, or divide words,
; and because it can be used on embedded CPUs.
; The numeric display word .A is also useful
; for testing the rest of the Core wordset.
;
; The required macros and CPU initialization
; are in file CAMEL80.AZM.
;
===============================================

;Z ><  u1 -- u2   swap the bytes of TOS
   head SWAB,2,><,docode
      ld a,b
      ld b,c
      ld c,a
      next

;Z LO  c1 -- c2   return low nybble of TOS
   head LO,2,LO,docode
      ld a,c
      and 0fh
      ld c,a
      ld b,0
      next

;Z HI  c1 -- c2   return high nybble of TOS
   head HI,2,HI,docode
      ld a,c
      and 0f0h
      rrca
      rrca
      rrca
      rrca
      ld c,a
      ld b,0
      next

;Z >HEX  c1 -- c2   convert nybble to hex char
   head TOHEX,4,>HEX,docode
            ld a,c
            sub 0ah
            jr c,numeric
            add a,7
numeric:    add a,3ah
            ld c,a
            next

;Z .HH   c --      print byte as 2 hex digits
;   DUP HI >HEX EMIT LO >HEX EMIT ;
   head DOTHH,3,.HH,docolon
      DW DUP,HI,TOHEX,EMIT,LO,TOHEX,EMIT,EXIT

;Z .B   a -- a+1   fetch & print byte, advancing
;   DUP C@ .HH 20 EMIT 1+ ;
   head DOTB,2,.B,docolon
   DW DUP,CFETCH,DOTHH,lit,20h,EMIT,ONEPLUS,EXIT

;Z .A   u --      print unsigned as 4 hex digits
;   DUP >< .HH .HH 20 EMIT ;
   head DOTA,2,.A,docolon
      DW DUP,SWAB,DOTHH,DOTHH,lit,20h,EMIT,EXIT

;Z ZQUIT  --   endless dump for testing
;   0 BEGIN  0D EMIT 0A EMIT  DUP .A
;      .B .B .B .B .B .B .B .B
;      .B .B .B .B .B .B .B .B
;   AGAIN ;
   head ZQUIT,5,ZQUIT,docolon
     DW lit,0
quit1:  DW lit,0dh,EMIT,lit,0ah,EMIT,DUP,DOTA
     DW
DOTB,DOTB,DOTB,DOTB,DOTB,DOTB,DOTB,DOTB
     DW
DOTB,DOTB,DOTB,DOTB,DOTB,DOTB,DOTB,DOTB
     DW branch,quit1
```

```
; 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
;
; CAMEL80.AZM: Code Primitives
;   Source code is for the Z80MR macro assembler.
;   Forth words are documented as follows:
;x   NAME   stack -- stack    description
;   where x=C for ANSI Forth Core words, X for ANSI
;   Extensions, Z for internal or private words.
;
; Direct-Threaded Forth model for Zilog Z80
; 16 bit cell, 8 bit char, 8 bit (byte) adrs unit
;    Z80 BC = Forth TOS (top Param Stack item)
;          HL =     W       working register
;          DE =     IP      Interpreter Pointer
;          SP =     PSP     Param Stack Pointer
;          IX =     RSP     Return Stack Pointer
;          IY =     UP      User area Pointer
;    A, alternate register set = temporaries
;
===============================================
; Macros to define Forth headers
; HEAD  label,length,name,action
; IMMED label,length,name,action
;    label    = assembler name for this word
;               (special characters not allowed)
;    length   = length of name field
;    name     = Forth's name for this word
;    action   = code routine for this word, e.g.
;               DOCOLON, or DOCODE for code words
; IMMED defines a header for an IMMEDIATE word.
;
DOCODE  EQU 0        ; flag to indicate CODE words
link    DEFL 0       ; link to previous Forth word

head    MACRO   #label,#length,#name,#action
            DW link
            DB 0
link    DEFL $
            DB #length,'#name
#label:
            IF  .NOT.(#action=DOCODE)
            call #action
            ENDIF
            ENDM

immed   MACRO   #label,#length,#name,#action
            DW link
            DB 1
link    DEFL $
            DB #length,'#name'
#label:
            IF  .NOT.(#action=DOCODE)
            call #action
            ENDIF
            ENDM

; The NEXT macro (7 bytes) assembles the 'next'
; code in-line in every Z80 CamelForth CODE word.
next    MACRO
            ex de,hl
            ld e,(hl)
            inc hl
            ld d,(hl)
            inc hl
            ex de,hl
            jp (hl)
            ENDM

; NEXTHL is used when the IP is already in HL.
nexthl  MACRO
            ld e,(hl)
            inc hl
            ld d,(hl)
            inc hl
            ex de,hl
            jp (hl)
            ENDM

; RESET AND INTERRUPT VECTORS
===================
; ...are not used in the CP/M implementation
; Instead, we have the...

; CP/M ENTRY POINT
            org 100h
reset:      ld hl,(6h)   ; BDOS address, rounded down
            ld l,0       ;    = end of avail.mem (EM)
            dec h        ; EM-100h
            ld sp,hl     ;    = top of param stack
            inc h        ; EM
            push hl
            pop ix       ;  = top of return stack
            dec h        ; EM-200h
            dec h
            push hl
            pop iy       ;  = bottom of user area
            ld de,1      ; do reset if COLD returns
            jp COLD      ; enter top-level Forth word

; Memory map:
;   0080h    Terminal Input Buffer, 128 bytes
;   0100h    Forth kernel = start of CP/M TPA
;   ? h      Forth dictionary (user RAM)
;   EM-200h User area, 128 bytes
;   EM-180h Parameter stack, 128B, grows down
;   EM-100h HOLD area, 40 bytes, grows down
;   EM-0D8h  PAD buffer, 88 bytes
;   EM-80h   Return stack, 128 B, grows down
;   EM       End of RAM = start of CP/M BDOS
; See also the definitions of U0, S0, and R0
; in the "system variables & constants" area.
; A task w/o terminal input requires 200h bytes.
; Double all except TIB and PAD for 32-bit CPUs.

; INTERPRETER LOGIC
=============================
; See also "defining words" at end of this file

;C EXIT     --           exit a colon definition
   head EXIT,4,EXIT,docode
            ld e,(ix+0)   ; pop old IP from ret stk
            inc ix
            ld d,(ix+0)
            inc ix
            next

;Z lit        -- x   fetch inline literal to stack
; This is the primtive compiled by LITERAL.
   head lit,3,lit,docode
            push bc      ; push old TOS
            ld a,(de)    ; fetch cell at IP to TOS,
            ld c,a       ;             advancing IP
            inc de
            ld a,(de)
            ld b,a
            inc de
            next

;C EXECUTE   i*x xt -- j*x   execute Forth word
;C           at 'xt'
   head EXECUTE,7,EXECUTE,docode
```

```
        ld h,b      ; address of word -> HL
        ld l,c
        pop bc      ; get new TOS
        jp (hl)     ; go do Forth word

; DEFINING WORDS
==============================

; ENTER, a.k.a. DOCOLON, entered by CALL ENTER
; to enter a new high-level thread (colon def'n.)
; (internal code fragment, not a Forth word)
; N.B.: DOCOLON must be defined before any
; appearance of 'docolon' in a 'word' macro!
docolon:    ; (alternate name)
enter:  dec ix      ; push old IP on ret stack
        ld (ix+0),d
        dec ix
        ld (ix+0),e
        pop hl      ; param field adrs -> IP
        nexthl      ; use the faster 'nexthl'

;C VARIABLE   --      define a Forth variable
;   CREATE 1 CELLS ALLOT ;
; Action of RAM variable is identical to CREATE,
; so we don't need a DOES> clause to change it.
   head VARIABLE,8,VARIABLE,docolon
          DW CREATE,LIT,1,CELLS,ALLOT,EXIT
; DOVAR, code action of VARIABLE, entered by CALL
; DOCREATE, code action of newly created words
docreate:
dovar:                ; -- a-addr
          pop hl      ; parameter field address
          push bc     ; push old TOS
          ld b,h      ; pfa = variable's adrs -> TOS
          ld c,l
          next

;C CONSTANT  n --     define a Forth constant
;   CREATE , DOES> (machine code fragment)
   head CONSTANT,8,CONSTANT,docolon
          DW CREATE,COMMA,XDOES
; DOCON, code action of CONSTANT,
; entered by CALL DOCON
docon:                ; -- x
          pop hl      ; parameter field address
          push bc     ; push old TOS
          ld c,(hl)   ; fetch contents of parameter
          inc hl      ;   field -> TOS
          ld b,(hl)
          next

;Z USER    n --       define user variable 'n'
;   CREATE , DOES> (machine code fragment)
   head USER,4,USER,docolon
          DW CREATE,COMMA,XDOES
; DOUSER, code action of USER,
; entered by CALL DOUSER
douser:               ; -- a-addr
          pop hl      ; parameter field address
          push bc     ; push old TOS
          ld c,(hl)   ; fetch contents of parameter
          inc hl      ;   field
          ld b,(hl)
          push iy     ; copy user base address to HL
          pop hl
          add hl,bc   ;   and add offset
          ld b,h      ; put result in TOS
          ld c,l
          next

; DODOES, code action of DOES> clause
; entered byCALL fragment
;         parameter field
;         ...
;         fragment: CALL DODOES
;         high-level thread
; Enters high-level thread with address of
; parameter field on top of stack.
; (internal code fragment, not a Forth word)
dodoes:               ; -- a-addr
          dec ix      ; push old IP on ret stk
          ld (ix+0),d
          dec ix
          ld (ix+0),e
          pop de      ; adrs of new thread -> IP
          pop hl      ; adrs of parameter field
          push bc     ; push old TOS onto stack
          ld b,h      ; pfa -> new TOS
          ld c,l
          next

; CP/M TERMINAL I/O
============================
cpmbdos EQU 5h        ; CP/M BDOS entry point

;Z BDOS   de c -- a   call CP/M BDOS
   head BDOS,4,BDOS,docode
          ex de,hl    ; save important Forth regs
          pop de      ;  (DE,IX,IY) & pop DE value
          push hl
          push ix
          push iy
          call cpmbdos
          ld c,a      ; result in TOS
          ld b,0
          pop iy      ; restore Forth regs
          pop ix
          pop de
          next

;C EMIT    c --   output character to console
;   6 BDOS DROP ;
; warning: if c=0ffh, will read one keypress
   head EMIT,4,EMIT,docolon
          DW LIT,06H,BDOS,DROP,EXIT

;C KEY     -- c   get character from keyboard
;   BEGIN 0FF 6 BDOS ?DUP UNTIL ;
; must use CP/M direct console I/O to avoid echo
   head KEY,3,KEY,docolon
KEY1:     DW LIT,0FFH,LIT,06H,BDOS
          DW QDUP,qbranch,KEY1,EXIT

;X KEY?    -- f   return true if char waiting
;   xx 0B BDOS ;   xx=don't care  rtns 0 or FF
   head querykey,4,?KEY,docolon
          DW LIT,0BH,DUP,BDOS,EXIT

;X BYE     i*x --   return to CP/M
   head bye,3,bye,docode
          jp 0

; STACK OPERATIONS
=============================

;C DUP     x -- x x      duplicate top of stack
   head DUP,3,DUP,docode
pushtos:  push bc
          next

;C ?DUP    x -- 0 | x x   DUP if nonzero
   head QDUP,4,?DUP,docode
          ld a,b
          or c
          jr nz,pushtos
          next

;C DROP    x --          drop top of stack
   head DROP,4,DROP,docode
poptos:   pop bc
          next

;C SWAP   x1 x2 -- x2 x1   swap top two items
   head SWOP,4,SWAP,docode
          pop hl
          push bc
          ld b,h
          ld c,l
          next

;C OVER   x1 x2 -- x1 x2 x1   per stack diagram
   head OVER,4,OVER,docode
          pop hl
          push hl
          push bc
          ld b,h
          ld c,l
          next

;C ROT   x1 x2 x3 -- x2 x3 x1   per stack diagram
   head ROT,3,ROT,docode
                        ; x3 is in TOS
          pop hl        ; x2
          ex (sp),hl    ; x2 on stack, x1 in hl
          push bc
          ld b,h
          ld c,l
          next

;X NIP   x1 x2 -- x2      per stack diagram
   head NIP,3,NIP,docolon
          DW SWOP,DROP,EXIT

;X TUCK  x1 x2 -- x2 x1 x2        per stack diagram
   head TUCK,4,TUCK,docolon
          DW SWOP,OVER,EXIT

;C >R   x --  R: -- x   push to return stack
   head TOR,2,>R,docode
          dec ix        ; push TOS onto rtn stk
          ld (ix+0),b
          dec ix
          ld (ix+0),c
          pop bc        ; pop new TOS
          next

;C R>    -- x   R: x --   pop from return stack
   head RFROM,2,R>,docode
          push bc       ; push old TOS
          ld c,(ix+0)   ; pop top rtn stk item
          inc ix        ;          to TOS
          ld b,(ix+0)
          inc ix
          next

;C R@    -- x           R: x -- x   fetch from rtn stk
   head RFETCH,2,R@,docode
          push bc       ; push old TOS
          ld c,(ix+0)   ; fetch top rtn stk item
          ld b,(ix+1)   ;          to TOS
          next

;Z SP@   -- a-addr       get data stack pointer
   head SPFETCH,3,SP@,docode
          push bc
          ld hl,0
          add hl,sp
          ld b,h
          ld c,l
          next

;Z SP!  a-addr --        set data stack pointer
   head SPSTORE,3,SP!,docode
          ld h,b
          ld l,c
          ld sp,hl
          pop bc        ; get new TOS
          next

;Z RP@   -- a-addr       get return stack pointer
   head RPFETCH,3,RP@,docode
          push bc
          push ix
          pop bc
          next

;Z RP!  a-addr --        set return stack pointer
   head RPSTORE,3,RP!,docode
          push bc
          pop ix
          pop bc
          next

; MEMORY AND I/O OPERATIONS
====================

;C !        x a-addr --   store cell in memory
   head STORE,1,!,docode
          ld h,b        ; address in hl
          ld l,c
          pop bc        ; data in bc
          ld (hl),c
          inc hl
          ld (hl),b
          pop bc        ; pop new TOS
          next

;C C!       char c-addr --   store char in memory
   head CSTORE,2,C!,docode
          ld h,b        ; address in hl
          ld l,c
          pop bc        ; data in bc
          ld (hl),c
          pop bc        ; pop new TOS
          next

;C @        a-addr -- x   fetch cell from memory
   head FETCH,1,@,docode
          ld h,b        ; address in hl
          ld l,c
          ld c,(hl)
          inc hl
          ld b,(hl)
          next

;C C@       c-addr -- char   fetch char from memory
   head CFETCH,2,C@,docode
          ld a,(bc)
          ld c,a
          ld b,0
          next

;Z PC!      char c-addr --   output char to port
   head PCSTORE,3,PC!,docode
          pop hl        ; char in L
          out (c),l     ; to port (BC)
          pop bc        ; pop new TOS
          next
```

```
;Z PC@      c-addr -- char   input char from port
   head PCFETCH,3,PC@,docode
            in c,(c)     ; read port (BC) to C
            ld b,0
            next

; ARITHMETIC AND LOGICAL OPERATIONS
=============

;C +        n1/u1 n2/u2 -- n3/u3    add n1+n2
   head PLUS,1,+,docode
            pop hl
            add hl,bc
            ld b,h
            ld c,l
            next

;X M+       d n -- d     add single to double
   head MPLUS,2,M+,docode
            ex de,hl
            pop de       ; hi cell
            ex (sp),hl   ; lo cell, save IP
            add hl,bc
            ld b,d       ; hi result in BC (TOS)
            ld c,e
            jr nc,mplus1
            inc bc
mplus1: pop de           ; restore saved IP
            push hl      ; push lo result
            next

;C -        n1/u1 n2/u2 -- n3/u3   subtract n1-n2
   head MINUS,1,-,docode
            pop hl
            or a
            sbc hl,bc
            ld b,h
            ld c,l
            next

;C AND    x1 x2 -- x3     logical AND
   head AND,3,AND,docode
            pop hl
            ld a,b
            and h
            ld b,a
            ld a,c
            and l
            ld c,a
            next

;C OR       x1 x2 -- x3   logical OR
   head OR,2,OR,docode
            pop hl
            ld a,b
            or h
            ld b,a
            ld a,c
            or l
            ld c,a
            next

;C XOR    x1 x2 -- x3     logical XOR
   head XOR,3,XOR,docode
            pop hl
            ld a,b
            xor h
            ld b,a
            ld a,c
            xor l
            ld c,a
            next

;C INVERT   x1 -- x2     bitwise inversion
   head INVERT,6,INVERT,docode
            ld a,b
            cpl
            ld b,a
            ld a,c
            cpl
            ld c,a
            next

;C NEGATE   x1 -- x2    two's complement
   head NEGATE,6,NEGATE,docode
            ld a,b
            cpl
            ld b,a
            ld a,c
            cpl
            ld c,a
            inc bc
            next

;C 1+       n1/u1 -- n2/u2          add 1 to TOS
   head ONEPLUS,2,1+,docode
```

```
            inc bc
            next

;C 1-       n1/u1 -- n2/u2          subtract 1 from TOS
   head ONEMINUS,2,1-,docode
            dec bc
            next

;Z ><       x1 -- x2     swap bytes (not ANSI)
   head swapbytes,2,><,docode
            ld a,b
            ld b,c
            ld c,a
            next

;C 2*       x1 -- x2     arithmetic left shift
   head TWOSTAR,2,2*,docode
            sla c
            rl b
            next

;C 2/       x1 -- x2     arithmetic right shift
   head TWOSLASH,2,2/,docode
            sra b
            rr c
            next

;C LSHIFT  x1 u -- x2   logical L shift u places
   head LSHIFT,6,LSHIFT,docode
            ld b,c       ; b = loop counter
            pop hl       ;   NB: hi 8 bits ignored!
            inc b        ; test for counter=0 case
            jr lsh2
lsh1:       add hl,hl    ; left shift HL, n times
lsh2:       djnz lsh1
            ld b,h       ; result is new TOS
            ld c,l
            next

;C RSHIFT  x1 u -- x2   logical R shift u places
   head RSHIFT,6,RSHIFT,docode
            ld b,c       ; b = loop counter
            pop hl       ;   NB: hi 8 bits ignored!
            inc b        ; test for counter=0 case
            jr rsh2
rsh1:       srl h        ; right shift HL, n times
            rr l
rsh2:       djnz rsh1
            ld b,h       ; result is new TOS
            ld c,l
            next

;C +!       n/u a-addr --           add cell to memory
   head PLUSSTORE,2,+!,docode
            pop hl
            ld a,(bc)    ; low byte
            add a,l
            ld (bc),a
            inc bc
            ld a,(bc)    ; high byte
            adc a,h
            ld (bc),a
            pop bc       ; pop new TOS
            next

; COMPARISON OPERATIONS
========================

;C 0=       n/u -- flag    return true if TOS=0
   head ZEROEQUAL,2,0=,docode
            ld a,b
            or c         ; result=0 if bc was 0
            sub 1        ; cy set   if bc was 0
            sbc a,a      ; propagate cy through A
            ld b,a       ; put 0000 or FFFF in TOS
            ld c,a
            next

;C 0<       n -- flag     true if TOS negative
   head ZEROLESS,2,0<,docode
            sla b        ; sign bit -> cy flag
            sbc a,a      ; propagate cy through A
            ld b,a       ; put 0000 or FFFF in TOS
            ld c,a
            next

;C =        x1 x2 -- flag test x1=x2
   head EQUAL,1,=,docode
            pop hl
            or a
            sbc hl,bc    ; x1-x2 in HL, SZVC valid
            jr z,tostrue
tosfalse:   ld bc,0
            next

;X <>       x1 x2 -- flag    test not eq (not ANSI)
```

```
   head NOTEQUAL,2,<>,docolon
            DW EQUAL,ZEROEQUAL,EXIT

;C <        n1 n2 -- flag           test n1<n2, signed
   head LESS,1,<,docode
            pop hl
            or a
            sbc hl,bc      ; n1-n2 in HL, SZVC valid
; if result negative & not OV, n1<n2
; neg. & OV => n1 +ve, n2 -ve, rslt -ve, so n1>n2
; if result positive & not OV, n1>=n2
; pos. & OV => n1 -ve, n2 +ve, rslt +ve, so n1<n2
; thus OV reverses the sense of the sign bit
            jp pe,revsense  ; if OV, use rev. sense
            jp p,tosfalse   ;   if +ve, result false
tostrue:    ld bc,0ffffh    ;   if -ve, result true
            next
revsense:   jp m,tosfalse   ; OV: if -ve, reslt false
            jr tostrue      ;   if +ve, result true

;C >        n1 n2 -- flag           test n1>n2, signed
   head GREATER,2,>,docolon
            DW SWOP,LESS,EXIT

;C U<    u1 u2 -- flag    test u1<n2, unsigned
   head ULESS,2,U<,docode
            pop hl
            or a
            sbc hl,bc    ; u1-u2 in HL, SZVC valid
            sbc a,a      ; propagate cy through A
            ld b,a       ; put 0000 or FFFF in TOS
            ld c,a
            next

;X U>    u1 u2 -- flag     u1>u2 unsgd (not ANSI)
   head UGREATER,2,U>,docolon
            DW SWOP,ULESS,EXIT

; LOOP AND BRANCH OPERATIONS
===================

;Z branch    --             branch always
   head branch,6,branch,docode
dobranch:   ld a,(de)    ; get inline value => IP
            ld l,a
            inc de
            ld a,(de)
            ld h,a
            nexthl

;Z ?branch   x --           branch if TOS zero
   head qbranch,7,?branch,docode
            ld a,b
            or c         ; test old TOS
            pop bc       ; pop new TOS
            jr z,dobranch   ; if old TOS=0, branch
            inc de       ; else skip inline value
            inc de
            next

;Z (do)    n1|u1 n2|u2 --  R: -- sys1 sys2
;Z             run-time code for DO
; '83 and ANSI standard loops terminate when the
; boundary of limit-1 and limit is crossed, in
; either direction.  This can be conveniently
; implemented by making the limit 8000h, so that
; arithmetic overflow logic can detect crossing.
; I learned this trick from Laxen & Perry F83.
; fudge factor = 8000h-limit, to be added to
; the start value.
   head xdo,4,(do),docode
            ex de,hl
            ex (sp),hl   ; IP on stack, limit in HL
            ex de,hl
            ld hl,8000h
            or a
            sbc hl,de    ; 8000-limit in HL
            dec ix       ; push this fudge factor
            ld (ix+0),h  ;   onto return stack
            dec ix       ;   for later use by 'I'
            ld (ix+0),l
            add hl,bc    ; add fudge to start value
            dec ix       ; push adjusted start value
            ld (ix+0),h  ;   onto return stack
            dec ix       ;   as the loop index.
            ld (ix+0),l
            pop de       ; restore the saved IP
            pop bc       ; pop new TOS
            next

;Z (loop)   R: sys1 sys2 --  | sys1 sys2
;Z             run-time code for LOOP
; Add 1 to the loop index.  If loop terminates,
; clean up the return stack and skip the branch.
; Else take the inline branch.  Note that LOOP
; terminates when index=8000h.
```

```
    head xloop,6,(loop),docode
            exx
            ld bc,1
looptst: ld l,(ix+0)  ; get the loop index
            ld h,(ix+1)
            or a
            adc hl,bc   ; increment w/overflow test
            jp pe,loopterm  ; overflow=loop done
            ; continue the loop
            ld (ix+0),l  ; save the updated index
            ld (ix+1),h
            exx
            jr dobranch  ; take the inline branch
loopterm: ; terminate the loop
            ld bc,4       ; discard the loop info
            add ix,bc
            exx
            inc de        ; skip the inline branch
            inc de
            next

;Z (+loop)  n --  R: sys1 sys2 -- | sys1 sys2
;Z          run-time code for +LOOP
; Add n to the loop index.  If loop terminates,
; clean up the return stack and skip the branch.
; Else take the inline branch.
  head xplusloop,7,(+loop),docode
          pop hl      ; this will be the new TOS
          push bc
          ld b,h
          ld c,l
          exx
          pop bc      ; old TOS = loop increment
          jr looptst

;C I        -- n   R: sys1 sys2 -- sys1 sys2
;C          get the innermost loop index
  head II,1,I,docode
          push bc     ; push old TOS
          ld l,(ix+0) ; get current loop index
          ld h,(ix+1)
          ld c,(ix+2) ; get fudge factor
          ld b,(ix+3)
          or a
          sbc hl,bc   ; subtract fudge factor,
          ld b,h      ;   returning true index
          ld c,l
          next

;C J        -- n   R: 4*sys -- 4*sys
;C          get the second loop index
  head JJ,1,J,docode
          push bc     ; push old TOS
          ld l,(ix+4) ; get current loop index
          ld h,(ix+5)
          ld c,(ix+6) ; get fudge factor
          ld b,(ix+7)
          or a
          sbc hl,bc   ; subtract fudge factor,
          ld b,h      ;   returning true index
          ld c,l
          next

;C UNLOOP  --  R: sys1 sys2 --  drop loop parms
  head UNLOOP,6,UNLOOP,docode
          inc ix
          inc ix
          inc ix
          inc ix
          next

; MULTIPLY AND DIVIDE
==========================

;C UM*     u1 u2 -- ud   unsigned 16x16->32 mult.
  head UMSTAR,3,UM*,docode
          push bc
          exx
          pop bc      ; u2 in BC
          pop de      ; u1 in DE
          ld hl,0     ; result will be in HLDE
          ld a,17     ; loop counter
          or a        ; clear cy
umloop: rr h
          rr l
          rr d
          rr e
          jr nc,noadd
          add hl,bc
noadd:  dec a
          jr nz,umloop
          push de     ; lo result
          push hl     ; hi result
          exx
          pop bc      ; ut TOS back in BC
          next

;C UM/MOD   ud u1 -- u2 u3   unsigned 32/16->16
  head UMSLASHMOD,6,UM/MOD,docode
            push bc
            exx
            pop bc      ; BC = divisor
            pop hl      ; HLDE = dividend
            pop de
            ld a,16     ; loop counter
            sla e
            rl d        ; hi bit DE -> carry
udloop:     adc hl,hl   ; rot left w/ carry
            jr nc,udiv3
                        ; case 1: 17 bit, cy:HL = 1xxxx
            or a        ; we know we can subtract
            sbc hl,bc
            or a        ; clear cy to indicate sub ok
            jr udiv4
                        ; case 2: 16 bit, cy:HL = 0xxxx
udiv3:      sbc hl,bc   ; try the subtract
            jr nc,udiv4 ; if no cy, subtract ok
            add hl,bc   ; else cancel the subtract
            scf         ;   and set cy to indicate
udiv4:      rl e        ; rotate result bit into DE,
            rl d        ; and next bit of DE into cy
            dec a
            jr nz,udloop
            ; now have complemented quotient in DE,
            ; and remainder in HL
            ld a,d
            cpl
            ld b,a
            ld a,e
            cpl
            ld c,a
            push hl     ; push remainder
            push bc
            exx
            pop bc      ; quotient remains in TOS
            next

; BLOCK AND STRING OPERATIONS
==================

;C FILL   c-addr u char --  fill memory with char
  head FILL,4,FILL,docode
            ld a,c      ; character in a
            exx         ; use alt. register set
            pop bc      ; count in bc
            pop de      ; address in de
            or a        ; clear carry flag
            ld hl,0ffffh
            adc hl,bc   ; test for count=0 or 1
            jr nc,filldone ;   no cy: count=0, skip
            ld (de),a   ; fill first byte
            jr z,filldone  ;   zero, count=1, done
            dec bc      ; else adjust count,
            ld h,d      ;   let hl = start adrs,
            ld l,e
            inc de      ;   let de = start adrs+1
            ldir        ;   copy (hl)->(de)
filldone:   exx         ; back to main reg set
            pop bc      ; pop new TOS
            next

;X CMOVE   c-addr1 c-addr2 u --  move from bottom
; as defined in the ANSI optional String word set
; On byte machines, CMOVE and CMOVE> are logical
; factors of MOVE.  They are easy to implement on
; CPUs which have a block-move instruction.
  head CMOVE,5,CMOVE,docode
            push bc
            exx
            pop bc      ; count
            pop de      ; destination adrs
            pop hl      ; source adrs
            ld a,b      ; test for count=0
            or c
            jr z,cmovedone
            ldir        ; move from bottom to top
cmovedone: exx
            pop bc      ; pop new TOS
            next

;X CMOVE>  c-addr1 c-addr2 u --  move from top
; as defined in the ANSI optional String word set
            push bc
            exx
            pop bc      ; count
            pop hl      ; destination adrs
            pop de      ; source adrs
            ld a,b      ; test for count=0
            or c
            jr z,dmovedone
            add hl,bc   ; last byte in destination
            dec hl
            ex de,hl
            add hl,bc   ; last byte in source
            dec hl
            lddr        ; move from top to bottom
dmovedone: exx
            pop bc      ; pop new TOS
            next

;Z SKIP   c-addr u c -- c-addr' u'
;Z          skip matching chars
; Although SKIP, SCAN, and S= are perhaps not the
; ideal factors of WORD and FIND, they closely
; follow the string operations available on many
; CPUs, and so are easy to implement and fast.
  head skip,4,SKIP,docode
            ld a,c      ; skip character
            exx
            pop bc      ; count
            pop hl      ; address
            ld e,a      ; test for count=0
            ld a,b
            or c
            jr z,skipdone
            ld a,e
skiploop:   cpi
            jr nz,skipmis   ; char mismatch: exit
            jp pe,skiploop  ; count not exhausted
            jr skipdone  ; count 0, no mismatch
skipmis:    inc bc      ; mismatch!  undo last to
            dec hl      ;   point at mismatch char
skipdone:   push hl     ; updated address
            push bc     ; updated count
            exx
            pop bc      ; TOS in bc
            next

;Z SCAN   c-addr u c -- c-addr' u'
;Z          find matching char
  head scan,4,SCAN,docode
            ld a,c      ; scan character
            exx
            pop bc      ; count
            pop hl      ; address
            ld e,a      ; test for count=0
            ld a,b
            or c
            jr z,scandone
            ld a,e
            cpir        ; scan 'til match or count=0
            jr nz,scandone  ; no match, BC & HL ok
            inc bc      ; match!  undo last to
            dec hl      ;   point at match char
scandone:   push hl     ; updated address
            push bc     ; updated count
            exx
            pop bc      ; TOS in bc
            next

;Z S=   c-addr1 c-addr2 u -- n   string compare
;Z          n<0: s1<s2, n=0: s1=s2, n>0: s1>s2
  head sequal,2,S=,docode
            push bc
            exx
            pop bc      ; count
            pop hl      ; addr2
            pop de      ; addr1
            ld a,b      ; test for count=0
            or c
            jr z,smatch ; by definition, match!
sloop:      ld a,(de)
            inc de
            cpi
            jr nz,sdiff   ; char mismatch: exit
            jp pe,sloop   ; count not exhausted
smatch:             ; count exhausted & no mismatch found
            exx
            ld bc,0     ; bc=0000  (s1=s2)
            jr snext
sdiff:              ; mismatch!  undo last 'cpi' increment
            dec hl      ; point at mismatch char
            cp (hl)     ; set cy if char1 < char2
            sbc a,a     ; propagate cy thru A
            exx
            ld b,a      ; bc=FFFF if cy (s1<s2)
            or 1        ; bc=0001 if ncy (s1>s2)
            ld c,a
snext:      next

*INCLUDE camel80d.azm  ; CPU Dependencies
*INCLUDE camel80h.azm  ; High Level words
            END
```

Special Feature

Intermediate Users

Wire and Cable

Little Circuits

by Dave Baldwin

I don't think I have ever learned anything by getting it right (?) the first time. Sometimes the worst thing that can happen is for a circuit to work the first time. You have no idea of what's to come. Reminds me of a meter protection circuit I designed for a school lab. Mine worked perfectly. The copies that everyone else built from my design set off the alarms anytime anyone came near them. Oops. The SCR in mine required a large pulse to turn it on, but all the rest turned on with a teeny spike.

WIRE AND CABLES

What can go wrong with wires? Well, even in a digital circuit (maybe especially in a digital circuit) wires are ANALOG things. They have resistance and inductance and between any two conductors there is capacitance. And remember, printed circuit board traces are just flat wires that are glued down.

WIRE RESISTANCE

You just installed the new gizmo board with microprocessor controlled power relays, but the power supply is ten feet away. You grab some 22 gauge wire and connect it up. After powering it up, you find out that every time the relays come on, the system resets. You check everything, you have good power supply voltages and everything seems to check out but it keeps resetting.

Everytime the relay comes on, the current required by the gizmo board goes from 100 mA to 3 amps. At 100 mA, the 22 gauge wire drops 32 millivolts, but at 3 amps, it drops .96 volts! The 5 volt supply at the gizmo board becomes 4.04 volts and your power supply sensing reset circuit reboots the gizmo and shuts off the relay. Just because of a little wire.

It turns out that a ten foot long piece of 22 gauge wire has .16 ohms of resistance. And the two wires, +5 and ground, add up to .32 ohms. In this example, you would have to use 16 gauge wire or larger to keep the voltage drop at the gizmo below 0.4 volts.

Of course, you used the latest CMOS technology to build your system. If you were even more unlucky, you had an I/O board near the power supply with a gate output going over to the gizmo. Every time the relay came on, not only did the gizmo reset, but the input connected to the I/O board blew out. It turns out that with 22 gauge wire, not only does the +5 volt line drop .48 volts, but the ground at the gizmo is .48 volts more positive than the I/O board ground. The signal from the I/O board will turn on the input protection diodes at the CMOS gizmo input whether it is high or low when the relay comes on.

WIRE CAPACITANCE

Well, you fixed the power supply wiring and corrected the I/O signal problem. Now you need to put some pushbutton switches on the gizmo unit. You use some flat ribbon cable to connect the switch logic to the gizmo processor board. Now you find out that every time you push the button something different happens. What now?

You put the scope on the signals to the switch logic and it's being scanned every 10 mS, so that's 100 Hz. Frequency can't be a problem, can it? The problem here is the capacitance between conductors in the ribbon cable and the input impedance of the CMOS logic. Typical capacitance between conductors in standard 28 AWG ribbon is about 10pf per foot. This doesn't seem like much, but the maximum input capacitance for an HC gate is about 10pf. This makes an AC voltage divider that could put 50% of the signal on adjacent conductors if the frequency is high enough.

Typical transition time from high to low or low to high for high speed CMOS (HC/HCT) devices is about 20 to 25 ns with a 5 volt supply. This roughly equates to 20 MHz in frequency. 10pf at 20 MHz is about 800 ohms impedance. This could allow several milliamps of current to flow between adjacent conductors. This very simple analysis shows that there are some obvious problems here and we haven't even considered the series inductance of the cable that turns it into a tuned circuit. How do we minimize these problems so our circuits will work properly?

Crosstalk problems can be viewed as voltage divider problems. The amount of signal coupled from one wire to another is proportional to the impedances of the voltage divider that has been created by the circuit. One solution is to lower the impedance at the end of the wires. You can use an AC termination down around 200 ohms. It needs to be an AC termination (capacitor in series with the resistor) because HC gates and bus drivers can't supply enough current to drive 200 ohms to 5 volts. 800 ohms in series with 200 ohms lets only 20% of the signal appear on the adjacent wire. Another solution is to lower the effective frequency by increasing the transition time. A resistor in series at the source end both increases the impedance of the offending signal and slows the transition time by reducing the amount of current available to charge the capacitance between the wires. Sometimes, if the ribbon cable is short, just putting a pullup resistor at the end of the cable can be enough.

PCB TRACES

It's common to put capacitors on inputs for switches to prevent stray signals from triggering the inputs accidently. So I decided to put some 8.2 uF tantalum caps I got from a surplus store on the switch inputs for a custom display. I got everything put together and powered it up to test it. It seemed to work at first. I pushed a few more buttons and it all went haywire. The buttons were supposed to work in a particular sequence to light up the display, but I could never get past the second button. I checked all the logic, the power supplies, and the sequencer ROM. Over and over again. Put the scope on everything. Lots of head scratching. I put the scope on a ground point on the circuit for some reason and pushed one of the buttons. The scope triggered on something. Ground is ground, right? I shouldn't be seeing anything. Push the button again, same thing. I twiddle with the knobs and keep pushing the buttons, trying to see something. (Gotta get a digital storage scope someday.) Each time I push the button, I get a pulse of more than a volt between the two ground points. Turns out that 1) these were very good caps with low series resistance, and 2) I didn't put a current limiting resistor in series with the switches. I was shorting the caps directly to ground each time I pushed a switch, and putting several AMPs of current into the ground system each time. The discharge current from shorting the caps had caused a voltage drop of a couple of volts in the ground system on the circuit board. This collapsed the power supplies and upset or reset the logic on the board each time. As soon as I put resistors between the switches and the caps to limit the current to a few milliamps, everything worked perfectly.

AVOIDING PROBLEMS

Most of these problems can be avoided by keeping your connections short and paying attention to the amount of current flowing in your circuits. Small single board computers with their own local power supplies almost never have these problems. Designers of high speed desktop computers and workstations always have to cope with these problems because of the speeds that the systems operate at, the power that is required to get the speed, and the closeness of the traces on the circuit boards.

When you have to run connections over longer distances, you need to start taking voltage drops and signal frequencies into consideration. How fast a signal goes from one level to another is a prime consideration. This is the basic limit on RS-232 connections. RS-232 data rates, even at 100 Kbits per second, aren't the problem. The fact that RS-232 receivers are relatively high impedance inputs (3 to 7 kohms) and that RS-232 drivers can go from low to high and back at 30 volts per microsecond is the problem. With long cables, crosstalk between the conductors causes too many errors. Other serial standards such as RS-422 and RS-485 avoid problems by using low impedance terminations at the end of the lines. This (and other things) allows RS-422 to be used out to 4000 feet at 100 kbps where RS-232 is limited to less than 70 feet at 19.2 kbps.

CONTACT

You can reach me through DIBs BBS at (916) 722-5799, 1200 to 14.4 kb, 24 hours. There is a TCJ conference where you can leave messages. I've created a special logon that allows you to get directly to the TCJ conference and file area and skip the new user questionaire. Call (916) 722-5799 and use the following logon:

First name? <COMPUTER>
Last name? <JOURNAL>
Password? <SUBSCRIBER>

The TCJ download area has a ProComm script for logging on. All of the Little Circuits articles are available in the TCJ file area in PM4 format. I can make them available in other formats if anyone is interested. If you also want access to other areas, log on with your own name and password.

REFERENCES

BELDEN Master Catalog 885.

Regular Feature

Group Reviews

SUPPORT GROUPS FOR THE CLASSICS

ZFest in Europe

As I was finishing off this issue, Jay Sage called to check on things. While discussing *TCJ*'s status, Jay mentioned the June 18th ZFest in Germany. He indicated that many interested ZCPR users from all over Europe will be gathering to discussion Z-topics and just have fun. If you are interested contact Jay or Helmut for directions and a schedule of events.

April Fools Newsletter

I received the other day a newsletter or more accurately a small book with the tittle "A BIT MUCH." This was the April Fools special newsletter of The NOVA OSBORNE USERS GROUP. There meetings are on the fourth Thursday in Springfield, VA. To get your copy or copies of their newsletter, contact William E. Kost, 7007 Brocton Ct., Springfield, VA 22150.

Now this special has so many puns and humorous items, I was impressed and more. Take this "What do you call a computer scientist? It doesn't matter what you call him. He's too involved with the computer to come anyway." Take that and multiply it by 203 pages and you have an idea of what I got in the mail. So William, thanks and "how'd you do that?" Bill Kibler.

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.

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.

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

Users Groups and Project Reports:
JW Weaver, Drawer 180, Volcano, CA 95689, BBS: (916)427-9038.

Regular Contributors:
Dave Baldwin, Voice/FAX (916)722-3877, or DIBs BBS (916) 722-5799 (use "computer", "journal", pswd "subscriber" as log on).

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, Germany, "Virtual" ZNODE #51, or CompuServe 100024,1545.

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 Undley 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, 2215 American Drive, Roseville, CA 95747, (916) 773-4822. 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.

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, 1480 Terrell Mill Rd. #870, 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.

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:
- Language Development: Automatic Generation of Parsers for Interactive Systems
- Designing Operating Systems: A ROM based OS for the Z81
- Advanced CP/M: Boosting Performance
- Systematic Elimination of MS-DOS Files: Part 1, Deleting Root Directories & an In-Depth Look at the FCB
- WordStar 4.0 on Generic MS-DOS Systems: Patching for ASCII Terminal Based Systems
- K-OS ONE and the SAGE: System Layout and Hardware Configuration
- The ZCPR3 Corner: NZCOM and ZCPR34

Issue Number 33:
- Data File Conversion: Writing a Filter to Convert Foreign File Formats
- Advanced CP/M: ZCPR3PLUS & How to Write Self Relocating Code
- DataBase: The First in a Series on Data Bases and Information Processing
- SCSI for the S-100 Bus: Another Example of SCSI's Versatility
- A Mouse on any Hardware: Implementing the Mouse on a Z80 System
- Systematic Elimination of MS-DOS Files: Part 2, Subdirectories & Extended DOS Services
- ZCPR3 Corner: ARUNZ Shells & Patching WordStar 4.0

Issue Number 34:
- Developing a File Encryption System.
- Database: A continuation of the data base primer series.
- A Simple Multitasking Executive: Designing an embedded controller multitasking executive.
- ZCPR3: Relocatable code, PRL files, ZCPR34, and Type 4 programs.
- New Microcontrollers Have Smarts: Chips with BASIC or Forth in ROM are easy to program.
- Advanced CP/M: Operating system extensions to BDOS and BIOS, RSXs for CP/M 2.2.
- Macintosh Data File Conversion in Turbo Pascal.

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.
- The Computer Corner.

The Computer Journal Back Issues

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

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

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

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

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

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

SPECIAL DISCOUNT

15% on cost of Back Issues when buying from 1 to Current Issue or all four volumes.
10% on cost of Back Issues when buying 10 or more issues.

	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

PLC & Forth

The Computer Corner

By Bill Kibler

Well here we are again talking about PLC's and control in general. Last time I explained how a PLC works in very fundamental terms. Learning how one works or is programmed in specific varies from vendor to vendor. I have seen large changes in each vendors implementation and thus if you really get involved I can only recommend you attend a factory schools.

If you want to write your own "sort of" PLC program, I have the code samples (or ideas) that I presented to our local Forth meeting. The basic idea is to create a table of BITS that represent inputs/outputs/relays that form the main function of the PLC. You in essence turn these bits either "on" or "off" to represent functions to be performed. Looping through this table of BITS looking for changes and making those changes, is actually all the program does.

My code sample just shows how to make that table, handle the bits, and create the run time program that would relate bits to devices. This is really just an example of how you might do it. To really do a PLC in Forth or any language requires many more lines of code and structure. The most difficult decision in the design process is handling I/O. Now updating inputs is rather easy, just copy the input data to memory locations, possibly as 8 bits at a time. Writing or updating the output is another question. Do you just blindly write the output or test and only write if different.

The reason writing becomes a problem depends on how you talk to I/O. We have a proprietary system talking to a PLC. It tells us when things change thus making updating easy. However, the PLC bit table can get changed and there is no PLC mechanism to automatically signal that output bits need updating with our proprietary ports. What has to be done is a comparison of what was last sent over the serial port and what is the bit table's current state. If they differ, the location is flagged for sending.

So what you will notice missing in my code sample is the input/output part of the program. I leave that for you to consider and experiment with. Of course you could consider buying a great already to run industrial plant control software package.

Forth Inc.'s Express

At our last Forth meeting, members Gary Sprung and Bob Nash (both work at Sacramento Municipal Utility District = SMUD) brought a demo version of Forth Inc.'s Express. SMUD is using this to control their Solar Cell system. Express is an object oriented shell (and more) that sits on top of their polyFORTH programming environment. (For more on polyFORTH see Charles Shattuck's mini-article in Reader-to-Reader.)

The demo that Gary gave was great and showed why they went to express. It works as a true real time multitasker. It has a very powerful demo or simulation mode that helps speed up development time. Gary couldn't say enough good things about the product. After watching the demo, it was clearly apparent that PLC ladder programs may be a thing of the past when compared to express.

How so you ask. Take controlling doors. Each door has a location, a name, a specific type of mechanism, alarms, groups and many other considerations that must be checked before opening. Using Express all these items are placed inside the "object" called "Door". Since different door opening mechanisms have different procedures for powering the opener, our object "Door" then has different functions it will perform based on that door type.

This is very powerful and easy to use once set up. When all the tables/items have been established, then adding a new door is just copying the old record with the minor changes for a new I/O address or door name. Easy!

NEXT?

Well that is it for this time. Keep hacking and sending those letters.

```
\ PLC    words to create a ladder logic system

HEX
CREATE IRTBL  100 ALLOT
                \ set aside a test area for working data
                \ address is position within IR table
IRTBL 100 ERASE        \ fills table with all zeros...
CREATE BATBL
 \ bit mask table for TESTing(AND)  and SETing( OR) data
01 c,  02 c,  04  c,   08  c,
10 c,  20 c,  40  c,   80  c,

CREATE BOTBL
      \ bit mask table for RESETing a bit by ANDing data
FE c,  FD c,  FB  c,   F7  c,
EF c,  DF c,  BF  c,   7F  c,

: BITTST ( bitval  tbladdrs - flag )
IRTBL + c@ AND 0<> ;

: GETBIT ( bit# tbladdrs -- bitval tbladdrs )
SWAP BATBL + c@  SWAP ;

: LD  ( bit# tbladdrs -- flag )  GETBIT  BITTST  ;
: LDNOT ( bit# tbladdrs -- flag ) GETBIT BITTST 0= ;
: OUT ( flag bit# tbladdrs -- )
GETBIT IRTBL +
               \ gets AND table value and real address
ROT 0<> IF CSET        \ sets bit by ANDing
ELSE CRESET            \ clears bit by NOT ANDing
THEN ;

: LDAND ( flag bit# tbladdrs -- flag ) LD AND ;
: LDOR  ( flag bit# tbladdrs -- flag ) LD OR ;

\  examples:
\  1 10 LD  2 10 OUT  = if bit 1 of 10 is notzero
\               then bit 2 of 10 = 1
\  1 10 LDNOT 2 10 out = if bit 1 is zero make bit 2 = 1
\  2 10 LD  1 12 LDAND  3 10 OUT = make 3 = 1
\                if both 2/10 and 1/12 <>0
\  2 10 LD 3 10 LDOR 4 10 OUT = make 4/10 = 1
\                 if either 2 or 3 is <>0
```

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.

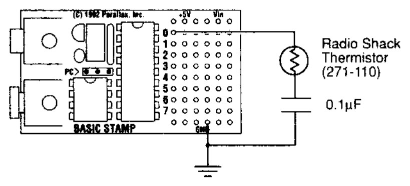
PnP BLUE or **PnP WET**

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

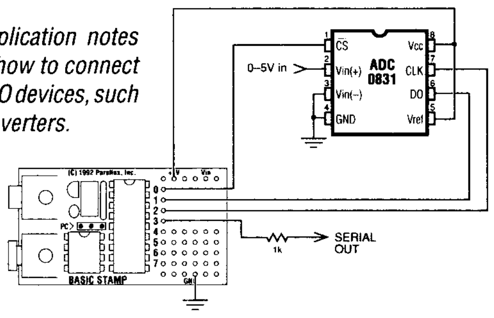
BASIC Stamp

$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