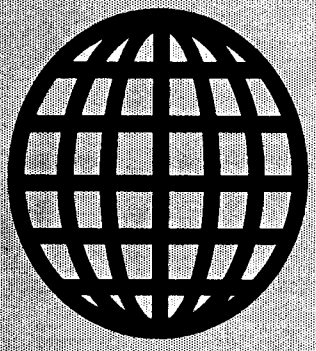
Providing Support Around The World



The Computer Journal

Issue Number 72 March/April 1995 US$4.00

Beginning PLD

Small System Support

Playing With Micros

8048 Emulator Part II

Small Tools

Real Computing

Support Groups

Dr. S-100

Moving Forth part 7.5

Centerfold - Rockwell R65F11

The Computer Corner

ISSN # 0748-9331

TCJ - For Having Fun With Any Computer!

COMPUTER HISTORY COLLECTION FOR SALE:

Books from 1950–1993

Magazines:
- **Byte: 1976–1989 149 issues**
- **Kilobaud: 1977–1980**
- **Fred Gruenberger's Computing News: 1953–1958**
- **Berkeley's Computers & Automation: 1952–1968**

ACM Publications: **Journal: 1956–1975 60 issues**
Communications: 1958–1980 236 issues
Computing Reviews: 1962–1980 117 issues

Semiconductor Company Catalogs 1967–1981

UNIVAC and IBM Publications

Computer Hardware including: WW II Sperry T-1-A analog bombsight computer made by GM's AC Spark Plug division
1973 National Radio Institute Model 832 Digital Computer
1977 Intel Intelec MDS 888 serial # CP 192
1977 Intel Prompt 48 serial # CH 196
IMSAI PCS–80/15 serial # 1369
SYM–1 as new in box with manuals

Toys including: 1959 Brainiac & Calculo Computer kits by Science Material Center

Please send long self addressed envelope with 55 cents postage to: Randy Liebermann, 2874 South Abingdon St., Apt. A1, Arlington, VA 22206–1363 Tel: 703–824–9733

Peripheral Technology Specials

486SLC 33MHZ Motherboard w/ CPU	**$119.00**
486SLC/66MHZ IBM,VESA,CPU,Math	**$219.00**
IBM board - Made in USA - 3YR warranty	
PT68K4/68000/16MHZ /w 1MB	**$249.00**
CDS/68020/25MHZ CPU	**$399.00**
OS9/68000 Includes C Compiler	**$299.00**
420MB Connor IDE Drive	**$215.00**
540MB Connor IDE Drive	**$309.00**
IDE/Floppy/Serial/Parallel	**$24.95**
1.44MB TEAC Floppy	**$49.95**
Panasonic Dual Speed CD ROM	**$159.00**
VGA Card ET4000-1MB,1280x1024	**$99.00**
VGA Monitor WEN .28mm 1024x768	**$229.00**

Free Catalog on Request

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

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

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

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

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

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

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

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

BROAD RANGE OF SUPPORT
- Currently we support the following microprocessor families (with more in development):

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

- All products require an IBM PC or compatible.

So What Are You Waiting For? Call us:

PseudoCorp
Professional Development Products Group
921 Country Club Road, Suite 200
Eugene, OR 97401
(503) 683-9173 FAX: (503) 683-9186 BBS: (503) 683-9076

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
Ron Mitchell
Dave Baldwin
Frank Sergeant
JW Weaver
Richard Rodman
Jay Sage
Tilmann Reh

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

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

TCJ The Computer Journal
Issue Number 72 March/April 1995

Editor's Comments 2

Reader to Reader 3

Dr. S-100 10
Compupro 8080/8086.
By Herb R. Johnson.

Small System Support 14
C and assembly language tutorial.
By Ronald W. Anderson.

Beginning PLD 20
The good and bad of using PLD's.
By Claude Palm.

Center Fold 25
Rockwell R65F11 Single Board Forth Computer.

Support Groups for the Classics 29
ZED-FEST at Trenton.

Real Computing 32
Programming languages.
By Rick Rodman.

Small Tools 34
Forth based tools for 68HC11.
By Calvin McCarthy.

Playing With Micros 36
Reviewing 5 micros to learn with.
By Bill Kibler.

8048 Emulator 40
A home built emulator.
By J. G. Owens.

Moving Forth 44
Part 7.5: 8051 Camel Forth.
By Brad Rodriguez.

The Computer Corner 50
By Bill Kibler.

EDITOR'S COMMENTS

Welcome to number 72 and a special on embedded systems. We bring you a number of articles about embedded devices and systems. Although not totally devoted to controllers, we have several articles that may help you understand what is happening in this field.

Starting this issue is the ever popular Reader to Reader section. We have several good letters and discussions that are a must to read.

Dr. S-100, Herb Johnson, steps up next with his letters bag and talk about a Compupro 8/16 system. He is followed by Ron Anderson and more straight talk about programming. Ron adds a new section on C programming so you can see what that is all about.

When we ran Claude Palm's letter about his IDE interface, some interest was sparked about doing PLD designs. Claude had indicated a willingness to explain his experiences and thus an article doing just that is called "Beginning PLD".

This issue's centerfold is the Rockwell R68F11 development board. We think the device is no longer available, however it does show a good simple design, complete with floppy controller.

Please read the section on Groups as it announces the 1995 Trenton Computer Fair and Zed-Fest. Jay Sage sent me the information, and I was going to be there, but due to family problems will have to pass it up. Rick Rodman zips in words on some 16 bit programming languages after the groups directory.

Starting our slate of special articles is a discussion of the group of tools for programming the New Micro's 68HC11 in Forth. The review of tools is by Calvin McCarthy of Canada, where he shows you some of the problems and solutions that arise when trying to interface to Embedded controllers.

At this point I finally get to run my Embedded Review article that helps you understand what you get for your money. I decided to do this article after my last embedded trainer purchase turned out to be rather different than I expected. I review five system including this issue's centerfold.

Part three of J. G. Owens Monitor/Emulator for the 8048 provides us with the hardware portion of the design. I must say I have never had a schematic quite like his before. It is an IBM graphic font file that runs about six feet long. It is all there including some of his comments.

Speaking of parts, we finish Brad Rodriguez's code listing for his 8051 CamelForth. Brad and several other authors are taking some time off to finish school work and that leaves only my last words of wisdom in the Computer Corner.

I conclude by pointing out a vendor of old software that is still selling and some PLC fodder for thought. For those wanting more PC/XT articles I make a request for help in my corner. Much like the problem I faced with embedded articles, all my PC/XT people are too busy as well. So if you have some good articles that explain and bridge the gap between old and new send them in.

Business News.

Well *TCJ* is doing about the same as the last two years. By that I means we have some readers, but not enough. I have been trying to get the word out, but earning my own living has been taking it's toll on the magazine and me. I am not giving up on *TCJ* nor will I let it be sold off to become a PC only rag. I am however looking for someone or group to take up the slack and or take it over.

I currently spend 80% of my time doing paper work. That is far too much in relationship to what should be happening on the editorial side. I went from a full time job to one that I could work 80%. That gave me one day a week to work on the magazine and a 30% cut in pay. The hope was that *TCJ* might make up the slack. One day a week is still not enough and has not brought in more readers or more money. It has allowed me to do some catching up as you have seen in a few old articles that finally found the top of my desk.

I have been writing for *TCJ* over ten years. Several others have burned out and faded away or just slipped far into the background. I would rather produce my corner and do the occasional article. I seldom have time to do articles, especially those needing lots of research.

Two ways appear as options. The preferred way is for all readers to get a friend to subscribe. Doubling our readership would give me a little cash from this work and push *TCJ* enough into the black that I might be able to hire some occasional help. My other option is finding a new publisher or buyer for *TCJ*. I have grave concerns for this option as once out of my hands it could be killed far too easily. These ideas and options are not new, I have been pondering and looking for help for some time. The dragging on and realization that I am starting to exceed my limits (and my families) is just taking it's toll. I need some feedback and soul searching on your part as supporters of *TCJ* to find an answer.

The Computer Journal is the only regular magazine supporting all the old systems, so we need some fresh ideas and help in getting the word out and thus more readers hooked on us. I look forward to hearing from you, soon.

Bill Kibler.

READER to READER

Letters to the Editor

All Readers

MINI Articles

Dear Bill,

Enclosed is the material for my ad in the *Market place* section of *The Computer JournaL*. I have included both 1x and 2x scale artwork, two copies each. I have also included two styles, one with a black background at the top and bottom and one with an all white background. Please feel free to select the format that will work best in your publication. I have enclosed a check for $50.00 as payment for the May/June and July/August 1995 issues.

I enjoyed talking with you on Monday and hearing your thoughts on my business ideas. My reasons for starting my own business are many. Having just joined the ever growing number of unemployed defense workers and not much chance of new employment; I felt that my 28 years in electronics was still worth something. Driven by the need to eat regular, I decided that my best bet was to start my own business. I have studied the competition very carefully and found a wide variety of systems and prices. From my own point of view most, if not all, these systems all have something lacking. For example; one vendor sells the basic controller board, but without BASIC or a debug monitor that costs extra. It seems to me that one is useless without the other. If you want some sort of I/ beyond that of the processor then that costs extra also. The well-rounded systems that include everything are usually out of the price range of the student or hobbyist. What I felt was needed was versatile, low cost, systems that come with everything you need.

My first product is an 80C32 based system on a small 4 by 6 inch double sided PC board. The board comes with 8 channels of 8-bit A/D, an 8 or 12-bit D/A, 24-bits of parallel I/O, real time clock, and a parallel printer port. The board also includes 32K of EPROM, with BASIC-52, and 32K of RAM. The RAM and real time clock are both battery-backed. While the 8051/52 family of parts is not as advanced as some of the newer Dallas Semiconductor or Signetics parts, they are well established and low cost. All of the components used in the system are readily available from a variety of sources including JDR Microdevices, Jameco Electronics and Digi-Key. The board will be available fully assembled or as a kit, yes I said KIT. Part of my motivation to offer the board in kit form grows from my own experience in kit building. It's been almost 28 years since I built my first kit, a Heathkit Oscilloscope, but I still remember the knowledge and experience that I gained. Another reason for selling in kit form stems from an editorial that appeared in the November 1992 issue of Elektor Electronics USA (now out of publication in the US).

In that editorial the editor stated that,

"Americans are rapidly losing their ability to make things with their hands. The number of helpless, hapless individuals grows steadily with each new generation."

I don't think that's completely true, but if it is then were all in trouble. At any rate, there's a lot to be said for putting something together with your hands and watching it work for the first time. I said earlier that hardware was useless without software. All of the software used in this project is public domain, including the BASIC-52 interpreter, and can be obtained from the Phillips/Signetics support BBS. Although modified to work more efficiently with my hardware, this BASIC is included at no charge (that's what public domain means). I have added 16 new commands to the BASIC plus a simple debug monitor, the cost of which is included in the price of the system. The only commercial software package that I used is an integrated development environment program called "ARMADILLO+™". ARMADILLO, developed by *Life Force Technology,* is a slick little program that integrates all of the utilities needed in microcontroller development into a single package. At the heart of the package is a terminal emulator with full upload, download and screen capture capabilities. I am in the process of making arrangements with the folks at *Life Force Technology* to include a reduced function version with my system. When I say reduced function, I don't mean cripple ware. The program will look and work just like the full commercial version except that it will only work with one system, my system. The exact details of this agreement have yet to be worked out, but when finalized I will be able to do what I set out to do; provide a complete system for a reasonable price. Oh, in case your wondering, ARMADILLO stands for Asynchronous Responsive Multi-Assembler Development Integrated Link to Logical Operation. The Phillips/Signetics support BBS phone number is 1-800-451-6644, 2400-8-n-1. The address of *Life Force Technology* is 5477 Rutledge Rd., Virginia Beach, VA 23464, Phone (804) 479-0973. Well, if all this works I have a couple of designs based on the Motorola MC68HC11 in the works to expand my product line.

Sincerely, Donald W. (Don) Coates
D. C. Micros

1843 Sumner Ct.
Las Cruces, NM 88001
Ph. (505) 524-4029

Thanks for the ad and this letter Donald. Normally I do some sort of introduction for new advertisers, but your letter explains it all. I am vary interested in knowing how well you do and I think you will do well, since your product and direction seem to be well thought out. I am interested in trying to get a small article from you on the problems or steps needed to use the BASIC-51. I may be doing the same with it later this year.

I too doubt that people in the USA are losing their ability to build with their hands, but unfortunately managers and politicians do. TCJ is built on the idea that people still do want to build things and your kit idea fits right in with our reader's practices. I usually tell people to just buy some old system to play with, but kit based projects, generally are a bit more positive (if the supplier has enough how-to in the manuals.)

This issue has turned out to be a special on embedded systems, so check out the other articles and reviews of systems, you fit right in with them. Thanks for joining TCJ, Donald. BDK.

Dear Mr. Kibler,

Enclosed is my check for another year's subscription. Thanks for an excellent magazine! I have been a reader of this magazine for a year, and every issue has been a wealth of information. I find that it takes me several days to digest the information in each issue. Few other magazines make me think so much.

One thing I would like to comment on is the readership of this magazine. I am a twenty-one year old college student. From what I have read of the letters and articles, most of the readers are considerable older than I. I suppose this is only natural, given the computers that are discussed here. Consider this: I was seven years old when the IBM PC was introduced. I suppose this makes me a child of the computer age. When it comes to computers, I have barely scratched the surface.

This then, is why I enjoy this magazine so much. It has given me numerous ideas for small projects and experiments. Although the only computer I own that falls under the scope of this magazine is an old Epson 8088, I am eagerly searching for more machines to add to my collection. In the past, I have seen older computers at garage sales and auctions, but I have been hesitant to buy them, not knowing where to turn for assistance. This magazine has given me that, and more.

One thing I would like to see in *The Computer Journal* is coverage of single board computers. I have lately been examining several of these based on the Motorola 6811. In one of my classes (I'm a computer science major), we dealt extensively with programming the 6811 EVB. For anyone not familiar with this, it is a single board computer produced by Motorola for evaluating the 6811. You connect it to a PC for programming and debugging. The only problem with these boards, though, is their cost, about $125. A much more attractive design which I am looking at right now is the Miniboard, developed at MIT. The Miniboard is again based on the 6811, and with a few other components, it fits on a 2" by 2" board. With that, you get something like 256 bytes of RAM and 1K of EEPROM, depending on the exact chip you use. You program it through a serial link with a PC, and it can communicate with other Miniboards, and control 4 switches or two motors. Total parts cost is about $50, including the CPU. The good part is that Motorola will give you samples of any of their chips for free if you are a college student. So, I already have the CPU, and am in the process of procuring the rest of the parts. I should have a very interesting computer when I am through with it.

Anyway, I feel that these computers should definitely be covered more here. They're inexpensive, easy to use, and give you a real understanding of assembly language and control applications (at least in my experience). So, I hope to see more. Do you feel that this is an appropriate area for the magazine? If it is, I would certainly be interested in writing about it.

Well, those are just a few of my thoughts. Again, thank you for an excellent magazine. I can see now why back issues are such a major source of revenue for you. I know I will be ordering some in the future, and reading all the issues to come.

Sincerely, Philip
hollenb@selway.umt.edu

Philip, you should find this issue a super bonus. It seems the embedded articles all got done and are starting with this issue. Actually we have always done small systems like the 6811, it just been some time since I or any of the regular writers has had anytime to put out an article.

You might look at what Don Coats (DC Micro) has for you. He is planning a kit version of the 68HC11 in the very near future that might be in your price range. Also some vendors sell their design as bare boards, although often at too much money for what they give you. So check them out and the articles in this issue, just for you (well not really just for you).

As to collecting the older machines, just start doing it before they get too expensive for you. The old one's are just great to really learn about full systems. The little guys are great for understanding the CPU design, but at some point you have to understand the whole system. I find the older, simpler CP/M (or similar machines) the best for learning how the hardware and software work together (DOS, BIOS, Video, I/O, etc..). Thanks for the letter and start collecting, now. Bill.

Hello Bill,

Thanks for sending the sample issue of *The Computer Journal.* I got a pointer to your magazine from Don Walterman, who you published a letter from in this issue.

Like Don, I'am an avid QL user. I've been with Sinclair computers since 1981. Let me tell you a little bit about the newsletter that I put out. I've taken the

following from a standard flyer/file I put out.

The *QL Hacker's Journal (QHJ)* is a newsletter published as a service to the Sinclair QL community. The *QHJ* is aimed at QL programmers. The *QHJ* is free to all QL programmers interested, and can be freely distributed to all. The *QHJ* is distributed in hard copy and as a text file via electronic mail (Internet and Compuserve). All back issues are available on disk, via e-mail, or Anon-FTP from garbo.uwasa.fi.

I've enclosed a copy of the current issue, plus an index of all past issues and articles.

Thanks for the offer of doing a swap of publications. Since my newsletter is smaller and published less often, I'm sure I'd be the one getting the better deal. But, I must decline the offer. I would not feel right getting such a good deal. You are welcome to be on my mailing list, either hard copy or electronic, or both. Since the *QHJ* is free, I'm extending you the same offer I give to all interested parties.

Of the listed Other Publications, I read four of them. I get both computer history magazines and both Sinclair magazines. I've written stuff for *"Historically Brewed", IQLR, and UPDATE.*

Besides being a QL enthusiast, I also collect computers. I am up to about 53. I like to specialize in the home computers of the late 70's and 80's. I don't collect too many of the S-100 or CP/M systems. I like the non-standard ones. I have almost the complete Sinclair, TRS-80, and Atari line. I have Commodores, TI-99, Spectravideo, Mattel Aquarius, etc. Most just sit in the closet, but I do occasionally bring them out and play with them.

I do have a few CP/M machines (Osborne, SuperBrain, Advantage). I know CP/M is still functional and can still get the job done. I've done a few term papers using WordStar.

Well, back to *The Computer Journal*, I don't know if I'm ready to subscribe just yet. Most of the articles are more hardware related and I'm a software guy. The most I do in hardware is built my own serial cables. I've tried learning about hardware, but never got past the basics of electronic.

But, like Don, I would like to offer my services if you are looking for Sinclair related information. If you ever get a QL I can provide you with lots of freeware. I specialize in collecting freeware compilers and interpreters for the QL.

From the computer collecting side, I've a few books dealing with computer history (PC's mostly) and would be more than willing to look something up for you.

I also keep tabs on various emulators for classic computers. The QL has a ZX81 and Spectrum emulator for it. There are emulators for the ZX81, Spectrum, CoCo, Dragon, C64, and others for MS-DOS. I've picked up a ZX81, Spectrum, and QL emulator for the Atari ST. If you are interested in any of these, let me know, I'm on the Internet and can download them fairly easy.

Keep up the good work with *TCJ*. Please let me know if you want to be added to the *QHJ* mailing list. I would enjoy sending it to you.

Happy Hacking,
Timothy Swenson
5615 Botkins Rd.
Huber Heights, OH 45424
(513) 233-2178
swensotc@ss2.sews.wpafb.af.mil
tswenson@dgis.dtic.dla.mil

QL Hacker's Journal

#1 January 1991
Structured SuperBasic Ratcliff/Obershelp Pattern Matching The Quebec Link
Pursuit of a Public Domain C Compiler Minix on the QL

#2 February 1991
Find-c
diskinfo
C beautifier

#3 April 1991
Herb Schaafls Small-C Programs
File Comparison
Real Windows for SuperBasic
C Compiler Comparison

#4 July 1991
Rand c
Cellular Automata
Iterated Function Systems

#5 August 1991
News
The Dutch Connection
QHJ Print Formatter
QROFF Postscript Formatter
2D arrays in Small-C

#6 November 1991
Italian Software
Dutch Connection II
RPN Calculator
Substring Searching in C
Levenstein Distance
QDOS Rights
Compiler Benchmarks

#7 January 1992
Core Wars
QLPatch
The German Connection
New QL

#8 March 1992
ASCII Dump
Check Bits for ASCII Files
Ansi C to K&R C
Strip-c

#9 June 1992
New Public Domain/Freeware QL Software
Software Engineering and OOPS on the QL?
Random Dots Stereograms
Infix to Postfix
Fletcher's Checksum

#10 September 1992
Programmer's Bookshelf
Maus.sys.ql
PGM and PBM on the QL

#11 November 1992
C68 v 3.03
Disk Eraser
LF/CR to LF in Editors
Random ASCII Stereograms
LZW Compression
Token Reconstruction

#12 January 1993
QL to Z88 Data Transfers
MacPaint File Printing
Maze Solution with CA
QHJ Index

#13 April 1993
Text Editors
QL Languages
Proglog Interpreter
EFORTH Interpreter
Programmer's Bookshelf Revisited Recent Ports
Byte Input in SuperBasic
FORTH Programming

#14 July 1993
More on Text Editors
Ten Commandments of C Programmers
Internet Conciseness Contest
Another Look at Mazes

#15 October 1993
Hex Movement Library
Base Conversion
Computer Language Humor
Internet Consiceness Programming Contest How Do They Do That - Editors,.
Stochastic Indexing

#16 January 1994
Quill Reader
Dice Percentage
QL Anon-FTP Server
Prolog Interpreter: A Second Look QHJ Reader Survey

#17 April 1994
Readership Survey
C Portability
Permutations
Prime Number

#18 August 1994
Complex Ascii Rotation
Approximate String Matching
Hello, World
Natural Language

#19 November 1994
Displaying TI Graphics Files Displaying QL Screens in MS-DOS Recent Freeware Releases
Big Numbers
Dynamic Windows - Another Approach Soundex

Thanks Tim for the sample and list, and yes send me your work by internet or CompuServe. I am interested to see the eForth and MINIX articles, but then the whole list of back issues looks pretty good too. Yes I do need an article on emulators, both specific and in general. I was going to do one, but too little time and too much to learn. Jay Sage is working on the CP/M emulators, but there are plenty of Sinclair emulators to learn about as well.

I have a copy of the latest IQLR and it is rather impressive (PO BOX 3991, Newport, RI 02840-0987, USA or IQLR 23 Ben Culey Dr., Thetford, Norfolk IP24 1QJ, Great Britain $20US,£25 per year). A very good publication, I thought done in England, but I see that was only because of all the British advertisers. Two of those ads got me going, a IDE interface for the QL (mostly because we are doing one for Z80's) and the QL for a PC/AT bus. Would like to know more about both of these. The problem is the cost of money exchange (forgetting no time to play anymore with my collection of - well enough machines) and too little knowledge of QDOS(?) and other Sinclair insides. I am interested in anything based on 68K that is not Apple (since Apple is a big world of it's own and not really hacker oriented).

Actually one ad that got me interested was the 5 QL's for $35 each, from a person in the USA (615-483-4153). Well I better stop thinking of QL's before I get myself and all the readers too curious to read on. Thanks Tim and how about a few articles? If you need any help on SuperBrains, I got some schematics of the QD. Bill.

Dear Bill,

I read about your interest in repairing the Timex Sinclair keyboard in the 93 Sept/Oct issue of *The Computer Journal* & thought you would like my solution to the problems I experienced.

My first attempt was to re-cut the end going into the pc board & then inserting a foam spacer so the end would not get bent & break off. This worked for 5 years & then it broke again.

Not having enough flex cable left to work with, I tried soldering wires only to vaporize the metal coating off, making my problems worse. Then I came up with what I thought was a great idea. First cut and shape the end coming out of the keyboard then unsolder the connectors from the pc board then solder some ribbon cable to them about 8-10 inches. Install the connectors on to the flex from the keyboard & use hot melt glue to secure the connectors to the top case so that the flex cable has very little to bend as any bending will eventually break again. Then solder the ribbon cable back to the pc board & you are done. This makes it much easier to open up the computer to work on & eliminated the other problem of trying to reseat the flex from the keyboard back to the pc board.

I used to experience white outs while pressing the keyboard which was extremely frustrating having to start all over again. I saw a lot of adds for "cures" from overheated regulators to poor ram connectors etc. I enlarged my heat sink with no improvement, replaced my CPU with a cmos version & applied a dip chip heat sink to the custom chip & still had lots of trouble.

I removed the CPU chip and looked very carefully & found a metal shaving shorting 2 pins intermittently. I removed it & it has worked fine ever since.

I had tape loading troubles & found out what is required of the cassette recorder which I have never seen in print. The recorder needs to have an output transformerless power amplifier & use at least a 6 volt power source preferably 7.5 volts.

If this is too difficult to obtain a circuit can be bought or built to boost the signal to the 5 volt peak to peak signal required for proper loading to occur. I modified a circuit that appeared in Popular Electronics many years ago. Their circuit used an opamp with a bi-polar 9 volt external power supply which I modified to a single ended power supply to operate directly from the 5 volt power inside the computer. I also had to change the opamp from the one they chose to a low voltage version to operate properly, it worked great.

There is a company that still sells software & hardware for the Timex Sinclair 1000 & they have the fast load software that works great & allows you to get a directory of your tape contents.

My address is Jim Hathaway II
3057 Scotland Dr.
Antelope, Ca. 95843

Your friend
Jim Hathaway II

Thanks Jim for that information. How about that new circuit, as I am sure there are a few readers interested in how to make their tape work better. I am aware of that keyboard problem, I rewired a keyboard that used ribbon cable years ago. I am really NOT a fan of the ribbon cable systems. Many of them use painted on conductors which comes off if removed from the socket, one shot

items! I am glad you found that floating piece of material, strange but that is often more of a problem than people think, good going! Thanks for the hands-on report. BDK.

To: B.KIBLER
Sub: 1802s, etc

I am a new subscriber, however I ordered some back issues, and in issue #63 JW Weaver asked in his column for information on Quest Electronics/Super ELF and the RCA 1802. I have the Super ELF construction manual, a series of construction articles from Popular Electronics on building a small SBC using the 1802, and some misc. software articles. Unfortunately I don't have the "Super Expansion Board" that Quest sold for the Super ELF...if you ever get a schematic I think it would make an OUTSTANDING centerfold for your magazine. Another company that was into 1802's was Netronics R&D in New Milford, CT. Mountain View Press has an assembly listing for 1802 fig-Forth available, and Newark Electronics still catalogs the chips, including a multiply/divide unit.

Also I have a problem that hopefully someone can help me with. I have a very nice keyboard made by Cortron (model #80-350181, US patent #3,035,253, serial #462811) which I would like to hook up to my Super ELF. However, I have no documentation and the decoder chip is missing. I —think— it used an 8748 which would be easy to get if I had a copy of the on chip program...so I'm hoping someone is familiar with this keyboard.

Anyway, nice magazine:) Especially the Scroungemaster II. I hope to be building one soon, but mine will be a single processor system (I think). Hey...maybe you could talk Brad Rodriguez into designing a new and improved "Super Expander Board" for 1802 systems, including 4 cascaded 1855 M/DUs and Forth.

Sincerely, Ken Deboy
Glockr@delphi.com

Interesting request Ken, and thanks for the letter. Well it has been some time since we have had an 1802 article. I think a long past contest winner was using an 1802 CPU. Guess I'll have to hunt it up in the back issues.

Till then, how about you writing on it? They say that you learn more by trying to explain something than by watching or reading, how about it? As to keyboards, well usually there is always another one a bit closer to your needs if you just look around for it. Although programming your own 8748 would sure teach you some good skills and make a good article. Oh well, hope you hear from someone, and thanks. Bill.

To: B.KIBLER
Sub: *TCJ*

Hello,

Thank you very much for the sample issue #70 of *TCJ*. Is it possible to subscribe (and order back issue #69) by emailing you my credit card number or would it be better to mail you a check? What a great mag!

I had a long email conversation with Herb Johnson a while back during which I managed to slip in a word about the PDP-11 simulator I've been working on for the last year or so (I never miss an opportunity to plug pet projects), and he suggested that I submit an article on it to *TCJ*. I objected on the grounds that it definitely fits into the "pet project" category and that no one who doesn't share my twisted values would want to hear about it but he said yep, sounds like the right kind of thing for *TCJ*. So I'd like to ask, would you be at all interested in an article about a PDP-11 simulator written in assembly language for the IBM PC? This would be a descriptive article, I'm keeping the source code to myself and besides there are 25K lines of it, a bit much to list. Anyway if you think you might be interested, please let me know if you have any guidelines for writing articles (I've never written one) and what slant you think would be best. I figure no one (in the 8-bit world anyway) wants to hear about the particulars of PDP-11 devices or memory mapping, but it might be useful to cover each subsystem (instruction interpretation, operand fetching, memory mapping, interrupts, delayed I/O events, DMA) in a general sort of way to show how they can fit together and what tricks you need to ensure compatibility w/o sacrificing speed. The PDP-11 differs in all the particulars from typical micros that others might want to write simulators for, but basic things like memory mapped I/O and delayed I/O events and the fact that individual I/O devices have to have their own state machines, should be useful to everyone writing a system simulator.

Re issue #70: it appears that you worked this out but just to confirm it, WRT the discussion of replacing 8" drives on p. 7, yes it's the 5.25" AT-style drives which can generally replace 8" drives. 8" drives turn at 360 RPM and use a data rate of 500kHz (MFM, 250kHz FM), and have 77 cylinders; 5.25" drives in 1.2MB mode (the default) have the same parameters except that they have 80 cylinders, so they're a good match. I use them in my simulator to simulate 8" disks and it works great, and if an 8" drive were substituted (and arrangements made for TG43 etc.) then the media ought to interchange too (I'll know for sure as soon as I get an 8" drive for my CompatiCard IV) with the same programming. Using an AT 1.2MB drive to replace one of the old so-called "quad density" drives (i.e. double density but 96tpi instead of 48tpi) is a little trickier since minifloppies normally use 250kHz (125kHz FM) data at 300 RPM. Some 1.2MB drives (particularly older ones) have an "/RPM" line which slows the motor down to 300RPM when grounded, so if you modify your drive (or controller, or cable) to ground this line then it can replace a normal "QD" drive (such as the Tandon TM100-4). Newer drives may not honor the /RPM line since it's more common in PCs these days to speed the controller up from 250kHz to 300kHz instead of slowing it down from 360RPM to 300RPM, it's cheaper and runs faster anyway (and the controllers shield the difference from software).

1.44MB drives use a 500kHz data rate and turn at 300RPM, which makes them look like an 8" drive with 20% more bits per track besides having 3 extra cylin-

ders like the 1.2MB drives do. Whether this will work with an 8" system w/o BIOS hacking depends... The main problem I can think of is with formatting. Chips like the NEC uPD765 do formatting largely automatically (for better or for worse), and will extend the final gap as long as necessary until it sees an index pulse. So they'll work fine with 1.44MB drives that are programmed like 8" ones, they'll just be a little slower and there will be a lot of wasted space on the end of each track. The WD179x chips however, format using a "write track" command which requires a byte of data (or a token representing marks or CRCs) for every byte (or byte pair anyway) on the track. That means if the track contains 20% more bits, the "write track" buffer needs to have 20% more data, and if the buffer doesn't have that much (it's always a good idea to add an extra dozen or so bytes of gap data at the end of the track with these controllers to allow for minor speed variations) then you'll end up writing a bunch of random memory (some of which may be interpreted as marks) on the end of the track until the index pulse comes around and terminates the write.

I'd like to second the endorsement of the SMC FDC37C65C+LJP floppy controller in Herb Johnson's column. I used one in an IDE/SCSI/FDC/RAM/COM*4 board I built for my old 8-bit IBM PC, all it took (over the buffering and address decoding that's shared with the other peripherals) was the SMC chip and five 150 ohm resistors. The data sheet warns that the chip is picky about ground planes, since I haven't made a PCB for this board (yet) I stuck a piece of pressure-sensitive copper foil to the underside of the chip and soldered jumpers to the ground pins, before plugging it into the PLCC socket. What makes the chip cool is its support of 2.88MB drives and the fact that the single density mode works correctly (unlike other PC-oriented FDC chips, which either blow off SD mode entirely or else require external connections and/or components). Unfortunately it doesn't generate the TG43 signal required to write most 8" drives correctly, even if you put it in the mode where it generates the equivalent signal (called /RWC for reduced write current) it makes the switch at the wrong track. That can be done in software though if you add an output port for it; it could be done in hardware too but that would get a little baroque, you'd need up/down counters to count steps and clear on /TK00, and some comparator chips (all duplicated for each 8" drive of course).

Speaking of 8-bit IDE, I'll be very interested to see the IDE article in back issue #69. I built an 8-bit IDE interface as part of my PC multi-I/O card and rather than use an LSI chip to handle the conversion between bytes and words, I used four TTL chips and one $1.19 PAL (which could be easily replaced with a handful of "glue" chips, I was tight for board space and had access to a PAL burner at the time). So I can definitely understand people's misgivings about using a $40 gate array unless you're paying for drill time on the PCB.

Keep up the great work! John Wilson <wilsonj@rpi.edu>

Thanks John for that clarification on drives. It seems everytime someone explains it, I find another fact I was not sure about. I always thought the 3 " drives were 360 RPM, or more accurately 300 RPM in 720 mode and 360 in 1.44 mode. Or am I getting the 3 and 5 mixed again, oh well I am sure we will hear more on this.

Like all of computing, there are many ways to build the same design (with and without PALs). That goes for chips too. I haven't had to talk directly to a FDC chip in some time now. Guess that is a problem with being an editor and paper pusher. Burning PALs really is a problem for the little guy. I found a Structured Design SD20/24 Pal burner for $10, but no manuals. Has a little 6802 in it, and a built in tape back up for files. Will not read burned PALs however which is what I wanted it for (just to make copies before they die).

As to E-mailing credit card number, I do get some that way, mostly Europe people, and personally I would feel safer if you did the check in the mail. It is far too easy for someone to get messages and take you number without yours or my knowledge. So keep up the good work and are you sure about those speeds? Thanks again John. Bill.

cc: B.KIBLER
Sub: Re: IDE I/O chip

HJ, BK,

I - JUST - got the latest *TCJ* ! Put me in the pot for two (2) of those wee little universal IDE boards.... It doesn't make much sense to buy just one. What price ? How the hell would I know. Have you purchased a reliable MFM drive lately ? Now ask me again, how much an IDE board is worth.

I have contacted MIX Software about putting MIX-C for CPM back out, or as Shareware, or PD... Waitting for them to get back to me. I have used MIX-C for the last 12 years, and except for the 10k hemeroid (stdio/headr) that is linked with each compliled program, its GREAT ! Any-way, thanks again for the opportunity to vent my guts. What happened to the Harris Z80H(25mhz) ????

Later, PLogan

Subject: Last call for MIX-C compiler for CPM-80
Bill,

Maria, at MIX Software (214/783-6001) indicates that they have 7 sets of MIX-C compiler for CPM-80. They will not be generating any more copies. They will not be releasing the compiler to Shareware, or Public Domain.

Looks like I will be shopping for another (supported) C compiler. Hate changing boats now that the river is long and wide.... Can't be THAT hard to write a compiler..... Maybe now is a good time to switch to a different compiler altogether.....

Thanks, PLogan.

OK and thanks for the word on MIX-C. You look at my Computer Corner article, I did find a place still selling CP/M software, including ToolWorks C. Not sure about support, but these folks have

been selling it for a long time and probably have some ideas about problems and such.

About the IDE board by Tilmann, got this last message from him:

To: B.KIBLER
Sub: GIDE Interest

Hi Bill,

You surely remember us placing a call-for-interest message in the previous *TCJ* issue. Unfortunately, only one single person has contacted me so far. I wonder if there really is such little interest in a theme which was discussed that extensively before.

I am now thinking about a modification which will hopefully increase public interest. This modification will allow for an optional real-time clock chip contained on the interface board. Perhaps you can place this information somewhere in the current issue and ask for requests again. You could also note that I don't understand that behavior (first discussing heavily, but then no real offers)!

Greetings, Tilmann

Well Tilmann and PLogan, I must agree. Where is all the requests that should follow the interest. I know I forgot to tell Tilmann, I will take a couple of whatever he makes as standard procedure. Now how about some you readers supporting our efforts. Not much of a request or expense. Herb Johnson is working with Tilmann to do a group shipment to him, and he will collect the money and distribute them on this side of the Atlantic.

I have a few machines that still need the old drives, so I keep an eye pealed at swaps, and let me say, it is getting very hard to find anything but IDE drives. So if you run anything using the old drives, you had better start making plans NOW to upgrade to IDE while it can be an easy change, and not after your drive has died with all your programs (you do back up - don't you?).

Well so much for the soap box, but really fans, please let the writers and people like Tilmann and Herb know how much you appreciate their work even if you don't need the card today, for you will need it soon! Bill.

Dear Bill;

Please find enclosed my check for $24.00 to renew my *TCJ* subscription for another year.

I thank you for the "Support Wanted" entry for my IMS International Model 518. The only reply that I received was from your regular author Rick Rodman.

Neither of us had any manuals for this thing, and he no longer had the IMS computer. Rick remembered enough about the IMS 5000 (NOT 518) to get me up and running. It was running, but there was no way in or out, except the keyboard and a hot wired extra serial port. That port, with printer attached, printed everything that went to the terminal screen, and nothing else.

I sent to him an 8" SSSD disk, holding the files that I wanted for the IMS. The most important, to me, of those files was IMP245 to be used for computer to computer RS-232 file transfer. In the selection of 50 overlays, there was only one that I thought that I could use. That was I2H8-5 Heath/Zenith 89 (8250 w/ baudrate generator). I used that one on my CCS S-100 rig. The IMS also has the 8250 UART.

Rick sent to me a 5" 96tpi disk that worked. The *.COM files all ran, the READ-ME Ascii files all typed. In the READ.ME file he included baudrate instructions that were different from that in the I2H8-5 overlay. Rick's special baudrate instructions worked, those in the overlay did not. Thanks again to Rick, I would not have thought of that one! He also expressed his doubts about the ability of the IMS 96tpi disk drive to double step for a 48tpi disk.

I took a good look at UNIFORM on the Epson QX-10. There I found "C DS DD 48 IMS 5000", wondering if the IMS would double step, I formatted a disk with this selection. I then loaded on some files, and tried it on the IMS. The IMS will double step, and read the 48tpi disk, but it ABORTS out when I try to write to it. The error message tells me that the disk is Write Protected.

The key to Rick's success was in the first line on the last paragraph of the READ.ME file, which read in part "and the printer/modem port at 28H". I did not know the port address. I did ' ..not know that the DB-25 receptacle marked PRINTER, is also the COMM port.

Many of Rick's articles are over my head, but he very graciously came down to my level of understanding. UNIFORM and 22DISK do not support this IMS 96tpi format. HATS OFF to Rick Rodman, a very good show for no manuals!!

Sincerely yours, Robert L. Edgecombe
9546 Los Palos Road
Atascadero, CA 93422
(805) 466-1619

Robert, I am glad Rick could help. It is one of the features our writers provide that I think many readers overlook. But then TCJ has a great staff of writers who get no pay and too little recognition for what they do. Guess they all may have to become "Hero's of The Year" at this rate. Thanks. BDK.

Wanted PC/XT Articles

Many of *TCJ*'s new readers have just purchased a PC/XT computer and are looking for help and advice at keeping these now collectible computers running. We are looking for hardware oriented help that has a general or platform independent concept about them. That means the material should have extra explanations for beginners and for others relating the information to non-PC/XT platforms. *TCJ*'s series on IDE drives is a great example of a cross platform article.

TCJ
PO Box 535
Lincoln, CA 95648

Regular Feature

Intermediate

Letters & Compupro

Dr. S-100

By Herb R. Johnson

"Dr. S-100's mid-Winter column" by Herb Johnson (c) Feb 1995
Internet: **hjohnson@pluto.njcc.com**

Introduction

While spring is in the air for my readers, I'm in the basement with my papertape reader/punch, and the system for the next few columns, the **Compupro 8/16**! After some correspondence and IDE progress reports, I'll continue from last column with my description of cards and the startup of this system. Oh, and I have some followup on last issue's foldout "star".

Networking for IDE

Tilmann Reh, the *TCJ* "German correspondent", has written a series of articles on his Z280 card and its IDE interface, which has led to his latest design of an **IDE interface** for any Z80 as a plug-in into the Z80 socket. I've been corresponding with him almost daily on design, construction, and import/export issues.

I encourage all my S-100 readers to read and consider his announcement from the last issue and (if it's here) this issue. Briefly, he needs to hear of **your** interest and **your** idea of what a reasonable price would be. With potential customers in hand, he can then get some boards produced and make some kits! My last correspondence with him suggests some expanded capabilities may become available! Even those readers with 8085, 8088, or other S-100 processors should be encouraged to respond, as Tilmann's design can be ultimately adapted to those processors. Contact myself or Tilmann without delay.

Correspondence

Addresses and such are in the References at the end of this column.

Stephen Shaw of **South Africa** asks for some **Apple IIGS's** :"I read about the Trenton Computerfest in *TCJ*. May I ask a favor - if you go next year could you keep your eyes open for Apple IIGS computers? We did not get many out here before Apple pulled out of the country in 1986 due to our government's racial policies. I noticed you sold an Apple II+ at this year's fair [*TCJ* #67 article]. I hope you will forgive me taking the liberty of writing to you like this, but we are three very enthusiastic Apple users who spend our spare time assisting the local school for the mentally handicapped, whose inmates use Apple II+ for word processing and general communications with visitors....[W]e have had dealings with other Apple II user groups in the US. They seem to be expensive and very profit oriented instead of trying to keep the Apple II alive."

"It is a great pity that we Apple users do not have a magazine that publishes circuits and other things to keep the Apple hardware side alive. It is heartening to see *TCJ* as it is a demonstration that some of us at least are determined to keep the pioneer computers alive." [*Stephen - TCJ is not adverse to being an Apple II support magazine also - there are many articles in the old back issues on Apples! BDK.]*

While I can't vouch for the writer personally, I can encourage interested people to contact him for more information. See the **Reference** section for an address.

Scott Barton of **Dublin PA** asks for **cassette-based** systems help: "Any kind of cassette controller, Tarbell, MITS, etc. Any cassette software, manuals, docs, or hardware. Also, any Altair, IMSAI, or Xitan related items." I'll send him my list. Scott previously bought a Xitan from me."It represents my primary interest in the hobby." Glad to help, Scott; even though you bought an IMSAI out from under me at the last Trenton Computerfest.

Arthur Smith of **Alexandria VA** asks about **drive conversions**: "There seems to be a lot of traffic [on the Internet comp.os.cpm area] regarding using 5-inch High density floppy drives to replace 8-inch drives. I've tried this and the results were lousy. Have you done such a thing and did it work? What am I doing wrong?" Arthur has a number of Compupro systems, and is also looking for "a small chassis to accommodate a 12-slot motherboard. I'm going to go out and buy a "mini-tower" and install a S-100 motherboard. I really am tired of lifting computers with large transformers!" Arthur would also appreciate any "late model" Compupro docs or software.

I'd kinda be skeptical that a 250 watt or 300 watt switcher could handle several Compupro cards, even with the 5-volt regulators removed; and the ventilation could be a problem too. But do the power consumption arithmetic (watts = volts * amps) and install a fan! As for the 5-inch/8-inch conversion: I believe the issues are rotational speed and media density, but in principle it should be doable [*watch Reader to Reader section for what others are doing. BDK.*] Either of these subjects are worth an article from Arthur!

I'd like to thank **Bob Harbour of Albuquerque NM**. He sent me some important info on Cromemco operation of CP/M. I've corresponded with him via Internet and he seems to be knowledgeable about these systems.

Amin Ismail of Dayton OH has some cards available: Konan serial and SMD hard disk controllers, IMS RAM cards, and a TEI mainframe S-100 box. Contact him for details.

Last issue's foldout

The editor of *TCJ*, **Bill Kibler** has a habit of running S-100 card schematics as his "foldout", usually independently of my articles. Last issue (#71) had a discussion of the **Hayes 80-301A** card, a popular 300 baud modem. It happens I have several of these, but no docs or software. Bill, can you give me a copy of the manual? *[in the mail..BDK.]* And, if anyone is interested in a card, contact me for terms.

Oldest continuous running system?

Paul D Willis of Coatesville PA takes the "lead" from Ramon Gandia: "Allow me to mention my IMSAI 8080 which I assembled in the fall of 1977. I was first introduced to the S-100 computer at college where we purchased an Altair 8800 in around 1975....When I graduated in 1977, I immediately purchased my IMSAI to keep active in the personal computing field. I originally used Processor Tech CUTER and ALS-8 as my operating systems, all cassette based of course. I still keep a modified working copy of ALS-8 to run for old times sake."

"Around 1982, I upgraded to a Northstar S-100 floppy disk controller and the NS DOS operating system. [This is how NorthStar got their start, producing systems for IMSAI and Altair computers.] This lasted less than a year when I finally got my Morrow Designs disk controller, two 8-inch drives, and CP/M. Somewhat later, I upgraded to an IMS Z-80 processor board which, with some minor modifications, worked with the IMSAI front panel. At this point, I was able to run NZCOM. I still use my system regularly for parts inventory and word processing. I also us it to produce BAUDOT code and American Morse to exercise my antique communications equipment."

"Any older systems still working out there?" Who's next?

Compupro 8/16

As I said last time, I recently acquired a Compupro S-100 system. The Compupro 8/16 was in part a reaction to the original IBM PC and in fact is generally superior to it. It "predicted" the availability of multifunction cards, and came with a lot more memory than the old PC or even XT! Here's a short list of the Compupro cards it has, not an unusual set for the time:

CPU 85/88: 8085 processor and 8088 processor, clock switchable between 2 or 5MHz on-the-fly, good to 8MHz.
Disk 1 floppy disk controller with two Qume 8-inch double-sided drives (also supports 5-inch).
Boot ROM for CP/M 80 or CP/M 86.
System Support I for console and boot support, serial, parallel, 4K ROM and/or RAM, timers, interrupt controller arithmetic chip, clock/calendar.
RAM 22: 128K by 16 bits, or 256K by 8 bits, static RAMs.
RAM 17(2ea): 32K by 16 bits, or 64K by 8 bits, static RAMs.
Interfacer 4: three serial ports, Centronic printer port, parallel port.
And some graphics cards...

and, my Jade Bus Probe a S-100/IEEE 696 card which displays **ALL** of the active bus lines, with a panel of LED's that extends above the cardcage.

Zenith fans may see in this configuration a Z-100 (or Z-121) system which, so I've been told, is a "clone" of the Compupro system. I'd appreciate it if someone looked hard to compare the two systems!

Card descriptions

The **RAM 17** supports "Extended addressing", which is the IEEE-696 extension of the address bus to 24 bits; and "phantom enable", which disables the RAM when the bus PHANTOM line is active (such as for power-up boot, when only a boot ROM is active on the bus). A convenient feature of the RAM 17 is the selective DISable of small blocks of memory near the top 64K, typically to permit ROM's or memory-mapped cards to operate. It is populated with 32 2K X 8 static RAM's, 6116's.

The **RAM 22** has four times the memory via the use of 32 8K X 8 static RAM's, 6264's; for a total of 256K bytes of memory. As it only makes sense to use in an extended memory system, there is no "selective disable" features, only a selection of the upper 6 (of 24) address lines for the appropriate bank of memory.

Both RAM cards monitor the "SXTRQ*" line on bus pin 58. When active, the memory card is being asked if it will support a 16-bit data access. The memory card asserts the SIXTN* line back to the processor card (or DMA card if it is bus master) and makes 16 bits of data available on the two 8-bit data busses. If SXTRQ* is not active, the memory card uses the bus in the "original" fashion of a pair of dedicated 8-bit output and input data lines.

The **CPU 85/88** is a dual 8085 and 8088 processor card. A convenient paddle switch at the card edge allows changing clock speeds between 2MHz and 5MHz. This and be convenient for testing slow memory, or just to slow down programs during debugging. Another reason for 2MHz operation is to support a **IMSAI front panel:** there is a DIP socket for it, and provision for disabling MWRITE (the front panel supplies it). And, the 8088 processor runs at 6 MHz, faster than the original PC at 4.77, so there!

The **System Support 1** card is used here as the console (terminal) interface. It includes interval timers, the serial port, and an arithmetic chip, the AM9511 (compatible with the Intel 8231 I think). Unlike the 8087 coprocessor for the 8088, this "coprocessor" is more like a "slave processor chip" that is I/O mapped. You feed data and computations to it, and receive results from it, under control of a program rather than a set of floating point instructions. It never was a cheap

chip, usually (and still is) above $100! There is also a "real time clock" calendar chip, the National MSM5832, with a connector to an optional battery to keep the time of day.

The Interfacer 4, is Compupro's serial card and parallel card. The three serial ports operate via 2651 (8051) UART chips. These are software programmable UART's, as opposed to the hardware strap-selectable features of the older 8050/2650 UARTs. There is a parallel port which is "Centronics compatible", namely it supports IBM-PC like parallel port printers; and a general purpose parallel port.

All the above are documented in the Compupro way, with signal descriptions, schematics, parts lists, and even some sample code!

The **Jade Bus Probe** did not come with this system. **Fred Hatfield** an old friend of 20 years, sent me this card last year along with a bunch of **Morrow** cards. Fred was an early supporter of personal computing in the 70's in Columbus OH, helping me and others in the first computer "club" in central Ohio. Today he still collects old computers in his native **New Orleans** and is active on the Fidonet CPMTECH maillist. Anyway, the Bus Probe is an "extended" S-100 card, with an array of 100 LED's extending inches above the other cards to display address, data, status, and other bus signals. In the absence of an **IMSAI** front panel, it's a very convenient display of activity. Since most S-100 processors operate at only a few megaHertz, you can actually follow the operation of a program. I'll refer to this display throughout this series.

I turned down the owner's original terminal, in favor of using my own favorite, the **Heath/Zenith H-19 terminal**. It has a nice crisp character display, can be programmed by simply putting it in local mode and typing escape sequences (good for changing baud rates!), and is a familiar part of the **Heath H-89** Z80 computer system. It is "VT52" compatible, an old DEC terminal standard similar to V100 and ANSI standards (ANSI.SYS for your MS-DOS fans).

Awakenings

Last time I told how I got this system. It was some months later that I got the time and space to begin setting it up. There was no great magic in connecting it all together, but a few points are worth mentioning here. (Similar information can be found in my *TCJ* **#58 (Nov/Dec 1992) S-100 article** on bringing up old systems.)

It's always a good idea to **inspect each cabinet**, examining connections, and powering up with the **box open for view**. This system came (as most do) with the S-100 bus in one box, and the dual 8-inch floppies in another box. Before powering up the S-100 system, I removed the cover (Compupro's come in covered metal cabinets) and **reseated each card** by pulling out the card, inspecting the edge connector for corrosion and shorts, and pushing the card back in. As I pulled out each card I also checked for **loose chips**: over the years, they can actually "wiggle out" of their sockets. Also, check chips for **corrosion at the pins** - I've seen it happen!

But be careful to **note where all the cables connect!** It can be difficult to read the docs and figure it out after you've pulled all the cables out! Generally floppy drives cables and connectors are not polarized, but no damage will occur if they are plugged in "backwards". That is, there is usually no "key" in the flatcable connectors nor in the edge connectors to keep you from connecting a drive "upside down" to a controller. However it is immediately noticeable on power up of both drive and controller, as the drive light IMMEDIATELY comes on BOTH drives (or one if you hook up a single drive). Just power down, reverse the cable and try again. All floppies have the odd-numbered pins grounded, and as the drive selects are active LOW, a reverse cable selects ALL DRIVES. This happened to me on this system.

When you first power up the computer or floppy drives, **look for smoke** and **listen for a "pop"**! This sounds odd, but tantalum capacitors, which are the small polarized caps of 10uF to 100mF generally used to filter the five-volt power lines to the chips, will sometimes short out over several years of non-use and actually **catch fire!** Or, they will **POP!** like a breaking dry stick. This is actually less dangerous than you might think, and generally IF BRIEF causes no other damage. You should power down, locate the fried component and replace it, and check all the fuses. Look for a charred device or circuit board area. Or, of course, for the source of smoke. Fortunately, this system (initially) passed the "smoke test".

Terminal tests

After the power-up tests, I connected terminal, drives and computer together. The most troublesome part of testing ANY "new" old computer is to figure out how to connect the serial terminal. I have a simple RS-232 "tester" which is simply a number of LED's in a small package that plugs between the terminal and the computer. You can watch the lights blink to see if the terminal or the computer is putting out a stream of characters, and with some experience get an idea of the baud rate (slow 300 or 1200 baud, fast 9600 baud). And, you want to make sure the terminal and the computer do not use the SAME line for sending characters: if they do, put in a cable or "null modem" that swaps the appropriate transmit/receive line pair (pins 2 and 3 on the standard DB-25 connector). Or...read the manuals, but most of the time who has them? Serial interfaces are very tolerant of this kind of abuse.

Of course, make sure your terminal works! The FIRST H19 terminal I tried had a bad connection to the filament of its CRT (picture tube, video tube) which caused the display to fade out and return only after beating the terminal severely. While this is very satisfying activity, it is not good computing practice. So I got another H19 terminal from the "good" pile and added this one to the "needs repair" pile.

Boot up

The short story is that, with minimal additional fuss, it booted up! I found the appropriate diskette (any disk that says

"MASTER" or "BOOT" usually will do, try not to use the ORIGINAL copy from Digital Research or, in this case, Compupro). It booted up in CP/M-80: the message read "64K CP/M-80 2.2m Compupro", which is the traditional boot up message for most CP/M systems. It says you have 64K of memory (less is actually available for programs), the version number (either 1.4 or 2.2), and some idea of manufacturer version or BIOS version (version m from Compupro). The usual commands worked: dir, stat, type, and so on. But, drive B did not seem to work. On the command "DIR B:", the drive light would come on (BIOS tells the computer to access the B: drive), but no seeks of the drive head mechanism were heard.

Hmmmmm.....Well, the drive was spinning, so it had 110VAC (some 8-inch drives use +24V DC for the drive motor); and the LED was lit, so it had +5 volts. Did it have all its other voltages: well, the OTHER drive did! How could that be the problem?

It turned out that EACH drive has its own POWER SUPPLY! AHA! It must be the +12V that powers the stepper motor that moves the head!. I replaced the 12-volt regulator, confirmed that the voltage was correct, and....no change! HUH???!??? Well, a false step. I changed the drive selector jumpers on each drive, and the B: drive, now the A: drive, still did not work. I swapped the cable connections....and it worked! NOW the original A: drive didn't work! How could it be the CABLE CONNECTORS? Hmmmm...what's that GREEN STUFF on a few of those pins??? Turns out to be that **corrosion** problem I mentioned earlier!

A couple of tricks I learned years ago came into play: I took a rubber eraser, cut it in half to make it thin, and used it to rub out the corrosion. I also took some **very fine grade sandpaper**, 600 grit, and lightly inserted it in and out of the cable connector to loosen the corrosion. Short of replacing the connector or the cable, these served to conveniently clean out the problem. I reassembled the drives and cables, and both drives operated correctly.

New Trouble!

Wouldn't you know...while writing this article I ran the system for testing. After several minutes of operation, I couldn't get it to reboot! After trying a few different disks while looking at the LED's, and while observing the behavior of the floppy drives, that it was doing SOMETHING. In fact, I could enter commands and the drive light would come on each time! After checking that the terminal was working (by shorting pins 2 and 3, transmit and receive, forcing it to echo its own output), I noticed that the "-18V" light on the Jade Bus Probe was OUT!

I checked the power supply visually. Like most S-100 systems, the Compupro supply consists of a transformer, diode rectifier (2 diodes, four, or a "bridge" of four in one package), and a HUGE electrolytic capacitor. And, in this case, a blown fuse! Fuses don't die for a reason, so after replacing it with a SLOW-BLO fuse (a fuse that takes a few seconds to activate), I powered up again while WATCHING very closely. Sure enough, the fuse went out, but not before I smelled the familiar odor of burning capacitor! Quickly I cut power and followed the scent to the rear of the chassis.

I PULLED THE AC PLUG from the back (very important), and pulled out the back three cards. I smelled each card, and found one that seemed to have the scent. Since this was a SHORT (what else would blow a fuse?), I got my voltmeter and set it to "continuity", where it would make a beep upon reading a short. I fetched an extender card which is marked with all the S-100 pins and saw that pin 52 was "-18 volts". Measuring each of the three cards from pin 52 to ground, only the disk controller (DISK 1) had a short reading!!! I visually followed the circuit trace to the first capacitor - most tantalum capacitors look like small colored gumdrops - desoldered it out, and remeasured the resistance at pin 52 to ground. The short was gone. A new cap (actually, an old one from a previously salvaged computer) and a new fuse, and all was well.

Next issue

People tell me they **really enjoy** hearing all these details about repairs and debugging hardware and so on: let me know if you'd rather see lines of code, some schematic scribbles, or whatever. I PROMISE I'll get serious next issue....if nothing else needs fixing...if the basement doesn't leak...And, by the way, I'll tell you a bit about the world of **paper tape**. Be there, aloha.

Graphics or math? - help me out!

There were two graphics cards with this box: One is a **Microsprite** card, supporting a standard color video (TV) monitor. The other, from **Illuminated Technologies**, using a digital RGB monitor. AND...an old **Canon PJ-1080A** color printer as well! "But it probably needs a new ink cartridge", the previous owner said. Anyone out there with paper, cartridges, or software for the Canon 1080A? Or software for testing the 9511 math processor? Or software for the graphics cards? And any interesting programs for the timers or MSM5832 clock chip?

References

Herb Johnson, CN 5256 #105, Princeton NJ 08543. (609) 771-1503.
Tilmann Reh, tilmann.reh@hrz.uni-siegen.d400.de
Stephen Shaw, PO Box 1404, Randfontein, 1760, South Africa
phone +2711 (011) 414-1608; Internet stephen.shaw@birdsnest.fast.iaccess.za
Fidonet 5:7107/22
Scott Barton, 113 Olde Pilgrim Rd, Dublin PA 18917.
Bob Harbor, 6609 Barnhart NE, Albuquerque, NM 87109-4106.
Frank J Wilson, PO Box 55, Tomales CA 94971
Microfax, 5620 F Kendall Court, Arvada CO 80002 (303) 467-1207
Dan Slayton, 1500 NE 40 PL, Oakland Park FL 33334
Amin Ismail, 5640 Waterloo Rd, Dayton, OH 45459, (513) 435-4476
Internet:
ISMAIL@udavxb.oca.udayton.edu

Regular Feature

68xx/68xxx Support

C & Assembly

Small System Support

By Ronald W. Anderson

Progress in C

I mentioned a book last time that has good information on the printer port and how to use it for other things. One of the things I found out during the debug session today is that there is another error in the book. It reports a 4 bit input port to be found at address BASE+3. That is, if the port is at hex 378, the input port is reported to be at hex 37B. After placing signals on the input bits and not being able to read them, I thought I'd try the only other possible address in that group, Hex 379 (BASE + 1). It then worked immediately.

Obviously I need a little more experience with this port and it's interface before I write any articles on how to do something interesting with it. I promise to do that soon. Presently at work I am running an LCD display interface and a keypad scanner on one parallel printer port. These are all functions internal to our computer box. I suspect we'll have to get fancier and worry about buffered inputs and outputs if we want to run signals outside of our box (other than to a printer).

C

I have the OK from Bill Kibler to present some material on the C language here. I thought I might start with a few introductory comments. First of all, if you don't know C, you will want to start with the latest ANSI (American National Standards Institute) version. It is far easier to use than the older and original K&R version in my opinion. If you've dabbled in the old version and then get into the new, you won't think so at first. Those #$%^&* standards committee folks have really made it slower and more frustrating to use. You won't believe that for long, however. The older version leaves a lot of things to the programmer. That is, the compiler doesn't check a great deal for dumb errors. The Ansi version is a lot better in that respect.

Having said that however, I have to assume that most of you will be using older computers like my old SWTPc 6809 system on which I run Windrush McCosh C, which is an old style version that complies with the old K&R standard. I'll concentrate primarily on the old style C and mention the places where the ANSI version is different (and better).

It is fair to contrast C with Pascal because Pascal was the language of favor a few years ago, before C really caught on. I dabbled in programming for a long time in assembler and BASIC, but when Pascal came along, I really learned a lot about how to program. I resisted C for a long time, as a matter of fact until a few years ago when it became plain that C was to be the language of choice for the immediate future.

Pascal was developed by a computer science professor to teach students how to program. Niklaus Wirth, the author of Pascal made it very hard for a student to do something dumb. For example if you want to add decimal 32 to an ASCII character to convert it from upper to lower case, you can't just mix types in an expression. You must explicitly convert the character to an ordinal (integer) value, add the integer 32 and convert the result back to a character:

```
CH := CHR(ORD(CH)+32);
```

All this is in the name of keeping the programmer aware of what he is doing. C was written by some scientists at Bell Labs who wanted to have a language that was concise and convenient to use to write operating systems (UNIX in particular). In C the conversion is done simply by:

```
ch +=32;
```

Actually C has a library function called tolower(). You can simply call that function:

```
ch = tolower(ch);
```

The price for this simpler approach is that you have to think more. In this case you know what you want to do and Pascal frustrates you by not letting you do it unless you tell the compiler that you know exactly what you are doing. C, on the other hand really has the philosophy "Good luck, you are on your own!" To be sure automatic type conversions have tripped up nearly all C programmers at one time or another, but I think we would rather put up with discovering our own stupidity now and then than to have to "cajole" the language into doing what we want it to do!

ANSI C plugs more of the holes and keeps you from doing REALLY stupid things most of the time. Believe it or not, C is a very simple language. There are few constructs to learn, and few symbols to memorize. What complicates C is the way in which you can combine the simple elements to do complex

things in one line. My favorite example is the one line loop that can copy a whole file. Suppose you have opened a file to read, and created a file to write, and that you want to copy the first file to the second. In C, you can write the loop that reads characters from the input file (infile) and writes them to the output file (outfile) AND checks for the end of file, all in one line:

```
while((ch = fgetc(infile)) !=EOF) fputc(outfile,ch);
```

This simple one line loop will work correctly even if the input file is empty and the first read gets the EOF flag. In Pascal it would look something like this. (No guarantees that my READ and WRITE calls are correct, but the form is correct).

```
READ (INFILE,CH);

WHILE NOT EOF(INFILE) DO
        BEGIN
        WRITE(OUTFILE,CH);
        READ(INFILE,CH);
        END;
```

You have to read a character from the input file before starting the loop, just in case the file is empty and the first read sets the end of file condition. Then you enter the loop that runs while not end of file. Since you have already read a character, you have to write it and then read another character, then go around the loop again. Frequently a While loop will need to be "primed" like this. C avoids that priming by allowing you to include the reading in the test. I must admit the C version looks more cryptic at first glance, but it is perfectly straightforward. The whole assignment statement takes on the value that was just assigned, so (ch = fgetc(infile)) takes on the value just assigned to ch. The only catch is that the assignment has lower priority than the test, so you have to parenthesize it to "hold it together". C has some rather complicated precedence rules, but you don't have to memorize all of them at once, just remember when in doubt to use parehtheses to insure proper order of evaluation.

It has been said by at least one computer science type that he was suspicious of any language that contained more punctuation than words. C certainly looks as though it qualifies. Here then is lesson 1 adapted from a quick course on C that I taught at work.

C Class Notes (Introduction)

C has been described as "Pascal with it's sleeves rolled up". The original C did very little error checking to keep the programmer from doing himself in by making a dumb mistake. Several years ago it was realized that C is a useful language (probably by the realization that a lot of people were using it) so a committee was established at the American National Standards Institute (ANSI) to write some standards for the language.

Now we have a version of C called ANSI standard C that has a few more checks on the programmer. It still, however is primarily designed to let the programmer do anything he wants to do (including shoot himself in the foot). Experienced programmers tired quickly of Pascal's preventing them from doing something that was perfectly logical. They spent all their time "programming around" some built in protection in Pascal. When C came along they felt a great freedom to program concisely. I think this is the primary reason for the rapid acceptance of C.

Let's write our first C program, a simple one to print "Hi there" on the screen. Here is the program:

```
/* First C program */
#include <stdio.h>
char message[] = {"Hi there"};

main()
        {
        puts(message);
        }
```

The first line is enclosed by comment delimiters /* and */. The compiler ignores all comments. They are for someone who reads the program, not for the computer. The C language as such has no facilities for input and output of information. These facilities are included in what the authors called the "standard library". The standard library is a library of functions (written in C) that are used to get information into and out of a C program. One of the functions in the standard library is "puts()" which will output a literal string or the contents of an array of characters. Just how it does this is something we will get into in more detail later. We can only use the function puts() if we include a file called stdio.h in our program. (That last statement is not strictly true, but in general the appropriate file must be included in order to use a library function). Such files are called "header" file and they declare functions so that they are usable by the program. We will have a lot to say about the standard library functions and various header files later. The C compiler library function documentation always indicates which header file or files must be included in your program in order to use the function.

The third line of the program "declares" a variable called "message[]". The inclusion of the square brackets makes message an array variable. The type "char" that precedes it make it an array of type char (character). The message in quotes and enclosed in curly braces is the "initializer" that sets the values in the array of characters to the characters "Hi there". The C compiler automatically adds a NULL character to the end of a string. a NULL is represented by the binary code 00000000. A space is not the same as a NULL. The code for a space is hexadecimal 20 or binary 00100000. When an array is initialized with an initializer as this one is, the compiler figures out how long the array is, and it doesn't need a "dimension". If the array is not initialized you must supply a dimension that tells the compiler how big the array needs to be. In other words, the

compiler need to know how many memory bytes to reserve for the array. In that case, for example, you would simply declare it as:

```
char message[20];
```

That would create space for an array of characters that could hold 19 characters. (Remember that NULL that C adds will take up one character space). It does NOT define the contents of the array. Uninitialized arrays may contain anything at all. There are other ways to put information into an array than to use an initializer. The standard library has string functions to copy and join strings into a character array, and you can assign values one character position at a time.

Moving to the next line, a C program MUST have a function called main(). This is the place where the program execution begins when you run it. The rest of our simple program is the main function. The main function contains one "statement" puts(message);. The word "message" is a "parameter". Functions may have one or several parameters "passed to them" when they are "called". The statement is bracketed by the begin { and the end } characters. A function must contain these brackets to delimit it. Such begin - end pairs are used to delimit (i.e. define the beginning and end of) compound statements and loops as well. We could well write our own function to do just what puts() does. Below is a simple one called printit(). Realize that puts() is not magic, but a simple function written in C very much like the one given below. (The whole situation here is analogous to our study of FLEX and our use of an operating system call, and then later writing our own function to replace it).

```
printit(string)
char string[];
        {
        int n=0;
        while(string[n] != 0) putchar(string[n++]);
        }
```

The second line identifies the type of the parameter "string". This is the "old style" parameter definition. New ANSI C would do it this way:

```
printit(char string[])
```

That is, the type of the parameter is defined directly in the function "declaration".

Of course we haven't eliminated the standard library here, just pushed the interface or call to it down one level. We have used the putchar() function of the standard library, which simply outputs one character (passed to it as a parameter) each time it is called. Then we've defined a loop that moves through the array one character at a time and outputs it by using putchar().

We access or read the contents of the 10th location in the array by using the array index. That is, since C calls the first location 0 and the last one 19, we would access the tenth item as message[9]. In the above function the array index variable n starts at 0 and is incremented until it finds the null or 0 that C has put at the end of our string. A good practice would be to stop printing the string either when a null is found or when the string length is reached, i.e. in the case of dimension 20, we could use:

```
while((string[n] != 0)&&(n < 20)) putchar(string[n++]);
```

The double ampersand (&&) is the logical AND symbol. The statement says to output characters while the present character is not null and while the count is less than 20. I ought to mention here (so I won't get letters from C programmers telling me that I did something that was unnecessary) that a value of zero or null is considered FALSE by C and anything else is considered TRUE. Thus the "!= 0" part of the above is unnecessary. a simple "while(string[n])" is sufficient. You might think of the != 0 as being "implied". I've included it in this first example for clarity until I could explain it. You might be puzzled by the [n++] notation. This is the last place in the loop where n is used. n++ tells the compiler to use the current value of n as the subscript for the string array reference and then increment n (add 1 to it). It is shorthand for the inevitable array index increment n=n+1; statement.

Let's discuss briefly the "declaration" of printit(). The "parameter" in it's declaration is "char string[]". This simply says that the function printit() expects to have the location of a string array given to it by the main program. In the main program we would "call" the function with the statement "printit(message);". The name message is the name of an array and C therefore "passed" the address of the array to the function "printit". We will of course get into this in a lot more detail a few sessions down the line. As we continue you will find that there are numerous ways of doing anything in C. It is very flexible. For example the puts() function will accept a literal string:

```
puts("Hi there");
```

In other words we wouldn't have had to define an array of characters and initialize it to "Hi there" in the first place. Our program could have been simply:

```
/* first program in C */
#include <stdio.h>
main()
        {
        puts("Hi there");
        }
```

C doesn't care about indentation or new lines. The program would compile and run the same way if it were:

```
#include <stdio.h>
main() { puts("Hi there"); }
```

There may be several statements on a line or a statement may

extend over several lines. A statement is terminated by a semicolon.

The #include directive must be on a line by itself because everything after the filename is ignored or treated as a comment. The main function, however can be all on one line. The reason we use indentation and new lines is simply to make the program more readable to a human reader.

This has been a quick look at what a C program is like. In the coming lessons we will look at each of the items here (and many others) in much more detail. Learning C is much like writing a program. If you try to do it all in one step it is a real chore. If you build it a piece at a time it turns out to be a lot of simple things put together.

I've been trying to learn C++, the object oriented version of C. I found it overwhelming at first until I read and reread the appropriate material enough so that the newly defined words began to make some sense. Each time I read it I learn something else and wonder why I didn't understand it the first time through. Repetition helps as does actually writing and running programs.

Assembler

Now we will continue our discussion of programming in 6809 assembler with lesson #4. Last time we read an integer number as an ASCII string and printed its binary representation. Maybe we ought to talk about disk files this time. At the assembler code level, in order to do anything with a disk file you first have to establish a data area of 320 bytes called a File Control Block or FCB for short. A disk sector in 6809 FLEX is 256 bytes long. The first four bytes contain housekeeping information and the next 252 contain data stored in the disk file.

The first 64 bytes of the FCB are used as the interface between the user and the operating system. All references here start at byte 0, i.e. as in C, the array index starts at zero, not 1. To open a file for read, for example, you put the filename (8 characters) into FCB locations 4 through 11. If the name is shorter than 8 characters you "left justify" them and fill the remainder of the 8 positions with zeros. Positions 12 to 14 are for the three character extension of the filename. Again if the extension is shorter than three characters you left justify them and fill with zeros. Next you put the drive number ($00 to $03) in byte 3 of the FCB. You put the "open for read" code ($01) at byte 0 of the FCB and do a system call (JSR FMS). You don't bother clearing or initializing the rest of the FCB.

After the FMS call, if the operation was successful (i.e. the system found the filename in the directory and opened the file) the error code byte (position 1) of the FCB will contain a zero and position 3 will contain the operation code (1). Generally you don't have to bother with the returned op code, just check to see that the error code is zero. The most likely error on an attempt to open a file for read is error $04, returned if the file could not be found in the disk directory for the specified drive. The file is not there, you specified the wrong drive, or misspelled the filename.

If the file opened successfully, FLEX changes the function code byte 0 to zero. This tells you that a file open for read is in the "read the next character" mode or if open for write is in the "write the next character mode".

Now you can read the file a byte at a time. You JSR FMS again and the first sector of the file is read into the last 256 bytes of the FCB, but FLEX takes care of more than that for you, the first byte of the file is returned in ACCA. On successive read operations (simply successive JSR FMS instructions) FLEX maintains a pointer to the next byte in the FCB to be returned, and feeds them to you one at a time. When the last byte of the sector is given to you, the next call to read a byte results in the next sector being read into the FCB. At the end of the file, an attempt to read another byte results in an error code in position 1 of the FCB. The EOF error is 8. After each byte is read, your program should test the error code for non zero. FLEX has made that easy for us by setting the zero flag in the condition code register to correspond with the error code. Rather than having to test the second byte of the FCB you can do a simple BNE ERROR.

The ERROR routine checks the error number and if it finds error 8 it exits the read file loop and closes the file. That is, you test for EOF and if it is found insert the Close code ($04) in byte 0 of the FCB, and do one last JSR FMS to close the file. Again check the error byte to see that the file closed successfully (error code 0), and you are done.

Writing to a file is very similar. You put the filename in the FCB along with the open for write code. Error codes are found in byte 1 of the FCB if the file already exists or if it can't be opened because disk space is all used or there is a disk error of some sort. Just as in reading a file, you feed the FCB a byte at a time (LDA byte, JSR FMS) and it places each byte in successive locations in the FCB. When a sector's worth of bytes is reached, FLEX writes a sector and lets you fill it's sector buffer again. When you are done writing you put the close file code 4 in byte 0 of the FCB and JSR FMS, then check the error byte.

Flex, believe it or not, actually has file compression built-in for text files only. It is rather simple as compression algorithms go, but it is effective. FLEX counts successive spaces on writing a text file to disk, and then writes a special two byte code to the disk if more than two spaces are found. It writes a horizontal tab character ($09) followed by the count of the number of spaces. When FLEX reads a text file it sees the $09 and interprets the next byte as a count of spaces to output to the user program. Thus 12 space characters are encoded $09,$0C, taking only two bytes on the disk. This scheme, used in FLEX in about 1978 is still beyond what MS-DOS does (exactly nothing at all other than to accept TAB characters $09 which the writing and reading program must interpret as the same number of spaces or the text format gets all fouled up). This

compression - expansion is entirely transparent to the user. FLEX FMS (File Management System) handles it all.

There are two kinds of files in FLEX, the text file described above and the binary file. A binary file is written and read exactly as you pass the bytes to the FMS. Executable files are always binary files and files containing text are treated as text files. When you write and read files by means of an assembler program you must set a "flag" byte in the FCB (byte 59) to $FF to signal a binary file. The default is a text file. This flag must be set AFTER the file is open but BEFORE any bytes are read or written. The code would be:

```
        LDX     #FCB
        LDA     #$FF
        STA     59,X
```

FLEX has an easy way to specify a default extension if none is specified. There is a system call SETEXT. You pass a code to SETEXT in ACCA and if an extension is not found in the FCB, SETEXT will set it according to the code. Setext must be called when the filename is in place in the FCB but before the file is opened. FLEX uses a number of default extensions. A text file as in the one I am writing at this moment generally has a default extension .TXT. The Assembler produces a file with the default extension .BIN (binary). Flex commands generally have the extension .CMD. A .CMD file is a binary file and you can convert a .BIN to a .CMD simply by renaming it such. BASIC uses .DAT for data files. Some programmers use .SRC for assembler source files, though I generally stick with .TXT. Any of these defaults can be overridden simply by using an extension. EDIT FILE will edit a file called FILE.TXT. (assuming your editor command is EDIT.CMD). EDIT FILE.DAT will edit a file called FILE.DAT.

Perhaps it is time to show some demo code. How about a program that can read it's own source file and dump it to your terminal? If you are a bit apprehensive to start playing with files, format a disk and edit this program file on that disk. Then assemble the program and try to run it. If you have made an error that could possibly damage the files on the disk, you'll only be out your work editing the file and assembling it. If you want to protect yourself against that possibility, back up your disk before you try to run the program. Actually opening a file for read is pretty safe. These precautions may be overkill until we get to the opening of an output file and writing to it.

```
* PROGRAM TO DUMP ITS OWN SOURCE TO YOUR TERMINAL

NAM     SELFDUMP

PUTCHR  EQU     $CD18
WARMS   EQU     $CD03
RPTERR  EQU     $CD3F
FMS     EQU     $D406
FMSCLS  EQU     $D403

START   LDX     #FCB
        LDA     #1          OPEN FOR READ CODE
        STA     0,X
        JSR     FMS
        BNE     ERROR       FMS SETS NON-ZERO CONDITION
*                           ON ERROR
LOOP    JSR     FMS         READ A CHARACTER
        BNE     ERROR
        JSR     PUTCHR      WRITE IT TO SCREEN
        CMPA    #$0D        IS IT A CR?
        BNE     CONTIN      IF NOT, OK
        LDA     #$0A        ELSE WRITE LF ALSO
        JSR     PUTCHR
CONTIN  BRA     LOOP        GO AROUND AGAIN

ERROR   LDB     1,X
        CMPB    #$08        IS IT EOF
        BEQ     DONE
        JSR     RPTERR      TELL USER WHICH ERROR
        JSR     FMSCLS      CLEAN UP BY CLOSING OPEN
*                           FILE ON ERROR
        JMP     WARMS

DONE    LDA     #$04        BRANCH HERE ON EOF
        STA     0,X         CLOSE THE FILE
        JSR     FMS
EXIT    JMP     WARMS

* THIS IS THE FILE CONTROL BLOCK NEEDED TO OPEN THE FILE
FCB     FCB 0,0,0,1
FCC     /SELFDUMPTXT/
        RMB 305

        END START
```

I don't think there's a whole lot new here. We haven't discussed the FLEX error handler. RPTERR reports the error number on the screen. If you have ERRORS.SYS on your system disk, it also prints an error message. If the file is not present you will get DISK ERROR 9, for example.

We finally have an example of full use of indexed addressing with an offset here. We did a LDX #FCB at the beginning of the program. Then we treat FCB like an array in BASIC. Instead of using FCB(13) to access the 13th byte of the array, we use 13,X. It amounts to about the same thing. The FMS calls do not change the value in the X register, so you don't have to reload it after any or every FMS call.

One confusing spot might be the line that reads FCB FCB 0,0,0,1. The first FCB is a label for the File Control Block. The second is the assembler Form Constant Byte directive. The 1 is the drive number. If your working drive is other than 1, change this number to that of your working drive. Next time we'll show you how to default to the working drive. Next we have the FCC (Form Constant Character) directive. Notice that the period between the filename and extension is missing. Remember that the separator is not used in placing the name and extension in the File Control Block. The filename I chose is 8 characters long so there need be no padding zeros between filename and extension.

This is sort of cheating. Since this program is only going to read one file (its own source file) I "hard wired" the filename into the FCB. Take care not to call your program something else, or if you do, make the appropriate changes in the FCC line above. If you choose a filename that is not 8 characters long, you have to fill the remaining bytes with 0s. You might do that as follows:

```
FCB     FCB     0,0,0,1
FNAME   FCC     /TEST/      FOUR CHARACTERS
        FCB     0,0,0,0     PLUS FOUR 0'S FOR A TOTAL OF 8
FEXT    FCC     /TXT/       PLUS EXTENSION
```

Had I not hard wired the name we might have hand coded:

```
        LDA     #'S
        STA     4,X
        LDA     #'E
        STA     5,X
        LDA     #'L
        STA     6,X
        LDA     #'F
        STA     7,X         ETC. ETC.
```

FLEX stores text files with each line terminated only with a CR. If you read such a file and write it to a terminal you will have all of the lines of the file printed to the same line on the terminal. Each CR needs to have an LF added so the cursor of the terminal moves to the next line. The above code therefore checks for CR ($0D) and adds LF ($0A) when one is detected.

I have explained the JSR FMS, I think, but there is another FMS routine called FMSCLS (FMS CLoSe) that is good to call on an error exit. If there is an error closing a file, for example, FMSCLS is supposed to close all open files and bail out. The manual warns that it is not good practice to use this 'cheat' call to close open files at the end of a program. It is better and safer to close each file individually by using the $04 code.

I am not the world's best program comment writer, but I think I do better than some. The comment shouldn't just echo the command. For example LDA #$0D doesn't need the comment "Load the Accumulator with CR". The LDA says to load the accumulator. The comment ought to say something about why the instruction is being used.

Actually this program leaves something to be desired in that I have used litteral constants rather than naming them. The program becomes more readable and easier to change later if we do the following to it:

```
* PROGRAM TO DUMP ITS OWN SOURCE TO YOUR TERMINAL
NAM     SELFDUMP

PUTCHR  EQU     $CD18
WARMS   EQU     $CD03
RPTERR  EQU     $CD3F
FMS     EQU     $D406
FMSCLS  EQU     $D403

* PROGRAM CONSTANTS
LF      EQU     $0A
CR      EQU     $0D
FOPENR  EQU     $01         OPEN FOR READ CODE
FCLOSE  EQU     $04         CLOSE FILE CODE
EOF     EQU     $08

START   LDX     #FCB
        LDA     #FOPENR     OPEN FOR READ CODE
        STA     0,X
        JSR     FMS
        BNE     ERROR       FMS SETS NON-ZERO CONDITION
*                           ON ERROR
LOOP    JSR     FMS         READ A CHARACTER
        BNE     ERROR
        JSR     PUTCHR      WRITE IT TO SCREEN
        CMPA    #CR         IS IT A CR?
        BNE     LOOP        IF NOT, OK
        LDA     #LF ELSE WRITE LF ALSO
        JSR     PUTCHR
        BRA     LOOP        GO AROUND AGAIN

ERROR   LDB     1,X
        CMPB    #EOF        IS IT EOF
        BEQ     DONE
        JSR     RPTERR      TELL USER WHICH ERROR
        JSR     FMSCLS      CLEAN UP BY CLOSING OPEN FILE
*                           ON ERROR
        JMP     WARMS

DONE    LDA     #FCLOSE BRANCH HERE ON EOF
        STA     0,X         CLOSE THE FILE
        JSR     FMS
EXIT    JMP     WARMS

* THIS IS THE FILE CONTROL BLOCK NEEDED TO OPEN THE FILE
FCB     FCB     0,0,0,1
FCC     /SELFDUMPTXT/
        RMB     305

        END START
```

Notice how the naming of constants makes a few of the comments redundant. Some of them could be removed without any net loss of information. Using names such as these results in what is called "self documenting code".

Perhaps next time we will show how to read a filename from the command line and put it into an FCB, then open the file and write it to the screen, a LIST utility for text files. Later yet, we will open a file for WRITE (scarry the first time) and write some text to it.

Just a quick little note here. Anyone out there interested in corresponding with me in Spanish? I know just enough to get in trouble. (I know enough to ask a question but not enough always to understand the answer). I spent one year so long ago I don't want to say, in high school Spanish, have been through the Foreign Service Spanish tapes a couple of times, spent four weeks in Mexico, went through the Foreign Service Portugese tapes and spent three weeks in Brazil. I wish fervently that there were a Spanish TV station in Detroit. I think an hour a night would get me going pretty quickly. I need practice in VOCABULARY!

I'm sorry, but I can't resist a little silliness here. I had a case of the Flu last week and got food poisoning on my mind. I thought of a good name for a Dick Tracy mobster. How about Sal Monella? Then there's the new restaurant up the street, Sam 'n' Ella's Cafe. I know a Professor at the University of Michigan, appropriately named Professor Learned. It is reported that there is at least one dentist named Dr. Payne. (I think I'd change my name to Comfort). With that, I'll stop until next time.

Special Feature

Hardware Support

Designing With PLD's

Beginning PLD

By Claude Palm

When Claude sent me his IDE material, he indicated he had some helpful words to say about developing PLD projects. I indicated a strong interest and he returned this letter. So if your considering PLD's in your next project, read on, else enjoy the fact you don't have to learn PLD's. BDK.

Dear Bill,

Thank you for publishing the PT IDE802 specs in issue 70. In response to you 'arm twisting' I put together the following general requirements and pitfalls for PLD design. As you can see there are many advantages, but also a few gotchas. And a good system will set you back around $5000, but there are ways around this.

But I'll begin with some response to your comments on the PT IDE802.

(1) You do not need to write two bytes or use the hardware handshake option. The example in the data sheet is only a demonstration of it. If you address ports 2-3 instead, the hardware handshake is disabled i.e. write the data once to CENT DATA 2, then toggle the STB line via bit 0 in CENT CTRL 3 port.

(2) I wanted the interface to be 100% compatible with the CAM (Common Access Method Committee) ATA (AT Attachment) standard, hence the separate CS pins so the address space can be split into two a la IBM (1Fx/3Fx for CS0/CS1). As the two signals are required by the IDE drive (if all features are to be used) they may as well be used by the chip, rather than bring in an extra address line for 16 consecutive addresses.

It may have been better to generate these CS signals internally and then bring them out to the drive, but that would require relinquishing a pin from interrupt or speaker output. I decided they would be more useful, as an extra CS were in many cases already available or could easily be provided. As the IDE drive only uses two ports with CS1 is selected, the remaining addresses were utilized for the Centronics interface. But nothing prevents me from providing another version of the chip with internal CS0/1 generation and only two interrupt outputs. 1 CS and 4 address inputs, CS0,1 and 3 address outputs.

Incidentally, if the CS1 pin is tied high the chip mimics the WD1000/1001/1002 HDO controller board interface except for DMA Acknowledge pin which is handled differently by the IDE or not at all. That pin was seldom used with those boards. At least I never saw one. A PT IDE802, 4 resistors and decoupling caps are the only components required to produce a WD100x replacement board. Existing software will normally work as is. Provided it loads all task file registers before each sector I/O and does not rely on previous data in these registers (the IDE drive maintains the task file differently). I did run the chip on a TRS-80 to test it, and it worked o.k. with existing software.

(3) The timing constraint (CS0 - IOCS16 - RD/WR) did stem from the IDE drive which does not advise if the next data I/O is to be 8 or 16 bits until some time after the data port has been selected. The chip of course needs to know this when a read/write commences so that it can decide whether or not to issue any RD/WR pulse to the drive, and how to gate the appropriate data. The only time the IDE performs 8-bit data I/O is in the read/write LONG command, for the 4 ECC (Error Checking and Correction) bytes transferred after the sector. But again I wanted 100% compatibility.

I have already relaxed these timings to the detriment of the RD/WR LONG command, but I felt that the benefits outweigh this, as the LONG instructions are seldom used. The main timing constraints are now due to the IDE drive itself. If in doubt, check the manufacturers specs, or the general ATA specs for the three different speed modes and the specs are available from CAM.

Someone also suggested a 6502 like interface which may have possibilities (a single RD/WR* pin in conjunction with an enable). Problem is that you have to make these decisions when designing anything: To do it one way or the other. At least with PLDs you only have to revise the software to make the changes, so expect a few different revisions to become available. Other possibilities include an I/O device rather than Centronics i.e. an 8-bit bidirectional, 8-bit input, and a 4-bit output port, with the IDE also usable as a straight 16-bit I/O port. If anyone contacts me with a specific request, and I think it useful, I am willing to supply that version at no extra cost.

Developing the PT IDEs

As for developing the chip, it worked first try, not counting dry runs on the simulator, but then the various modules had been

tried and tested before. I did a couple of changes from the prototype to improve the speaker output (could only be set and toggled before) and also to allow all task file registers to be accessed without upsetting the sequencer.

When developing the original IDE engine I had a misunderstanding of the IDE drive, and ended up with 511 byte sectors, but these things happens in the best of families. I also had a more serious instability problem that was hardware related: The chip supplied incorrect data at certain patterns. That took some finding, but was tracked down to the IDE drive not yet driving the data inputs while the chip was starting to drive the S100 bus. Due to heavy loading on the data out pins by the bus, with still floating IDE inputs, the chip started to oscillate until the IDE asserted the data inputs. This only occurred on certain data patterns, causing the chip to issue the odd extra read pulse to the drive which acted on it, and promptly produced the next word of data while the S100 bus was still busy reading the previous byte. The pull-up resistors on the IDE socket in the Hardboard was the resulting cure.

Designing with PLDs in general

Many years ago I made the decision to start using PALs (the bipolar versions) when I read about the new EPLD chips by ALTERA. After some investigation I got hold of the INTEL 5C032 (I think) and did some experimenting. They came out rather well, and after I obtained some of the larger 5C060 chips I was hooked, even if they were still rather slow devices in those days. They were also expensive, but like all chips the prices fell before long. I scrapped my plans for PAL chips and started using EPLDs instead, and have never looked back since.

I normally do not design for, or use discrete logic. That is much too time consuming and require either high cost or large volume runs to recoup design costs (designers have to eat too). When appropriate, I do slip in the occasional SSI/MSI chip in a design - there is a 74ACT138 in the CPUZ180, but that is all. So if there is a ready made chip available, I use it.

A prototype circuit can be used for many different projects with little or no hardware changes. For example, the IDE engine was developed on an S100 card containing only the one chip. The Centronics I/F was developed on the same card but with a printer attached to the output socket rather than an IDE drive. I use windowed EPLDs for this as they can be erased and recycled as something else.

Actual development on the larger PLDs is done in similar fashion to general software - one module at a time. When a module is working you save it and start on the next. Eventually they are combined into a single program (chip). Most of the testing is done by a simulator.

In a tight fit (such as the PT IDE802) it takes considerable juggling to fit the final results into the available macrocells, especially as there are different types of cells in the '18xx. This is similar as to squeezing computer code into limited memory.

To convert a PLD design into discrete TTL (working) would take a lot more time and effort than to do the original design. The available functions differ completely. TTL has a large number of different macro functions while PLDs use basically sum of products and little else. There are of course registers and flip flops, but in most cases they are too limited in their function to use.

Similar, to convert an existing TTL design into a PLD is also difficult. The result is always much better when designed from ground up for either TTL or PLD. It is like writing a program in BASIC then converting into C or vice versa. It can be done, but it is not practical.

For that reason I do have a bone to pick with you about requesting equivalent TTL (assume working) circuits for any PLD design submitted. It can be done with small chips using only a few gates, but not with larger ones. For example: in the PT IDE100, each S100 DI pin (8 of them) uses a buffer, an 8-input OR gate driven by 8 AND gates having 26 inverted and 15 non-inverted inputs. That is 328 inputs with 208 inverters to 64 AND gates just for the DI bus! - and there are another 3 data busses.

Only one data bit was actually designed in that case , the rest were generated internally by a compiler. The result is that each bit has its own decoding circuitry. Such waste would never do in a TTL circuit, but is common practice with PLDs. It both reduces delays and makes timing calculations straight forward. That way a signal will only pass through a single macrocell.

Inside the PLD

First a caveat - I won't attempt to describe PLD architecture in detail. I (think I) can recall an earlier article in *TCJ* describing the basic macrocell and sum of products logic. I will assume the reader has some familiarity with this and with Boolean algebra in general. Detailed description of that is well beyond the scope of this article.

Suffice to say, the PLD contains a number of macrocells of similar (but in most chips not exactly same) construction. These can be thought of as discrete TTL chips. The cells are configured to their desired functions and are wired together to form the entire design. The output from a cell can go to a pin, or if 'buried' just to other cells. Some cells have more than one output. Some pins are input only while others are configurable as inputs or outputs or both.

The main part of the cell is the AND/OR (sum of products) array. In the EP18xx each of the 48 cells have 10x88-input AND gates driven from the pins and other cells. Half of the inputs are inverted. 8 of these gates (product lines) feed an 8-input OR gate which can also be configured as a NOR gate. One feeds the output enable for a buffer to a pin. The last feeds a reset input to a register that can be positioned between the OR

gate (sum) and buffer. The signal can be tapped at one or two points, brought to other cells, or back to its own cell as a feedback. If the register is used it can be clocked from one of the AND gates or a dedicated clock pin. It can be configured as a D, JK or T flip flop. The EP18xx has about 48,500 connections that can be on or off.

As the AND/OR array can be inverted at either/both ends, it can become a OR/AND array as per DeMorgans theorem. The array output can be brought back to an input, inverted if necessary, so transparent latches, RS flip flops can be constructed in the cell. These can take on some rather weird input schemes to suit your design. XOR gates are also constructed from the array.

The smallest PLDs (20 pin EP330 or GAL 16V8) contain 8 macrocells with 8 I/O pins and 10 input pins, with more limited configurations. As you can see, the whole thing is rather flexible.

Design Process

The design process begins the same way as a discrete TTL design. First the specifications (what should the design do?). Then a simplified block diagram. Each block is broken down into smaller blocks and finally into discrete Boolean equations. When satisfied, these equations are minimized, manually or by a logic minimization program. Next stage is to maximize the result into available functions. Here the discrete TTL and PLD design split.

To maximize for TTL you try to squeeze the design into the minimum number of chips (can-count) or you will be left with a large number of partly utilized ICs. This take considerable knowledge of available chips, each with different timing characteristics, so timing calculations become difficult. A signal normally has to pass through several chips before reaching a destination. Signals are tapped halfway through functions (to minimize can-count), while still more are tapped further down the line. If a desired signal is available somewhere, it is normally used rather then regenerated. This means a wide variation in delay paths which can easily lead to troubles. As several chips are used sequentially with delays specified as min/max the final delay can be difficult to calculate with any accuracy.

PLD design is much simpler. There are only a few functions available, making maximization easier. This is done by software that is aware of the chip's internal architecture. That program is probably the same that performed the minimization in the first place so you feed in your original equations and take out sum of products equations ready for a PLD macro cell.

Most of the time you enter Boolean functions, but sometimes it is easier to describe a function in state machine lingo. That looks even more similar to a computer program. You define all possible states and what events will cause change from one state to another. Such a description is more difficult to translate into TTL chips, but present little problem to the PLD compiler.

Yet another way to enter a PLD function is to use a truth table, but this becomes tedious with more than a few inputs. In most designs you use a combination of these methods, whichever is the easiest for the particular part you are working on. The compiler recognizes and differentiates between a Boolean equation, state machine code and a truthtable, and still works out a sum of products equation.

Multiple similar functions can be vectored to simplify design. This is like a FOR-NEXT loop i.e. design a function then order it repeated for a number of times. That is what I talked about earlier by designing one data bit in a bus, then letting the compiler take care of the rest.

Similar to computer programming, you have macros and subroutines rather then defining similar functions over again. Larger functions can reside in separate files that are brought in as needed, and you may also have a collection of library files with useful functions.

The final 'fitting' (to place these equations into the PLD and not run out of pins or macrocells) can cause considerable problems resulting in a lot of juggling before it all fits. Software to do this is aptly called a 'fitter'. Like a compiler/linker for computer programs they work well when there is plenty of space, but in a tight fit they cannot beat human intervention. Thus when the PLD compiler comes out with a 'Can't Fit' message you can either use a larger or several chips or do your own juggling. If the latter, you can still use much of the compiler's output, it may only need a little massaging.

As you can see, the whole process is much more like writing a software program in a high level language than designing hardware. You actually tend to forget about chips and gates. You use a normal text editor to enter your design, but you must of course learn a new syntax. This varies widely with the various languages available, but the principles are the same. You have PALASM, CUPL, ABEL etc. but an AND function is still an AND function even if it is entered as a different character. If you are familiar with CUPL you could still read and understand a PALASM file. They are not that different, nor can I say that one language is better than another, just a bit different.

Most compiler's understand a superset of the original language - new functions that make the design process easier. Similar to new functions appearing in computer programming languages.

In most design programs once you have the ASCII text file(s) containing your design, the minimize, maximize, compile and fit sequence can be performed without user intervention and will only halt on a fatal error. As in the more familiar compile and link process.

A PLD compiler generates a JEDEC file as its final output. You are also provided with other files that report on utilization,

simulation results, etc. The JEDEC file is what it is all about. That is equivalent to the COM or EXE file in a computer.

Design programs for PLDs normally come with a simulator, so you can test your design before committing it into hardware. The test files are frequently larger than the original design file. You specify as many input conditions you can come up with and feed them through the simulator. You can then view the resulting output as if it had come from the chip. Most bugs are caught that way, but as with all programs, it is only as good as your input file. Fortunately these simulators accept a variety of input schemes (such as FOR-NEXT loops) so you don't have to set every input pin to every possible combination. On a complex design, writing and viewing simulation data is time-consuming, but still beats doing it in the real world. You can also access points internal to the chip that may be impossible without a simulator.

There are many PLD design programs on the market, from a few hundred to many thousands of dollars. The more expensive ones also accepts TTL designs in graphic form and convert them for PLDs. For most people this would be an overkill. As long as the software contains the functions I have described, and it supports a reasonable assortment of PLDs you should be able to design them efficiently.

The PLD compiler needs to know the architecture of the chip it is working with, but does not need to know it intimately. That is very important to understand. Over time the PLD chips do change: new revisions come out, old ones are discontinued. As long as the basic architecture remains the same the compiler will still work as the JEDEC file requirement is basically unchanged for a particular family of chips. Unless something entirely new comes out, your compiler will still be usable for years to come. As an example we have talked about the EP18xx chips. EP1800 was the original version and has been discontinued. EP1810 supersedes it and EP1830 is a later parallel version, quite different internally, yet they are all JEDEC code compatible. As code assembled for an 8080 can still be used for the Z80 or even Z380 chip.

One program I have used, worth mentioning here, is Intel's PLDSHELL. It is low cost and suitable for beginners, but still quite powerful. It has a list price in the $200 region, but I have seen it discounted here (Australia) for well under $100. Yet it covers all the essentials including the simulator, and covers a variety of chips. Runs on MS DOS as a shell (as the name implies), and comes with a well written manual and lots of design examples. Definitely recommended. It does not approach the more professional programs as ALTERA's but then it's not kilobucks either. It is PALASM compatible.

Hardware - the difficult part

The real problem for the beginner and the occasional PLD designer is the hardware. Not just the machine itself which is complex enough, but the software that runs it too. Especially if it is to support a variety of chips.

The machine accept's JEDEC file data and will burn the chip from that. This file is basically a collection of ASCII 0 or 1 but lots of them. The position in the file denotes the function, with 0 for on / 1 for off. The programmer will translate these into the correct voltages and timing pulses to the correct pins on the chip. Some fancier programmers can also exercise the programmed chip to check that it actually performs as expected. It takes only a second to program and verify a small chip, and up to 5 seconds for the largest.

The real problem is that most chips undergo some revision every couple of years. The more different chips you use the more revisions you have to contend with. Frequently a new revision will require programmer software changes to cope with it. As the JEDEC code stays the same, it may refer to different locations in the new chip, and would produce an entirely different result if the machine was not set up with software for the new revision. The voltages and timing specs tend to alter with new revisions, and if not heeded, may cause insufficient programming (the chip will forget what it was supposed to do after some time).

In many cases the revision is not marked on the chip but must be determined by the machine, and the programming algorithm adjusted accordingly. To confuse matters, I will give you an example: the EP1810 is marked EP1810. The EP1830 also marked EP1810, but is completely different both soft and hardware wise. A speed rating of 35 or more is an 1810 while 30 or less is an 1830. If the machine can't distinguish between the two versions it will mis-program the chip (destroy it). And if your programmer only supports the EP1800 or 5C180 it will actually program a chip marked EP1810 with a speed rating >30ns (at least in some fashion, data retention may not be up to scratch). It will however mis-program an EP1810 with a speed rating <30ns. The same problem also applies to other chips. Sometimes I wonder why these complications.

The upshot of all this is if you bought your programmer and the company went out of business later, you only have an expensive door stop when the next revision of the chips come out. Similar with the cheap second hand machine. The chip manufacturers can often supply programmers, somewhat expensive and may only be set up for their own chips, but at least reliable. The Intel PLDSHELL program mentioned does interface with an Intel programmer, but I have no idea about its cost, but I believe it is very versatile. Other manufactures do recommend third party machines that can normally be relied on. Just remember that it is imperative that you are supplied with software updates and that your vendor/manufacturer can keep up with them. Caveat emptor.

Note however that as the JEDEC file remains the same, your old design will still program a new chip version correctly, so your files remain current. It is only the software that runs the programming machine itself that is changed.

Here at Palmtech I use a programmer designed in house and I request programming specs from the chip manufacturers when

new revisions come out. As soon as the machine sees a chip and does not recognize the internal rev data it objects. That happened to the last lot of EP330's I got. This method would probably not suit the majority of readers.

You could of course buy a programmer that handles only a couple of similar chip types, 20-24 pin GALs for example would probably be the cheapest option. But that is a serious limitation considering the plethora of PLDs on the market.

There is another solution to the problem: Most supply houses will program your chips for you for a modest fee (probably around a dollar, or less for quantity) if you supply them with your JEDEC file. With the simulator, you can be fairly sure your design will work, and you won't be stuck with an obsolete machine or pay kilo-bucks for one that has guaranteed support. And, as I said before, a reasonable design program won't set you back too far, and will stay current for a long time. So, by buying pre-programmed chips you can get into PLDs and also be able to take advantage of the full range available.

Which PLD family?

The next question is 'What type of PLD to use?'. There are two main types: EPLDs and GALs. They are both CMOS and are roughly similar in architecture but quite different electrically. Either type is available from several sources, often under different part numbers i.e. EP610 = iPLD610 = 85C060 = PALC610. The two types or not code, pin, or even functionally compatible.

The EPLD was originated by ALTERA and comprises of EPROM cells. They come in windowed ceramic packs that are ultra violet light erasable for trial and error, or plastic one-time programmable which are much cheaper. Electrically, I/O is similar to the 74ACT chips.

GALs came originally from LATTICE and uses EEPROM cells, that is they can be erased and reprogrammed in the machine (some can even be programmed by a serial interface obviating the need for a programmer). They are therefore much easier to use for developing purposes. Their I/O is closer to the 74HCT series.

Some other pros and cons are:

* GALs are rather power hungry. The smallest 20-pin GAL (16V8) draws 75mA compared to 40mA for a comparable EP330 (the 68-pin EP1810 EPLD draws only 100mA). EPLDs are also available with a standby current consumption of a few micro Amps. There are low-power GALs available with power consumption comparable to EPLDs but they are a lot slower.
* GALS provide considerable less drive current at the output pins and are asymmetric. The GAL 16V8 will source 3 mA at TTL levels, compared to 20mA for the EP330. That makes GAL devices unsuitable for bus-drivers.
* Data retention (how long does the device remember its function) is specified at 10 or 20 years in GALs, but this is reduced at elevated temperatures. EPLD manufacturers talk about >100,000 years. This makes GALS shaky for some commercial applications, as you sometimes don't know at what temperature your design will be used at.
* GALs are about 25% faster then EPLDs in their respective top speed versions.
* Pricing is similar between comparable devices.
* EPLDs are more demanding on programming hardware, and are non-erasable or must be erased externally.
* Both types are available in many different sizes and packages, with the EPLD having a considerable greater variety of architectures. Packages come from 20 to 250 pin.

In short, GALs are easier to program for the beginner but place severe restrictions on the design. For example, there is no suitable GAL device for the PT IDE100 or 802. The nearest is the 84 pin pLSI1032 but it cannot drive the S100 bus and drive current would be marginal for a Centronics port. It does contain a lot more internal functions but at more than twice the cost of the EP1810.

A nice feature of the later EPLDs is that the programming margin (how well the cell is programmed or erased, not just if a cell is on or off) can be determined by the machine. That way you are certain the chip will remember its tasks long term.

There are a couple of other types worth mentioning: PALs were the original programmable devices, but are becoming obsolete. They are bipolar technology and rely on fusible links for programming, same as PROM memory chips. They are only available in the smaller packages, and require special programming hardware. They are very low cost, and most are pin and JEDEC code compatible with GALs. It is common to replace GALs with PALs in a production version to reduce cost.

At the other end there are FLEX PLDs (Static RAM cells) that are loaded from an external EPROM or such at power on. They are only available in larger versions and are rather expensive, and I have never tried them. Maybe one day when the need arises. And of course, they don't need a programmer, but you have to include the EPROM in your circuit.

To gain further insight into the PLD chips I suggest you obtain some data books from the various manufacturers: ALTERA, AMD, CYPRESS, INTEL, LATTICE, NSC, PHILIPS, SGS-THOMSON.

I hope this has given the reader some food for thought. But if you do designs I don't think you can afford not to go in for PLDs. As with anything else, start with the smaller devices to familiarize yourself. They are inexpensive so no real harm if you damage a few. If you go in for EPLDs, get a few windowed devices and an EPROM eraser to experiment with, then use plastic chips for actual work.

TCJ Center Fold

Special Feature

All Users

Rockwell R65F11

This center fold was prompted by the Embedded report and hopes that some interest in this product might stir Rockwell to make some of the chips avaialable. I fear however that all production and documentation of this device has long since been lost.

I have had this development package since shortly after it was produced. When Allen Bradley purchased Rockwell I had hopes of seeing this device used in place of the Basic modules which have become so popular in PLC systems (industrial controllers). I did see some products using it in the late '80s at a WesCon show. There were two vendors of small embedded systems selling boards with the F11 chip. I look back now and see that these vendors were proably a few years ahead of their time. Products like this now sell well, while back then were just starting to be considered for use.

The major feature of this implemetation is the use of a disk controller and thus disk drive to hold your program. Most embedded system relie on the Host development platform for this use. As you will see in later articles, there are advantages and handycaps for either concept. I used this platform with drive many times for testing and learning. I found it adequate but problemmatic. A friend tried it and became a NON-Forth user. There are a number of bugs that an inexperienced user will not understand. These bugs will just frustrate you, since your inexperience will not give you the skill to understand the bug is in the ROM and not your code. This is unfortunate but a very common situatuion with beginners and complex systems.

There is nothing special about the design, in fact it is a very good example of a minimal system. The use of ROM for address decoding was very popular until the use of PALs replaced them. Few of the newer ROM programmers can program these long forgotten devices. The use of the disk controller and corresponding code in the ROM actually added little to the design. The design shows how any device can be added to a basic design. Typical additions in a minmal design like this might be for more serial or parallel ports. The board does come with a printer port, making it totally self contained. The use of a terminal is all that is needed to do development.

SOFTWARE CONSIDERATIONS

When the R65F11 is reset due to power up or reset, several variables assume default values which may need to be changed by the user. One of these is the top of memory pointer used by the disk interface. Reset assumes the presence of 8K of RAM from $0300–$1FFF. In the example given here, only 6K of memory is available ($0300-$17FF). In order for the disk buffers to work, the top of the memory must be set to 6K. The following commands will perform this function:

HEX 1800 MEMTOP DECIMAL [RETURN]

The R65F11 has been designed with vectored I/O so that user written I/O routines may be used as an alternative to the internal ones. The console input and output is vectored through the locations UKEY ($0044) and UEMIT ($0046) and the disk is vectored through UR/W ($30A). As an example of using these vectors, Listing 1 shows a method of patching UEMIT to direct the console output to a parallel printer port. This listing also shows a method of using the headerless code generation to create words and then forget the heads (only) of those words which will not be used again.

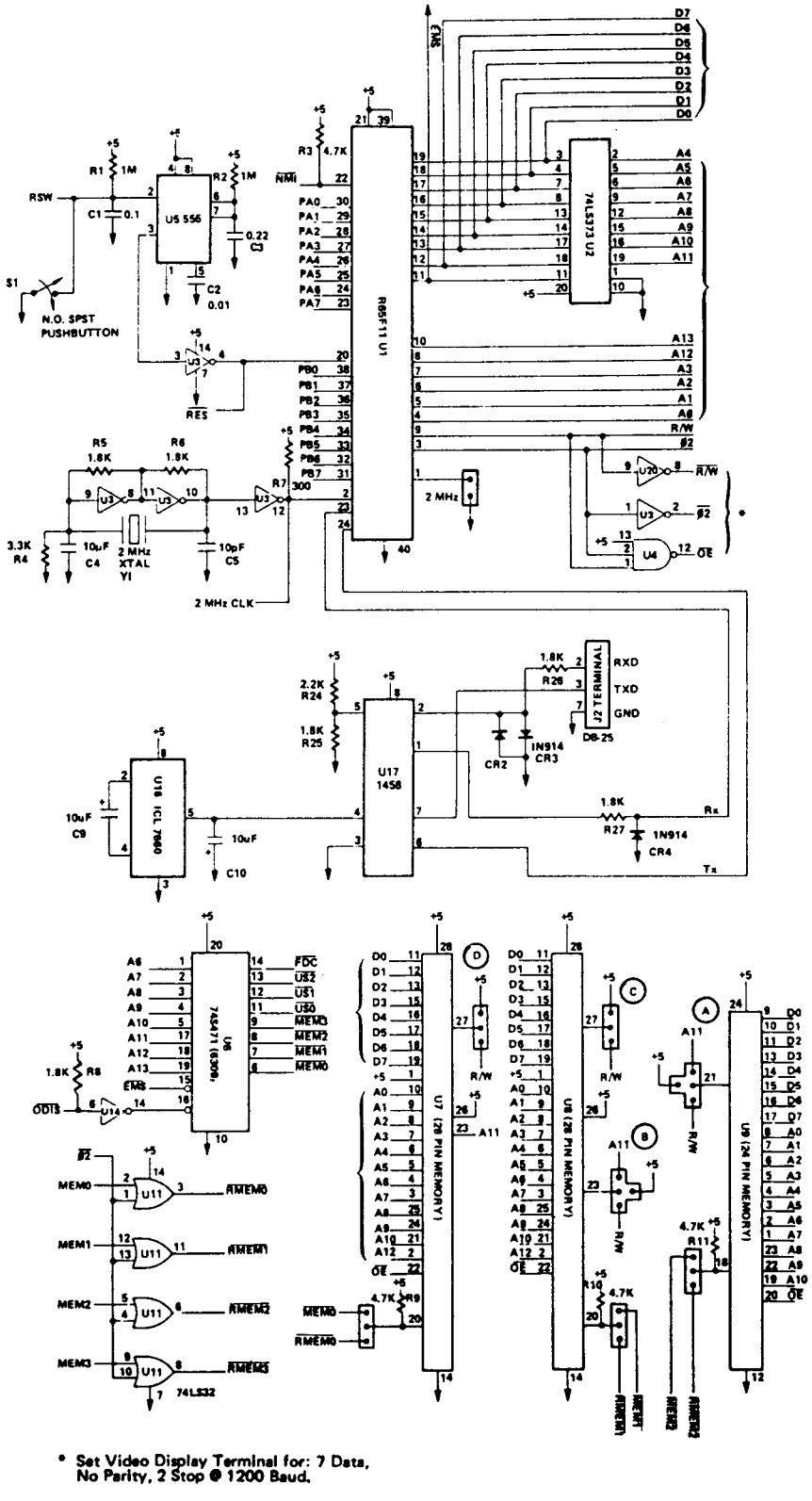
Listing 1. Implementing a Printer Driver

```
SCR #10
0   ( CENTRONICS DRIVER FOR RSC FORTH DEVELOPMENT
    BOARD )
1   ( FILTER OUT FORM FEEDS ) BASE @ HEX
2   LATEST DP @ 100 + H/C
3   CODE STB-PRT 10 3 RMB, 0 2 RMB, 0 2 SMB, RTS, END-CODE
4
5   CODE ACK-WT BEGIN, 11 3 BITSET UNTIL, RTS, END-CODE
6
7   CODE HOUT 1 STA, OA # CMP, O = NOT IF,  ' STB-PRT CFA
    @  JSR,
8    ' ACK-WT CFA @  JSR, THEN, RTS, END-CODE
9   0 HEADERLESS !
10  : P-ON [ ' HOUT CFA @ ] LITERAL 0046 ! ;
11
12  : P-OFF F5EF 0046 ! ;
13  : FORM-FEED OC EMIT ;
14  ' P-ON LFA ! BASE !
15  ;S
```

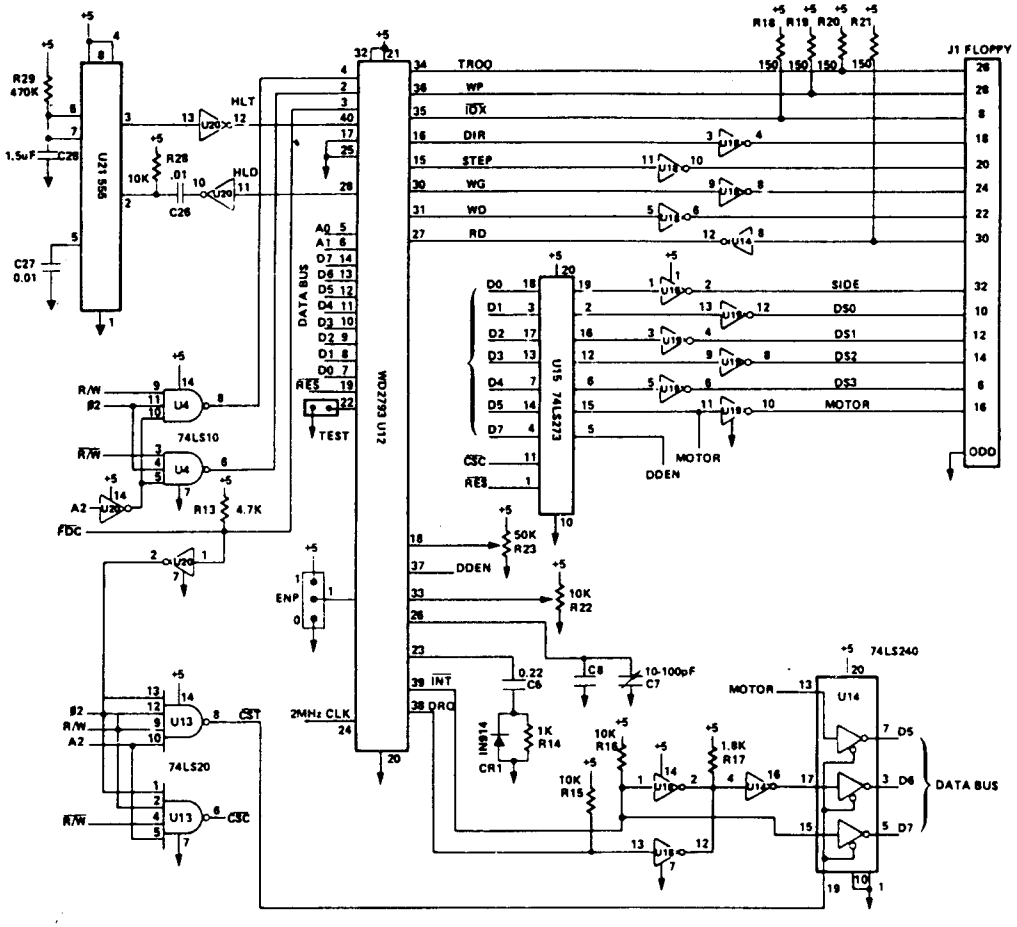
R65F11 • R65F12
Application Note

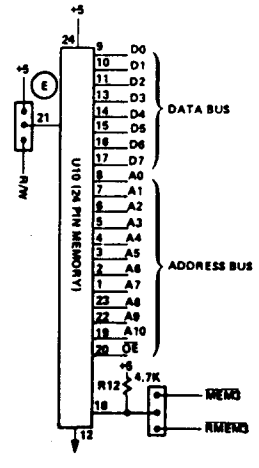
A LOW-COST DEVELOPMENT MODULE FOR THE R65F11 FORTH MICROCOMPUTER

by Joseph W. Hance
Rockwell International, Semiconductor Products Division, Newport Beach, California



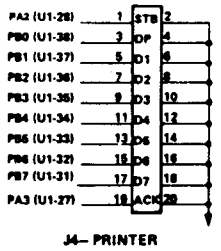
* Set Video Display Terminal for: 7 Data, No Parity, 2 Stop @ 1200 Baud.

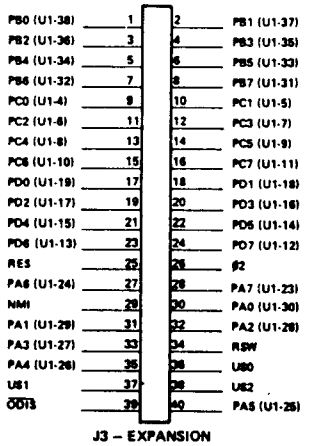


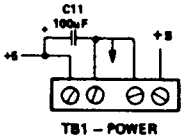


0.1 uF BYPASS CAPACITORS

C12
C13
C14
C15
C16
C17
C18







Application Note Order No. 2162
Rev. 1, October, 1983
Document No. 29651N65

Table 2. Jumper Selection

Socket	Part Type	Jumper Position at Pin
U7		p27 p20
	8K x 8 RAM	R/W $\overline{\text{RMEMO}}$
	2764 PROM	+5 $\overline{\text{MEMO}}$
	2732 PROM	+5 $\overline{\text{MEMO}}$
U8		p27 p23 p20
	8K x 8 RAM	R/W A11 $\overline{\text{RMEM1}}$
	2K x 8 RAM	R/W R/W $\overline{\text{RMEM1}}$
	2764 PROM	+5 A11 $\overline{\text{MEM1}}$
	2732 PROM	+5 A11 $\overline{\text{MEM1}}$
	2716 PROM	+5 +5 $\overline{\text{MEM1}}$
U9		p21 p18
	2K x 8 RAM	R/W $\overline{\text{RMEM2}}$
	2732 PROM	A11 $\overline{\text{MEM2}}$
	2716 PROM	+5 $\overline{\text{MEM2}}$
U10		p21 p18
	2K x 8 RAM	R/W $\overline{\text{RMEM3}}$
	½ 2732 PROM	+5 $\overline{\text{MEM3}}$
	2716 PROM	+5 $\overline{\text{MEM3}}$

U1 R65F11

For 2 MHz device (R65F11AP), install jumper at Pin 1 to Ground; and set VDT for 2400 Baud.

U12 WD2793

For test, install jumper at Pin 22 to Ground.
Enable (1) or Disable (0) precompensation at Pin 1 per disk drive vendor recommendation.

Note:

For location of jumpers see schematic, silkscreen, or board artwork.

Table 1. Decode PROM (U6) Coding

The following configuration is assumed:

Funct	Address Space	Code	Signal	Socket
FDC	$0100–$013F	7F	FDC	U12
RAM1	$0200–$07FF	F7	MEM3	U10
RAM2	$0800–$0FFF	FB	MEM2	U9
RAM3	$1000–$17FF	FD	MEM1	U8
ROM	$2000–$3FFF	FE	MEMO	U7
Not Used	$1800–$1FFF	FF	No Chip Selected	
Page Zero	$0000–$00FF	FF	No Chip Selected	
Page One (ex FDC)	$0140–$01FF	FF	No Chip Selected	

msb/lsb	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	FF	FF	FF	FF	7F	FF	FF	FF	F7	F7	F7	F7	F7	F7	F7	F7
1	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7	F7
2	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB
3	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB	FB
4	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD
5	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD	FD
6	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
7	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
8	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
9	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
A	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
B	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
C	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
D	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
E	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE
F	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE	FE

Parts List

Description	Part Number (Vendor)	Description	Part Number (Vendor)
R65F11 Development Board			
U1	R65F11 (Rockwell)	R4	3.3 K Ohm
U2	74LS373 (4)	R5, R6, R8, R17, R25–R27	1.8 K Ohm
U3, U20	74LS04 (4)	R7	300 Ohm
U4	74LS10 (4)	R14	1.0 K Ohm
U5, U21	555	R15, R16, R28	10 K Ohm
U6	74S471 (6309) (4)	R18–R21	150 Ohm (½W)
U7	R65FR1 (Rockwell)	R22	10 K Ohm POT (Bourns 3009P-1-103)
U8	28-Pin Memory	R23	50 K Ohm POT (Bourns 3009-1-503)
U9	24-Pin Memory	R24	2.2 K Ohm
U10	24-Pin Memory	R29	470 K Ohm
U11	74LS32 (4)	CR1-CR4	1N914 or eq
U12	WD2793 (Western Digital)	C1, C12–C19	0.1 μF
U13	74LS20 (4)	C2, C26, C27	0.01 μF
U14	74LS240 (4)	C3, C6	0.22 μF
U15	74LS273 (4)	C4, C5	10 pF
U16	ICL7660 (Intersil)	C7	10–100 pF variable (Sprague GKF70000)
U17	LM1458		Optional
U18, U19	7406	C8	10 μF electrolytic
J1	34 pin (3M #3431-2002)	C9, C10	100 μF electrolytic
J2	DB25 (AMP 2065-84-2)	C11	Not Used
J3,	40 pin (3M #3432-2002)	C19–25	1.5 μF
J4	20 pin (3M #3428-2002)	C28	N.O. Pushbutton (Chicago Switch EIA50)
TB1	0.2" center terminal strip (Buchanan #SSB404)	S1	2.000 MHz Crystal
R1, R2	1.0 M Ohm	Y1	
R3, R9–R13	4.7 K Ohm		
R65F12 Adaptor Board			
U22	R65F12 (Rockwell)	C12	100 μF electrolytic
U23	Wire Wrap Pins (40 ea.)	J6	34 pin (3M #3431-2002)

Notes:

1. All resistors are ¼ W, 10% carbon unless otherwise noted.
2. All capacitors are 16 V ceramic disks unless otherwise noted.
3. All ICs should be socketed: 40, 28, and 24 pin — 2 each; 20 and 8 pin — 4 each; 14 pin — 7 each.
4. 74LS devices should be purchased to Texas Instruments specifications or equivalent.

SUPPORT GROUPS FOR THE CLASSICS

Regular Feature

Group Reviews

Trenton Zed-Fest Announcement

I am happy to announce that there will be a Zed-Fest 95 during the weekend of the Trenton Computer Festival. The official announcement follows. Please help by spreading it around wherever you can — to bulletin boards, CompuServe, GEnie, user groups, newsletters, etc.

The Trenton Zed-Fest 1995

The Zed-Fest in connection with the 1995 Trenton Computer Festival (TCF) will take place again this year at the Stage Depot Motel over the weekend of April 21-23. The motel is undergoing extensive changes since new management took over just before last year's Zed-Fest, so we don't know for sure which meeting rooms we will have, but we have been promised a room for both Friday and Saturday evenings for informal get-together. No technical presentations are planned, but I am hoping to conduct a live GEnie Real Time Conference (RTC) session right from the meeting rooms.

I do not yet have an extensive list of those attending, but I can report the very exciting news that Helmut Jungkunz will be coming all the way from Munich, Germany, to be with us. I hope this will be a big extra incentive for you to join us as well.

There will again be an "All-Day CP/M Conference" on Saturday as part of the Trenton Computer Festival at Mercer County Community College. Hal Bower is the only presenter I know of who is scheduled at the moment. He will be describing Cheap-LAN, a simple local area network.

After the TCF session winds down on Saturday, we expect to adjourn to a local pizza parlor to stage the annual Anchovy-Pizza-Eating Contest. Lee Bradley and I could definitely use some competition this year. There's still plenty of time before the meeting, so start training now!

Lodging Information

The Stage Depot Motel provides not only convenient but also low-cost lodging. Things get busy in the area around the time of the Trenton Computer Festival, so I urge you to make reservations just as soon as you can.

Here is the information about the motel. Please don't forget to mention that you are with the Zed-Fest when you place your reservations. If enough of us register, we will not have to pay for the meeting rooms. Also note that, at least in past years, the registration lobby closed at 11 PM. If you will be arriving later than that, special arrangements should be made (for example, have one of us pick up your keys for you). The room rates below are tentative and do not include taxes. Because of the renovations, the rates for some rooms may increase a few dollars. Ask the management when you place your reservation.

Stage Depot Motel
Route 31
Pennington, NJ 08534
609-466-2000
800-93 STAGE
(800-937-8243)

larger rooms (2 double beds)		
	1 person	$44
	2 people	47
smaller rooms (1 dbl, 1 sgl bed)		
	1 person	$37
	2 people	$44

for suites or other arrangements, inquire with the management

At a later time I will post a message with an ASCII map showing how to find the motel and the TCF site. For those who wish to try locating them on a road map, the motel is on Route 31 north of the city of Trenton and, I would guess, about 5 miles north of I295. Mercer County Community College is on Route 535 east of the city of Trenton and east of I295, in or near the town of Mercerville.

If you are planning to come, please let me know by any of the following means:

Internet mail to:	SAGE@LL.MIT.EDU
GEnie mail to:	JAY.SAGE
a message on my Z-Node:	617-965-7046 (v.32bis)
	617-965-7785 (v.fast)
	617-965-7259 (2400 bps)
a FAX to me at work:	617-981-5328
real postal mail to:	1435 Centre Street
	Newton, MA 02159-2469

Hope to see lots of you there!
Jay Sage

TCJ Staff Contacts

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

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

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

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

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

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

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

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

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

Tilmann Reh, Germany, E-mail: tilmann.reh@hrz.uni-siegen.d400.de. Has many programs for CP/M+ and is active with Z180/280 ECB bus/Modular/Embedded computers. Microcontrollers (8051).

Helmut Jungkunz, Munich, Germany, ZNODE #51, 8N1, 300-14.4, +49.89.961 45 75, or CompuServe 100024,1545.

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

USER GROUPS

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

Sacramento Microcomputer Users Group, PO Box 161513, Sacramento, CA 95816-1513, BBS: (916)372-3646. Publishes newsletter, $15.00 membership, meetings 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.

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.

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

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

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

The Pacific Northwest Heath Users Group, contact Jim Moore, PO Box 9223, Seattle, WA 98109-0223.

The SNO-KING Kaypro User Group, contact Donald Anderson, 13227 2nd Ave South, Burien, WA 98168-2637.

SeaFOG (Seattle FOG User's Group, Formerly Osborne Users Group) PO Box 12214, Seattle, WA 98102-0214.

OTHER PUBLICATIONS

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

The Analytical Engine, by the Computer History Association of

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

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

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

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

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

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

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

Historically Brewed, A publication of the Historical Computer Society. Bimonthly at $18 a year. HCS, 2962 Park Street #1, Jacksonville, FL 32205. Editor David Greelish. Computer History and more.

IQLR (International QL Report), contact Bob Dyl, 15 Kilburn Ct. Newport, RI 02840. Subscription is $20 per year.

QL Hacker's Journal (QHJ), Timothy Swenson, 5615 Botkins Rd., Huber Heights, OH 45424, (513) 233-2178, sent mail & E-mail, swensotc@ss2.sews.wpafb.af.mil. Free to programmers of QL's.

Update Magazine, PO Box 1095, Peru, IN 46970, Subs $18 per year, supports Sinclair, Timex, and Cambridge computers.

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

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

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

AAA Chicago Computers, Jerry Koppel, (708)681-3782. SS-50 6809 boards and systems. Very limited quanity, call for information.

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

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

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

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

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

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

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

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

DYNACOMP, 178 Phillips Rd. Webster, NY 14580, (800)828-6772. Supplying versions of CP/M, TRS80, Apple, CoCo, Atari, PC/XT, software for older 8/16 bit systems. Call for older catalog.

32-Bit Systems

All Readers

Language Topics

Real Computing

By Rick Rodman

Programming languages

I've said before that object-oriented programming is merely a notational convention to enforce modularity. Let me add to this the following principle: Programming is the notation of an algorithm for two purposes - first, to communicate an algorithm to a machine, and second, to communicate an algorithm to a human reader. And it is this **second** function which is most important.

The sole convention for evaluating a language processor is its efficiency in communicating an algorithm to a machine. The sole convention for evaluating a language **itself** is its efficiency in communicating the algorithm to a human reader.

We can point to several programming languages each of which is based on a single unifying concept. If you can grasp the underlying concept, learning the language is simple; if not, it's impossible. I'll refer to these as "pure" languages.

Well-known are Forth, whose underlying concept is the stack, and Lisp, whose underlying concept is the list, as used in something called a "lambda arithmetic". Smalltalk, with its opaque syntax, is based on a message model. APL is based on an array model. When using a pure language, you have to formulate your problems into its underlying model (some might use the verb "shoehorn"), then use tools of the language to solve your problem. In a pure language, language processing can be very efficient - I once saw a laptop, made in the mid-1970s, with an APL interpreter running on an 8008.

By contrast, our more "conventional" languages, such as Pascal and C, offer a variety of data structures combined with a variety of functions and features offered with no single organizing model. They can be viewed as being composed of a number of "sublanguages" - for example, the stream I/O sublanguage, the printf/scanf sublanguage, and so on. Each of the unrelated sublanguages must be learned to become proficient in the language itself. The sublanguages are not independent, but "layered" in semi-overlapping ways; so, the ways in which the layers interact must be learned as well. I refer to these languages as "layered" languages.

The king of layered languages is, of course, PL/I. In PL/I the same verbs, such as SUBSTR, addition, etc., are used in each of the widely disparate sublanguages, such as that pertaining to bit strings. This confusing technique became known as "operator overloading".

Now consider C++, in which a thick crust of Byzantine notation has been layered atop C. To become proficient in C++ is to learn, not only this new layer, but to peek into each of its many interactions and interdependencies with all of the foregoing sublanguages beneath.

Since one has to learn a language either to write it or to read it, the increasingly complex languages used in programming today are exacting an ever higher cost in human energy while doing an ever poorer job at communicating algorithms. I like C, and program in it all the time - in fact, I like a lot of the ANSI extensions - but C++ is just too complicated. What we really need is C—: something like Small-C, but with structures and efficient code generation.

I disagree with Ed Yourdon when he says that 20% of a programmer's skills become obsolete each year. The core concepts are actually not changing at all, yet hard-learned lessons of the past are discarded like flinging away last year's fashions. Take a look at the sample code Microsoft delivers with their compilers. What do you see? Global variables laying around everywhere. Vague, nondescriptive or even "cutesy" variable and function names. Most damning: Poor commenting - or none at all. These people have not learned the lessons of the last 50 years of programming. And they're the very same people who tell you that C++ will make your code better! All of this, by way of introducing you to...

Linda

In SQL, a new technique was developed for explicitly layering a sublanguage within a language, by means of a precompiler. Some computer scientists at Yale University have invented a language called Linda for distributed programming. Rather than being a special new language, like Occam, they've described a few functions which make up Linda as a layered sublanguage. This is smart, because Occam failed precisely because it was a new language.

Linda is intended to hide all of the implementation details of parallel programming, allowing the programmer to concentrate on his application. It consists of only six statements which manipulate a bunch of shared data objects called

"tuples". What is a tuple? It is a data object with a publicly known structure and name. All of these tuples are stored in an amorphous, undefined area called "tuple space". (Obviously this space must be somewhere, but you don't need to be concerned about it.)

The six Linda statements are: OUT, to put a tuple into the space; IN, to wait for a matching tuple to show up in the space, and delete it from the space when it's found; RD, which is similar but copies without deleting; INP and RDP, which don't wait but only 'poll' the space; and EVAL, which puts a unit of work into the space for someone to perform.

Linda is a pure language with a simple model: the 'single shared tuple space.' It layers itself atop whatever other language you like to use (C, Pascal, etc.); you use a preprocessor to process the Linda statements, followed by a compiler for your base language. Linda's big pitfall is, in fact, that 'single shared tuple space'. Implicit in it (and anywhere you see the word 'single') is a master/slave situation, and the series of attendant problems that always go with it (starting with the infamous "server down").

More down to earth than Linda, and hence more visibly complicated, is RPC API (Remote Procedure Call). Rather than a large public "tuple space", RPC is designed around cross-network transactions between specific entities. While it appears daunting, it's actually much cleaner than programming directly at the socket level. A master/slave architecture is just a special case of a more general peer-to-peer design.

Distributed computing

Thus having deftly (?) moved from programming languages to network programming, I'll now deftly move on to distributed computing. Now that so-called "client/server computing" seems to be the most accepted way to do things, people have started to notice that it has a number of drawbacks. In the classic client/server model, a client processor handles the user interface, and a server does the data manipulation. While the primary problem of client/server is the waywardness of the client, the iceberg looming below is the good old master/slave problem again.

Novell Netware is a classic master/slave design: If the server goes down, all computing ceases. You can use redundant disk arrays; you can put in fault-tolerant power supplies; you can use error-correcting memory; you can have environmental controls for temperature and humidity; you can have multiple CPUs processing the same instructions and checking each other's results; you can do all of these and more - people have! Yet, inevitably, the server will go down. The server **must** go down.

To benefit from this lesson, we need to design our distributed computing systems in such a manner that tasks are redundantly distributed among a committee of peers; ideally, transparently so, so that, should any machine go down, another can pick up its duties with little disruption. For example, the Internet can survive any machine going down or coming up, simply because its design does not assume that any particular machine must be up. System design properly starts with the system as functioning, then secondarily addresses initialization issues and, thirdly, deinitialization issues.

"Well, that's a lot of generalities," you say. "Let's see some specifics." I'm afraid specifics will have to wait for next issue, as I'm running short of space.

Submarine patents

Unisys Corp., present owner of a patent on Terry Welch's "W" contribution to the commonly-used Lempel-Ziv-Welch (LZW) compression algorithm, has begun to demand royalties on use of that patent. Compuserve has been receiving most of the community approbation, because their GIF image format is the most widely known use of LZW, but they are not the bad guys.

The bad guys are selfish, greedy people who exploit the dull stupidity of U. S. Patent Office workers to make end-runs around the law and get patents on mathematical algorithms.

Some people cry: "Foul! How can they start demanding money for use of a patent which they have let people freely use for almost 30 years?" They can. Unlike copyright law, patent law is tailor-made for such abuses, and the so-called "submarine patent" is a typical scam. And individual inventors virtually **never** benefit from this lousy system.

The response of the International JPEG Group is typical of most public-spirited programmers: Remove GIF and/or LZW support from their software. Those of you who have disks full of GIF pictures, convert them to another format while you still can. Compuserve is working on a new format which will probably use JPEG compression.

So who benefits from this? As in nearly every case in which the legal system impinges on humanity, humanity loses.

Next time

We shrug our shoulders and get back to work on the Littlenet circuit and software, and try to make more than one computer do something useful. Winter's the best season for hobby computing. Hope to get snowed in!

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

Special Feature

68HC11,Support

Forth Based Tools

Small Tools

By Calvin McCarthy

Development Tools for the F68HC11 Microcontroller

A modern microcontroller contains a microprocessor surrounded by a collection of interface capabilities within a single package. When a high level language is added to this foundation the result can be very easy to use. As an example the Motorola M68HC11 provides a modified version of the M6809 processor with digital I/O ports, A/D converters, a timer, easy to use interrupts, and synchronous and asynchronous serial ports. Add a high level language like Forth and you gain a monitor which provides full access to all chip capabilities with the power of programmability.

I am a Forth affectionado and a Motorola microprocessor enthusiast. When I saw the advertisement for the New Micros Inc. F68HC11 with its embedded Max-Forth I coveted it as a great solution for some of my embedded controller ideas. On purchasing a F68HC11 based single board computer I discovered, to my delight, that it was even more professional and easy to use than I had expected. This has been confirmed as I have continued to work with it and learn more about its secrets.

You only need an RS232 interface and a 9600 baud terminal program to start using the New Micros Inc. single board computer. This will allow you to immediately execute the embedded Forth words from the keyboard and allow you to create colon definitions and add them to the dictionary. On trying this you will immediately recognize the limitations in this direct method of creating words and will see the advantages of using a PC or equivalent as a development platform. The PC can provide an editor, disk storage, and code download capabilities.

Your code development sequence could be something like this. Write the code in your favorite text editor and save it to disk. Leave the editor, enter the terminal, then download the code. While using the terminal, exercise the words downloaded, discovering their elegance or their bugs. Leave the terminal and return to the editor to continue coding. This sequence is inevitable when working with simple tools, although the switch between terminal and editor can be tedious.

The F68HC11 manual says that the terminal program used to download Forth code text files must have a wait-for-echoed-character capability and a wait-at-end-of-line capability. As each character is sent it is echoed, then at the end of each line, when the CR is sent, the terminal must wait for the returned Line Feed before it begins to send the next line of code. I did not have such a terminal when I received the NMI single board computer so I was stuck. What to do? Forth was available on my Atari ST so I cobbled together a one block terminal program. Another two blocks of code and I could download blocks of Forth code created in the Atari Forth block editor. I was winning. Now I had the terminal, could save the code, and download to the F68HC11.

Time passed and I bought a PC. I put out a request on the Amateur Radio packet network for an implementation of Forth for this new machine. Brad Rodriguez replied and generously sent me a copy of Frank Sergeant's Pygmy Forth. On trying it I was overjoyed. The source code was there, it had an easy to use editor, and there was lots of extra functions available. Now how should I use it? My first priority was to port my F68HC11 tools over to Pygmy Forth. Thanks to Brad Rodriguez serial port code found in the BRADTOOL.SCR file in the Pygtools distribution files the attempt was successful.

The tools I created for F68HC11 software development have made my code, download, test, code cycles almost painless (Of course, it will never be absolutely painless). They provide the following functions:

1. Terminal program
2. Forth block code editor in Pygmy Forth
3. Forth code download from Pygmy blocks
4. Forth code download from ASCII text files
5. Binary code download using the M68HC11 bootstrap mode
6. Download of Motorola format S19 encoded binary files to RAM or EEPROM

All of these functions are available without leaving the Pygmy Forth environment. The development cycle goes something like this. You create the Max-Forth code in the Pygmy editor. Hitting ESC takes you out of the editor to the Pygmy Forth "ok" prompt. You execute the word "T9600" to start the terminal and a CR will bring up the "OK" prompt of Max-Forth. You can now exercise the F68HC11. Hitting ESC returns you to Pygmy Forth where you will again see the "ok" prompt. Enter the editor again or do anything else available in Pygmy. Download blocks of code by executing the word ">SBC". You will be prompted for the block numbers then the download will begin. The code will be echoed so you can watch (sometimes

I do something wrong and the download fails. It would never happen to you). With the download finished you are left in the terminal function at the "OK" prompt where you can try out the code you have just added. There is no awkward closing the editor to switch to the terminal application.

Using the Tools:

The tools are found in the executable Pygmy Forth program T9600.COM and my source code accompanying this article. My tools use COM1.

T9600

1. Type "T9600" to use the terminal program.
2. Type CR - the Max-Forth "OK" prompt will appear. You continue as though you are using any other terminal program.
3. Hit ESC to return to the Pygmy "ok" prompt.

>SBC

1. Type ">SBC" to download Forth code you have in open files in Pygmy.
2. Enter start block number.
3. Enter end block number. The download will proceed.
4. (Optional) Hit ESC to interrupt the download and return to the Pygmy "ok" prompt.
5. Hit CR to see the Max-Forth "OK" prompt. You are in the terminal program. You can exercise the added words.

FILE>SBC

It is possible to download three different types of files to the F68HC11 from disk with the word FILE>SBC or the word SBC-INCLUDE: Forth code; Binary code in Motorola S19 format; and pure binary code.

Forth text files:

You may write Forth code using any text editor you have (you may not like the block editor of Pygmy). eg. you could download a file called "SREC.4TH".

1. Execute 9600-BAUD 4TH " SREC.4TH" FILE>SBC
2.Alternately Execute 9600-BAUD 4TH SBC-INCLUDE SREC.4TH

S19 format binary code:

You may create M68HC11 code using assembler or C. The assembler will save the result to disk in the Motorola S19 format. If you have an S19 code loader in the microcontroller you download with the following:

1. Execute 9600-BAUD S19 " FILENAME.S19" FILE>SBC.
2.Alternately Execute 9600-BAUD 4TH SBC-INCLUDE FILENAME.S19

Binary files:

The F68HC11 has a bootstrap mode. You can load a bootstrap mode S19 code loader program at 1200 baud into the first 256 bytes of RAM. (See the F68HC11 rescue for details of use)

1. Execute 1200-BAUD BIN " BOOT6811.BIN" FILE>SBC
2. Alternately Execute 9600-BAUD 4TH SBC-INCLUDEBOOT6811.BIN

USING THE TOOLS

Rescuing the F68HC11:

Sometimes you can get yourself into deep trouble with the F68HC11. You can put an autostart jump vector into the $B600-$B603 internal EEPROM memory location which points to buggy code. Every time you reset the machine it jumps to the buggy code and it is impossible to regain control. Because the code is in non-volatile memory even turning off the power does not help. There goes $30.00.

The machine looks in $B600-$B601 for a $A44A or $A55A flag. If the flag is found it uses the address found in $B602-$B603 as the jump address pointing to the autostart code. To rescue the machine you must change the $A44A or $A55A to anything else. The following procedure will save you $30.00 every time you use it.

1. Set the F68HC11 to Bootstrap mode.
2. Reset the F68HC11.
3. Load the binary file BOOT6811.BIN into the F68HC11.
4. Execute the terminal program T9600.
5. Enter repeated CR until the curser moves to the left margin.
6. Enter the character "I" <CR>(this points the bootloader program to internal EEPROM. "X"<CR> points to external EEPROM)
7. Enter the following string: "S104B600FF88 S9". This clears $B600 to $FF.
8. Set the F68HC11 to the operating mode, either single chip or external memory.
9. Reset the F68HC11. It will respond with the Max-Forth sign-on prompt and you are saved.

(This same procedure can be used to download any S19 file. Load the S19 downloader, then for step 7 use File>SBC to download the S19 file.)

The BOOT6811.BIN code came from the Motorola Application Note AN1010, "MC68HC11 EEPROM Programming from a Personal Computer" and the file has been included with the code submitted with this article. As well, a Hexadecimal listing is included for those who can create the file for themselves.

Continued on page 39
Source files on DIBs BBS, GEnie, and in next issue.

Special Feature

Ebedded Controllers

What You Get?

Playing with Micros

By Bill Kibler

Learning Embedded Controllers

For some time now, I have wanted to provide you with a review of what you get when buying one of the many embedded controller learning systems. The manufacturers call these system by many names: development systems, evaluation products, and prototype design systems to just name a few.

The first such system I came accross was the RSC-Forth system by Rockwell International. The system was based on the Rockwell R65F11 and R65F12 chips. These are 6501s with Forth in ROM. The date here is 1983 time frame and the product did fairly well for some time. The current status of the chips is being researched and more about them in a later issue.

The product was typical of what one obtained at the time. For your purchase (so long ago I do not remember how much) you got a development board, ready to run after adding power (five volts) and a serial port. For documentation you got the RSC-Forth User's Manual (Doc #29651N51, Order #2148 of October 1983) and a number of standard product hand outs.

This early product had some rather advanced things going for it. The development board could talk directly to a disk drive and thus save your work directly on disk. A companion board plugged into the main CPU socket so you could use the newer version of the chip (F12). All of this is covered in the User's manual. I do remember using the system and all worked fine except for a bug in the clock words. You can work around this bug, but the learner might give up before understanding why things don't work correctly.

Since then the world of embedded controllers has progressed considerably. I consider Motorola the learder in the area of providing new users with excellent learning systems. Recently Siemens and many others have also considered these inexpensive learning tools as good marketing tools and joined in the excitment.

I say excitment because you have little idea of what you will get when you order up one of these systems. I will review 5 such systems and hopefully make your testing one of these products more of a wonderful experience, than a poor surprise. Lets start with price.

Motorola and now others have typically set price in relationship to the device. That means a 68000 device might be $68 for a system. Motorola also typically uses a contest introduction of a new product to sell their newest chip design. The gimic here is a slightly higher price with an offfer of a rebate when you enter a completed design project using their system. There will also be some wining prize if your desgin is choosen best overall. Basically they sell the systems at or below their cost. What they gain is experienced user, good exposure, and product market leverage. The user gets a system for less than normal cost, usually a training manual that teaches you how to develope a product, some free software, and many nights of fun.

In studying the ones I have, I found also a new slant, support software. In the past you got only the minimum needed to develop a running project, typically an assembler and loader. Now days selling C compliers, Fuzzy logic, and Macro assemblers for embedded systems is big business. So now these systems come with running sample versions of real products, including full Real Time Excutives so you can have more than one task running in your demo system. Beware however that some of these products have been crippled and will only run limited amounts of I/O, or use a reduced set of libraies. Still you should be able to learn how they work and whehter or not to buy.

I have listed the facts in separate side bars to let you consider each option. I plan on doing a bit more on each product at a later date, and solict your comments and experiences with them. This is by no means a complete sampling, but a current assesment of what can be acquired if you act qucikly on seeing one of these promos. The fine print on the 80166 said their $166 price was only good during the four day run of the conference the handouts were given away at.

That brings up the next topic is how does one hear about these specials. Most major magazines that are supported heavily by these companies will run the sales promos. Going to conferences and visiting only the vendors section (usually Free or $25 charge for the day) will find plenty of good promos. If you deal with one of the larger distributors or sales reps, they can often inform you of these specials and see that you get yours. You must act quickly as Motorola contest boards are normally limited to low thousands (thats one to ten thousand units produced) and often these get sold within the first few days of their release. I remember one offer that went over so well,

Motorola did a second run of the systems.

The next question is, can you learn from these systems. I say yes! Some are better than others at learning. Take Motorolas 68HC16 DSP promo. The contest board was a DSP (Digital Signal Processor) add on to their regular evaluation board. A fairly good book explaining how to do DSP programming was their main tool for teaching. I felt it was a very good introduction and start for people learning how DSP worked. These are not for the novice engineer however, you will need to have done some programming and know something about electronics. I feel that almost all of the systems require that you have played around in the area before.

Can you do real work with these systems? In all cases these units could be used as standalone systems. They were not designed as such, as they usually contain RS232 interfaces and tools that might not be needed in a final product. You certainly can test your program and any hardware interfacing you might need before going on to the final design. For one up jobs they often can be more expensive than other methods until you consider all the free software and built in tools you get.

Lastly is there a better way to learn and develop new products, for these chips I think not. That is my bottom line, cheapest and fastest way to develope both skills and products in the embedded world. Now if the question is can I learn about general purpose computer hardware with these units, rather unlikely. They tend to be narrowly designed and thus have an embedded only slant. For general purpose and beginners, old Kaypros or S-100s are still my choice. If you are after embedded knowledge then give them a try the next time they are on sale!

Rockwell 65F11/F12

This 1983 Product was an early design built for developing total systems for later production. It came with:

1) System Board with R65F11 (1MHZ), WDC2793 (FDC), 6K RAM, 8K ROM with V1.8 of RSC-FORTH, on a 5.5" by 8.0" board. 34 pin Floppy port, DB25 for serial, 40 pins of expansion port, centronics printer port, and reset pushbutton.

2) RSC-FORTH User's Manual, Doc #29651N51, order #2148, Oct. 1983. An excellent manual with plenty of samples, glossary, and technical explanations.

3) Application Note #2162, "A low-cost development module for the R65F11 Forth Microcomputer". This is a must as it contains the schematic and PC board silk screens so you could make your own boards. Also has the program for the one PROM used for address decoding.

4) Product Description #2145, "R6501Q One-Chip microprocessor", this is the CPU description used as the basis for the FORTH CPU. It contains the hardware and instruction set information.

5) Other documents sometimes shipped with this product. 29650N30, order #202 "R6500 programming Manual"; 29650N31 order #201 "R6500 Hardware Manual"; 29651N49 order #2146 "RF65F11 and R65F12 Forth based Microcomputer Product description"; 29651N59 order #2156 "RSC-FORTH Reference Card".

Rockwell product availablity currently under research.

Motorola 68HC05

The 68HC705 is an EPROM based version of the 6805 CPU. The chip contains a bootstrap loader and the supplied board is used mostly for programming the CPU. Sales promo in 1989, cost $68.

1) The M68HC05PGMR is the 4.25" by 6.5" programming and evaluation board. Contains mostly sockets for the CPU (two types supported) and the EPROM socket that can contain a program for copying to the CPU. RS232 drivers, switches and LEDS, plus one output header for evaluation of I/O capabilities.

2) "M68HC05 Microcontroller Applications Guide", 1989. This manual covers how the chip works and is programmed. They use a thermostat example as the training project. Somewhat beginner oriented, but covers fundamentals quickly and really intended for engineers and programmers without embedded experience. Enough information and schematic detail is provided to actually build and use the thermostat project.

3) M6868HC05PGMR2/D1, "Programmer Board User's Manual #2". I believe the #2 refers to the second version of the programmer board, which is covered in detail (schematic but no silk screen).

4) "M68HC05 Instruction Set". Instruction by instruction explanation.

5) MC68HC705C8 CPU chip was included with document BR594/D which is the product summary literature and technical specifications.

6) One MSDOS 360K floppy with PC interface program (modem program) and "FREEWARE" programs to assemble 6805 code.

Evaluation and Contest boards avaialble from your local Motorola distributor.

Siemens/Rigel 80C166 Promo/Evaluation System

You will find a surprise in this package, as the evaluation board is made by Rigel of Florida. The price is a little steep at $199, but if you catch them at a show, it proably will be $166. I have yet to use all the software, so I am not sure how "crippled" it is. Most programs are "DEMO", but full functional versions. Comes in two book boxes, one with Rigel CPU, other literature and disks. The Rigel price for the RMB-166 is normally $180 plus shipping, so all the DEMO software is free so to speak.

1) Rigel Corp RMB-166 V1.1. This 4" by 6" board has the CPU, 2 32K RAM chip (HI & LO), spots for two EPROMs, 2 PALS, and RS232 driver device. Host and AUX serial ports, system bus strip, I/O port strip, and jumpers and switches for the various modes and operations possible. Unlike Motorola PCB, this CPU chip is surface mounted and not in a socket.
2) READS166 Demo V1.00, Rigels Embedded Applications Development System.
3) RMB-166 User's Guide, v1.1 January 1994.
4) Sales Literature by Siemens 80C166/83C166/88C166.
5) Product Guide for 166/167/165
6) Siemens '166 Family Applications Notes.
7) Errata Sheet 6/16/93 release 1.2.
8) Product Information SAB 80C166/83C166.
9) Fuzzy Tech from Inform Software Corp. V3.1 MCU-166 Explorer Edition (fuzzy logic software package).
10) HCR166 80C166 Asmbler/Downloader V1.0 (Hill Country Research).
11) CMX Real-Time Multitasking OS Evaluation Package Version BSO66E1-3.40.
12) Rigel Corp. RMB-166 schematic.
13) RTXC Eval Kit BSO/Tasking C V4.0 (DEMO).
14) Siemens User's Manual SAB 80C166/83C166.
15) Addendum to User's Manual.
16) Data Sheet 3.90 80C166.
17) TK1963 166 Eval Package C Compiler, Assembler, Monitor for PCDOS, V4.0 1-28-94, BSO/Tasking.

Siemens evaluation kit available from your local Siemens distributor. Rigel board available directly from Rigel Corporation, PO Box 90040, Gainesville, Fl 32607, (904) 373-4629. I will review the software supplied and AMR's Forth for this board in a later issue.

Motorola 68HC16 DSP Contest Board

You get two rather large flat boxes of items. One contains the regular M68HC16 evaluation products, the other has the DSP adapter board and DSP training books. A mid 1992 push to sell the new 68HC16 product and push it's DSP (digitial signal processing) abilties. Has more tools and yet still cost $68 after rebate ($136 before rebate) if you sent in the contest sheets.

1) M68HC16Z1EVB evaluation board, with 68HC16Z1, RAM, ROM, I/O headers, and a development area. CPU chip is socketed for replacement.
2) DSP DEMO board which can be used to do 5 band Audio Spectrum Analyzing when hooked to the EVB board.
3) M68HC16Z1EVB/D User's manual rev 1. Explains not only how to use the board, but some of the software included as well.
4) "Digital Signal Processing and the Microcontroller" by Motorola. Good intro and explanation of using embedded type controllers for DSP work.
5) HC16 Frequency Analyzer Project Manual.
6) Analog-to-Digital converter 8/10 ADC spec sheet.
7) M68HC16 Toolkit Project Software.
8) MAX274 8th order Active filter specifications.
9) MC14489 LED Driver spec sheet.
10) M68HC16 CPU16 Reference manual.
11) Queded Serial Module Reference manual(QSMRM/AD).
12) MC68331 User's Manual(covers the SIM and SRAM Modules).
13) General Purpose Timer Reference Manual(GPTRM/AD).
14) Evaluation Product Flyer and HC16 Contest Rebate instructions.
15) Toolware M68HC16 Macro Assembler Ver 4.1 MASM16 and Manual.
16) M68HC16 C Complier & Source Level Debugger by Intermetrics Microsystems Software Inc. Demo Kit by Whitesmiths.
17) M68HC16Z1 Tech Summary of 16 bit Microcontroller.
18) How to use the FREEWARE BBS by John Dumas, Motorola, Apr. 1991.
19) M68HC16Z1 Device Information Rev B.
20) Memory Map specificiations sheet.
21) M68HC16EVB/AN1 Application Note: Software vendors list.
22) M68HC16Z1EVB/PKG parts list and schematic diagrams.
23) A. T. Barrett MCX-16 Real Time Kernel for 68HC16.
24) U.S. Software GO FAST Floating Point Software Package.
25) Quickstart Demo Disk and Guide.
26) EVB16 Disk by P&E Micro Computing Systems Inc.
27) Filter Design & Analysis System for HC16 Ver 1.0 and Disk.

Evaluation and Contest boards avaialble from your local Motorola distributor.

AMR 8051 Development System

AMR is one of *TCJ*'s advertisers and produces many small systems. I borrowed their AMR51LC to demo for my current employer. The basic board prices start at $99 for one complete board, or $199 for a low-cost development package. Full developer's package is around $400. You get more hardware options and accessories as you go up in price. The full package has a keypad, small LCD display, cables, power transformer, more memory, and the full Forth devlopment package with plenty of samples and utilities already done for you. I liked AMR's construction better than Riegel's which I thought was a little sloppy. The demo package I have is the "Low Cost System" and would normaly cost $199.

1) AMR51LC mother board with 8K RAM & ROM, serial port devices, lithium battery, socketed SC87C51CCL44 CPU. This 4" by 6" board has a 2.5" square prototyping area (bed of holes) and two header strip sockets for stacking or I/O output. All devices are in sockets.

2) AMRForth and Hardware User's Manual. This 300+ pages of manual contains all you need to know to use the Forth development system and work with the hardware platform. While others give you the manuals separately, all devices used in all varations of AMR's boards (complete with schematics) are presented in this one book. Although not a beginners guide, it does provide enough samples to get a novice user up and running applications quickly.

3) AMR8051 FORTH software disk. This contains the source screens and utilties used to run the "tethered" Forth. A small kernel of Forth is inside the system that talks to a running Forth on your PC Clone. You can download and run your application quickly and get results of operations on the PC while the code is running on the target system.

4) F-PC V3.56, the PC Forth program, on which AMRForth runs and cross complies for downloading to the platform. This is a full system with wordprocessing, file management, debugger, windows like presentation, and multitasking operation. Currently the most popular sharewhare Forth system in use today.

5) 1 Amp wall transformer to power the system.

6) Adapter and cable for 9 pin AT serial port to 6 conductor RJ flat (phone type) ribbon cable, that plugs into the board's serial port.

Available from : AMR 4600 Hidden Oaks Lane, Loomis, CA 95650-9479, (916) 652-7472. I will cover this product and tethered Forth operation in a later issue.

Tools from page 35.
Building the T9600.COM program:

Block 11 of T9600.SCR is the load block for the tools. It depends on block 1 of BRADTOOL.SCR to load some preliminary words needed by the serial port words. First copy Block 12 of T9600.SCR to block 1 of BRADTOOL.SCR. This configures this block properly.

1. Start with PYGMY.COM as the foundation of the tools.
2. Execute RESET-FILES to close all files in Pygmy.
3. Open T9600.SCR by executing " T9600.SCR" 5 OPEN.
4. Load block 11 of T9600.SCR by executing 5011 LOAD
5. (Optional) Execute RESET-FILES to close all files in Pygmy.
6. Save the executable program with SAVE T9600.COM (or any other name you chose to give it)

Tool Extension:

Forth supports software reuse. Many functions are available for immediate use without any programming at all. The appropriate Forth word is invoked, the word is executed, and the result is returned. If you want to accomplish something more complex, then you can use the available words in new combinations to create additional words. This would suggest that you can create more tools with the Pygmy Forth capabilities. I started with the terminal, then created >SBC. Later, out of desperation I created the code to rescue the F68HC11 from the errant autostart. From that came the code to download Forth text and S19 files. So far, I have not needed more tools but you do not have to stop here. The endless possibilities of Forth are at your fingertips.

Resources:

Calvin McCarthy
12 Wedgewood Cr.
Gloucester, Ontario, Canada K1B 4B4

Pygmy Forth v. 1.4
Frank C. Sergeant
809 W. San Antonio St.
San Marcos, TX 78666

PYGTOOLS v1.2a
Lyle Greg Lisle
2160 Foxhunter Ct.
Winston-Salem, NC 27106
GEnie Mail: L.SQUARED
Internet: L.SQUARED@GENIE.GEIS.COM

New Micros Inc.
1601 Chalk Hill Road,
Dallas, Texas 75212, U.S.A.

Motorola Literature Distribution
P.O. Box 20912,
Phoenix, Arizona 85036, U.S.A.

Regular Feature

Editorial Comment

Building 8048 Systems

8048 Emulator

By J. G. Owens

THE STORY OF AN 8048 ETC. EMULATOR Part two.

LIMITATIONS OF THE EMULATION

If this is your first real-world lesson in emulation, be advised that *all* emulators have limitations; they *never* behave exactly the same as the real thing. They will *say* they do, but they're lying.

This 8048 emulator can probably only emulate 8748 and 8749 designs with 6 mHz clocks. I believe a late-breaking 1984 development in the 8048 world introduced 12 mHz clocks, but they were and probably still are more expensive, so I wasn't interested.

Also, P20, P21, P22, and P23 must be specified in hardware as to their inputs and outputs. A real 8048, in single-chip mode, allows you to use these pins as input or output, and change their orientation during program execution. In this emulator, you use H3 and a 16-pin header wired-up as you desire, to select the input/outputness of the things, and you're stuck with 'em; a third signal path is used when you want to emulate them attached to an external 8243 (I'm not convined that'd actually work...). See the schematic nearby H3 for example headers. (The technical problem is the 8048's dual usage of P20-P23 as I/O pins and as external addresses; since the target is "faked" into executing from external memory, the emulator design emulates the lost I/O pins by some cunning scheme, but it wasn't *that* cunning.)

Also, the OUTL instruction *can't* be used in an *emulating* program, but *should* be used in a running program. This is handled in your source with a macro like this:

```
emulate equ false;adjust things for emu-
lator;  probably only movx instead of
outl.

bus_a macro
    if emulate
        movx (r0),a
    endif
    if !emulate
        outl bus,a
    endif
endm
```

Note the encouraging certainty in the comment! I can be reasonably certain also that the emulated OUTL timing is different than the real thing.

U1:THE MAIN 8039 MONITOR

U1 executes the monitor program from an EPROM in U4 using the external bus mode of the 8039 (the 8039 is an 8048 family member with no internal ROM). Two 8243 extender chips are used to get some more I/O; these are parts sold by Intel specifically for this purpose, and the 8048 has special signal timings just for them.

The 8048 uses the familiar ALE strobe (i.e. 8085, 8088, and onwards ad infinitum) to convert a short moment of the data bus into external address lines A0..A7 (U7).

U22:THE TARGET 8039

U22 looks a lot like U1, which is of course the charm of the whole thing; it really would've made more sense for the U1 MONITOR to have been a more powerful processor, but in those glorious days I had a real "bootstrap" attitude.

HOW DOES THE MONITOR CONTROL THE TARGET?

The TARGET U22 normally reads program code from two RAMS, U19 and U21. But note the circuitry around U15, U16, known as OPREG2 and OPREG3. The general idea is that when the MONITOR wants to, it can force the TARGET to read a single instruction from one or both (two-byte instruction) of these registers *instead* of RAM. So the general scheme is that the MONITOR laboriously deludes the TARGET into executing various little sequences in single-step mode, to produce whatever results the monitor wants.

HOW DO BREAKPOINTS WORK?

One important feature of an emulator is the ability to run code at full speed but stop at a breakpoint. Actually, this is the *most* important feature.

The breakpoint circuitry's around U9, and works fairly predictably: the MONITOR sets-up a desired break address on brkA8-brkA11 and mP10..17 and enables the three comparators, which'll bang the TARGET into single-step mode when they compare correctly. Naturally there are various timing issues which have been handled flawlessly I'm sure as the accompanying diagram surely demonstrates.

DO BREAKPOINTS ALWAYS WORK CORRECTLY?

No. I was *so* disappointed when I found out that code like

```
ASBR:  .. do something..
```

```
        RET
ANOTHER:
        .. do something else ..
```

would erroneously breakpoint at the RET when I had specified a breakpoint at ANOTHER. That is, the 8048 apparently emits the next instruction address during a RET, thus triggering the breakpoint. But it was a problem I could live with.

Then years later I had a *real* emulator costing many dollars, *and it did the same thing*; with a different processor!....

HOW DOES THE MONITOR TELL WHAT ADDRESS THE TARGET STOPPED AT?

The emulator can free run, and then be stopped at any time (the single-step mode is asserted without the help of the comparators). When this happens, the MONITOR software figures-out where the TARGET stopped by turning-on the comparators and trying every single possible address, one after another. Moral: Software is often simple *and* stupid.

HOW DOES THE EMULATOR SOCKET WORK?

Moving on to the page of material near the EMULATOR socket U30 in the schematic document DB8039.A, basically all I can write is that various aspects of known 8048 timing are used to fake-up the single-chip interface, and it has definitely worked more than once. And the aspects I thought were important when I was fighting with it are documented in the DB8039.A text.

HEADER H3

As noted elsewhere, the OUTL instruction must be emulated in to-be-emulated source as "MOVX (R0),A"; the little forest of parts around U24 and U27 provide the hardware side to this kluge by various arcane means. The restrictions noted regarding P20..23 I/O are implemented via U26, H3, and U25.

The idea of an "8243" header, a sample of which is shown in nearby H3, is when you'd emulate the Intel 8243 port-expansion gadget; you'd hook one up, and use that header for H3; the note says "don't mix with input/output" which simply means if you try and conduct input/output on that port it won't work, which'd normally be the case anyway (you can't use an 8243 on an I/O port; it was intended to be used with an extended bus where P20..23 were used as address bits).

HEADER H4 AND WHERE ARE THOSE PARTS?

On my emulator, the H4 header is not installed. The part is in a little plastic box, constructed as described, but it isn't installed. The 100k SIP on H4 pins 9 to 14 would be used to simulate the input pull-ups on a real P2X pins. But the H4 C8 and R2 potentiometer! I must've been planning to delay a strobe somehow, but it looks like I gave-up on the project, but forgot to note it. Bad bad....

I just checked the circuit, and it is as I surmised; the 100K pull-ups are wired, and should probably be used as appropriate, i.e. when/if P20-23 are used as inputs (to simulate the "quasi-bidirectional" nature of those ports; see discussion somewhere above), but the C8 and R2 part are unused; no wiring is connected to those positions on H4. So I forgot them; perhaps someday I'll a note in some Kaypro diskette as to what I thought I was doing with them. I can *almost* remember now — but not quite....

POWER; RS-232

I have what looks like one of those wonderful digital surplus Commodore supplies in there, with the 12 volts or something attached to RG1.

RS-232 voltages are supplied in a vague way; the schematic shows what I call "+?V" and "-?V" supplied, using the input to the 5 volt regulator for +?V — usually around 8 volts or so — and a -5 volt supply for -?V.

Since then I've learned you can usually communicate with a PC's serial port without these voltages, using simpler means, i.e. you can use diode limiting or a 1489 RS-232 translator for input, as in the emulator schematic (the 1489 translates -/+ 12 volts to TTL levels using only ground and 5 volts) and then use a high-current TTL inverter as output to the PC's RS-232 input; this may work OK, or at least it has several times for me; the average PC card apparently sees ground as low and 5 volts as high.... Your mileage may vary....

THE MONITOR SOURCE

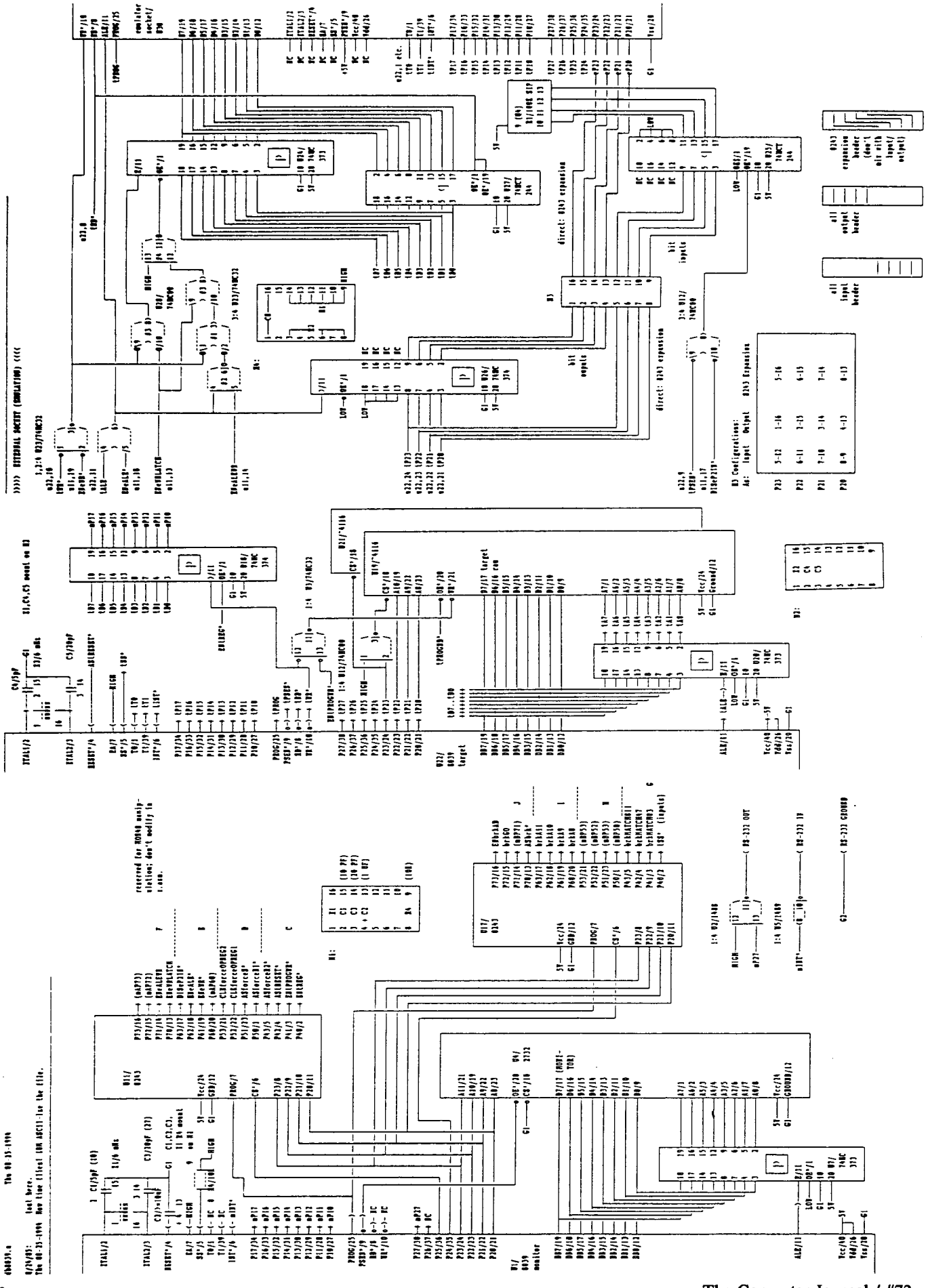
FILES: XCOM.LB Common constants for monitor source.

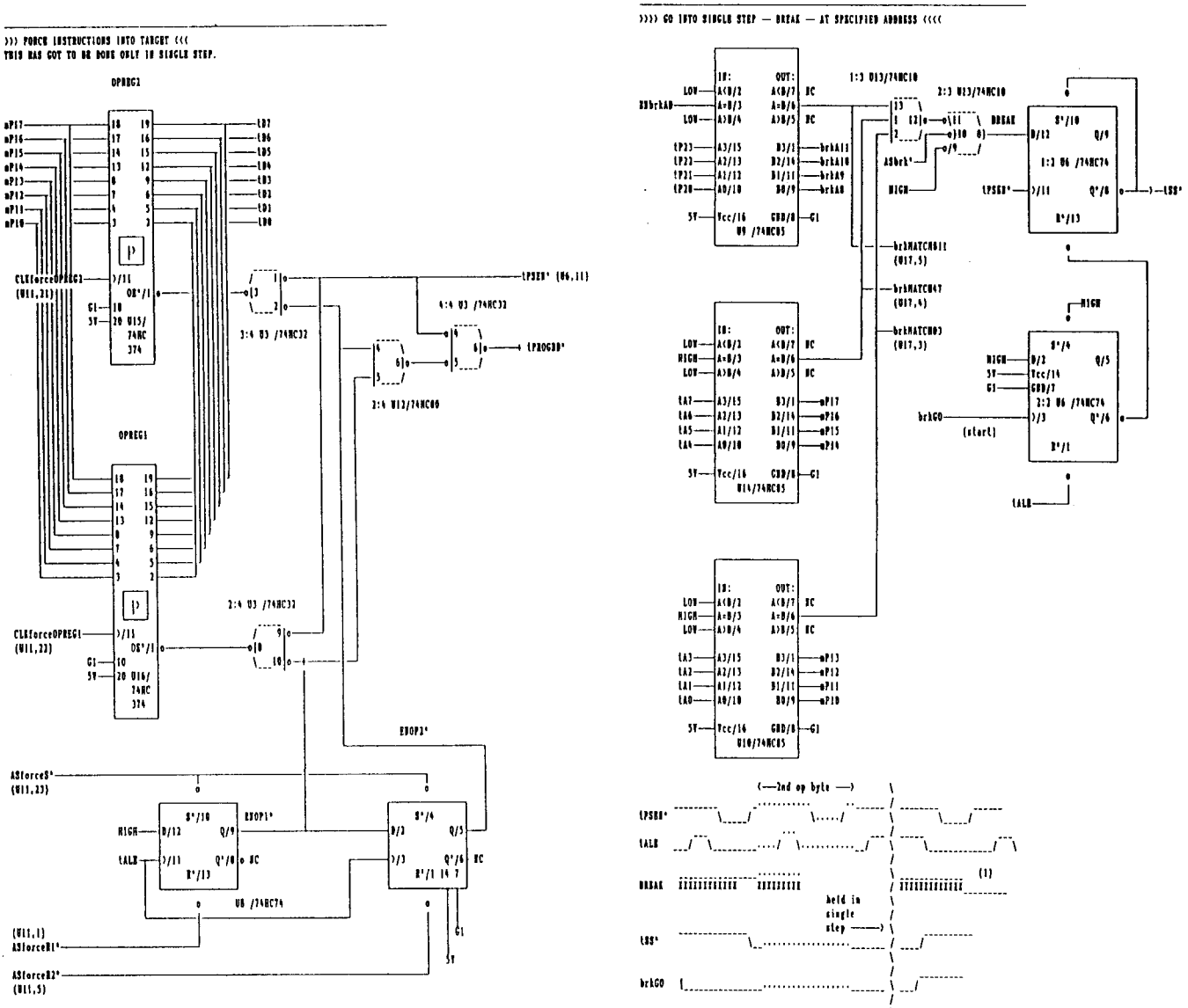
X1.MAC	Source.
X2.MAC	Source.
X.MAK	Make file (Borland MAKE 3.5?).
X.HEX	Hex image.
X.BIN	Binary image.

X.MAK works with Borland Make 3.5 but probably with earlier versions of the Borland make programs, or perhaps other make programs. If you don't have a make program, X.MAK is so simple it should be easy-enough to figure out. I mean, we're talking about the least of your problems.

Next time we will continue with discussion of the monitor source code. The next pages are the schematics and parts list for the emulation hardware used. BDK.

The source and schematic files are on Genie and DIBs BBS.





The diagram shows two-op-code instructions. For a one op-code instruction, load OPREG2, and bring ASforceR2* down and then up. The little shift-register will always eventually allow normal program reads. The point of this arrangement is so that the target processor can do page reads -- reads of its own program space. ASforceS* is just a mechanism for initialization and might not be necessary.

Special Feature

Intermediate Users

Part 7.5: 8051 continued

Moving Forth

by Brad Rodriguez

MOVING FORTH
Part 7.5: CamelForth for the 8051
by Brad Rodriguez

Under the prodding of Our Esteemed Editor, I present CamelForth for the 8051, second part. The first part appeared in the last issue #71.

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

; branch and ?branch are done with sjmp and jz,
; respectively, using the following routines
; which leave a value in A.  Typical use:
;       lcall zerosense, jz destadr
;       lcall loopsense, jz destadr, lcall unloop
; LEAVE may exit loop by branching ^—here

        .drw link
        .set link,*+1
        .db  0,7,"?BRANCH"
qbranch:
zerosense:              ; n — leave zero in A if TOS=0
        movx a,@r0      ; new TOS in a:r2
        mov r2,a
        inc r0
        movx a,@r0
        inc r0
        xch a,dph       ; DPH=new TOS hi, A=old DPH
        orl a,dpl       ; A=0 if old TOS was zero
        mov dpl,r2      ; new TOS lo in DPL
        ret

; LOOP and +LOOP are done with jz, using the
; following routines which leave a value in A.
; If the loop terminates, (index crosses 8000h),
; a nonzero value is left in A.  A=0 to loop.
; Typical use:
;       lcall loopsense, jz destadr, lcall unloop
; LEAVE may exit loop by branching ^-here
; The topmost loop index is in regs r7:r6.

        .drw link
        .set link,*+1
        .db  0,6,"(LOOP)"
xdoop:
loopsense:              ; —   leave 0 in A if 'loop'
        mov a,r6        ; add 1 to loop index
        add a,#1        ; ...leaves OV flag set if
        mov r6,a        ; loop terminates
        mov a,r7
        addc a,#0
        mov r7,a
        jb psw.2,termloop ; jump if OV set
takeloop:
        clr a           ; OV clear, make A zero
        ret             ;       to take loop branch

        .drw link
        .set link,*+1
        .db  0,7,"(+LOOP)"
xplusloop:
plusloopsense:          ; n — leave 0 in A if '+loop'
        mov a,r6        ; add TOS to loop index
        add a,dpl       ; ...leaves OV flag set if
        mov r6,a        ; loop terminates
        mov a,r7
        addc a,dph
        mov r7,a
        movx a,@r0 ; pop new TOS, OV unaffected
        mov dpl,a
        inc r0
        movx a,@r0
        mov dph,a
        inc r0
        jnb psw.2,takeloop ; jump if OV clear
termloop:
        clr a           ; OV set, make A nonzero
        cpl a           ; to force loop termination
        ret

;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.
        .drw link
        .set link,*+1
        .db  0,4,"(DO)"
XDO:    ; limit index —
        pop dr3         ; get return adrs in r3:r2
        pop dr2
        push dr6        ; push previous index
        push dr7
        movx a,@r0      ; get (-limit) + 8000h
        inc r0          ; = (~limit) + 8001h
        cpl a           ; in r5:r4
        add a,#01
        mov r4,a
        movx a,@r0
        inc r0
        cpl a
        addc a,#h'80
        mov r5,a
        push dr4        ; push this fudge factor
        push dr5
        mov a,r4        ; fudge+index -> r7:r6
        add a,dpl
        mov r6,a
        mov a,r5
        addc a,dph
        mov r7,a
        push dr2        ; restore return addr
        push dr3
        ajmp poptos     ; go pop new TOS

;C I            —  n        R: sys1 sys2 — sys1 sys2
;C              get the innermost loop index
        .drw link
        .set link,*+1
        .db  0,1,"I"
II:     dec r0          ; push old TOS
        mov a,dph
        movx @r0,a
        dec r0
        mov a,dpl
        movx @r0,a
        mov r1,sp       ; get copy of SP
        dec r1          ; skip return address
        dec r1
        dec r1          ; skip hi byte of fudge
        clr c
        mov a,r6        ; index-fudge = true index
        subb a,@r1
        mov dpl,a
        inc r1
        mov a,r7
        subb a,@r1
        mov dph,a       ; leaves true index on TOS
        ret

;C J            —  n        R: 4*sys — 4*sys
;C              get the second loop index
        .drw link
        .set link,*+1
        .db  0,1,"J"
JJ:     dec r0          ; push old TOS
        mov a,dph
        movx @r0,a
        dec r0
        mov a,dpl
        movx @r0,a
        mov r1,sp       ; get copy of SP
        dec r1          ; skip return address
        dec r1
        dec r1          ; skip inner fudge factor
        dec r1
        mov dr3,@r1     ; outer index hi
        dec r1
        mov dr2,@r1     ; outer index lo
        dec r1
        mov b,@r1       ; outer fudge hi
        dec r1
        clr c
        mov a,r2        ; index-fudge = true index
        subb a,@r1
        mov dpl,a
        mov a,r3
        subb a,b
        mov dph,a       ; leaves true index on TOS
        ret

;C UNLOOP       —        R: sys1 sys2 —  drop loop parms
        .drw link
        .set link,*+1
        .db  0,6,"UNLOOP"
UNLOOP: pop dr3         ; get return adrs in r3:r2
        pop dr2
        dec sp          ; discard fudge factor
        dec sp
        pop dr7         ; restore previous loop index
        pop dr6
        push dr2        ; restore return addr
        push dr3
        ret

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

;C UM*          u1 u2 — ud      unsigned 16x16->32 mult.
        .drw link
        .set link,*+1
        .db  0,3,"UM*"
UMSTAR: movx a,@r0      ; u1 Lo in r1
        mov r1,a
        inc r0          ; u1 Hi in meml

        mov a,r1        ; u1L*u2L -> B:A -> r3:r2
        mov b,dpl
        mul ab
        mov r2,a
        mov r3,b

        mov a,r2        ; u1L*u2H -> B:A
        mov b,dph       ;       add into r4:r3
        mul ab
        add a,r3
        mov r3,a
        clr a
        addc a,b
        mov r4,a

        movx a,@r0      ; u1H*u2L -> B:A
        mov b,dpl       ; add into r4:r3
        mul ab
        add a,r3
        mov r3,a
```

```
        mov a,r4
        addc a,b
        mov r4,a
        clr a           ; w/possible cy->r5
        addc a,#0
        mov r5,a

        movx a,@r0      ; u1H*u2H -> B:A
        mov b,dph       ; add into r5:r4
        mul ab
        add a,r4
        mov r4,a
        mov a,r5
        addc a,b
        mov dph,a       ; result in dph:r4:r3:r2
        mov dpl,r4
        mov a,r3
        movx @r0,a
        dec r0
        mov a,r2
        movx @r0,a
        ret

;C UM/MOD  ud u1 — u2 u3   unsigned 32/16->16
        .drw link
        .set link,*+1
        .db  0,6,"UM/MOD"
UMSLASHMOD:             ; DPH:DPL = divisor
        movx a,@r0      ; r2:r3:r4:r5 = dividend
        mov r3,a        ; note stack order:
        inc r0          ; ^ xxxx
        movx a,@r0      ; | xxxx
        mov r2,a        ; high hi byte \ low
        inc r0          ; adrs lo byte /  cell
        movx a,@r0      ;     hi byte \ high
        mov r5,a        ; R0—> lo byte /  cell
        inc r0          ; on———
        movx a,@r0      ; entry
        mov r4,a        ;
        mov r1,#17      ; loop counter
        clr c
        sjmp div2
div1:                   ; division loop
        mov a,r3
        rlc a
        mov r3,a
        mov a,r2
        rlc a
        mov r2,a
        jnc div3
        ; here cy=1, cy:r2:r3 is a 17 bit value,
        ; we know we can subtract divisor
        clr c
        mov a,r3
        subb a,dpl
        mov r3,a
        mov a,r2
        subb a,dph
        mov r2,a
        clr c
        sjmp div4
div3:   ; here cy=0, r2:r3 is a 16 bit value
        clr c
        mov a,r3
        subb a,dpl
        mov r3,a
        mov a,r2
        subb a,dph
        mov r2,a
        jnc div4
        ; borrow occurred — undo the subtract
        mov a,r3
        add a,dpl
        mov r3,a
        mov a,r2
        addc a,dph
        mov r2,a
        setb c
div4:   ; here cy=0 if subtracted, cy=1 if not
        cpl c
div2:   mov a,r5
        rlc a
        mov r5,a
        mov a,r4
        rlc a
        mov r4,a
        djnz r1,div1
        mov dpl,r5      ; put quotient in TOS
        mov dph,r4
        mov a,r2        ; push remainder on stack
        movx @r0,a
        dec r0
        mov a,r3
        movx @r0,a
        ret

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

;C FILL     c-addr u char —  fill Data mem w/char
        .drw link
        .set link,*+1
        .db  0,4,"FILL"
FILL:   mov r4,dpl      ; stash char temporarily
        movx a,@r0      ; get count in r3:r2
        mov r2,a
        inc r0
        movx a,@r0
        mov r3,a
        inc r0
        movx a,@r0      ; get addr in DPTR
        mov dpl,a
        inc r0
        movx a,@r0
        mov dph,a
        inc r0
        mov a,r4        ; get char in A
        inc r3          ; adjust r3,r2 for djnz loop
        inc r2
        sjmp filltest
filloop: movx @dptr,a
        inc dptr
filltest: djnz r2,filloop
        djnz r3,filloop
        ajmp poptos     ; pop new TOS

;Z I->D     c-addr1 c-addr2 u —  move Code->Data
; Block move from Code space to Data space.
;       ?DUP IF
;       OVER + SWAP DO
;       DUP IC@ I CI 1+
;       LOOP DUP
;       THEN 2DROP ;
        .drw link
        .set link,*+1
        .db  0,4,"I->D"
ITOD:   acall QDUP
        acall zerosense
        jz itod2
        acall OVER
        acall PLUS
        acall SWOP
        acall XDO
itod1:  acall DUP
        acall ICFETCH
        acall II
        acall CSTORE
        acall ONEPLUS
        acall loopsense
        jz itod1
        acall UNLOOP
        acall DUP
itod2:  acall DROP
        ajmp DROP

;Z D->I     c-addr1 c-addr2 u —  move Data->Code
; Block move from Data space to Code space.
; On the 8051 this is identical to CMOVE.
        .drw link
        .set link,*+1
        .db  0,4,"D->I"
DTOI:   sjmp CMOVE

;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.
;       ?DUP IF
;       OVER + SWAP DO
;       DUP C@ I CI 1+
;       LOOP DUP
;       THEN 2DROP ;
        .drw link
        .set link,*+1
        .db  0,5,"CMOVE"
CMOVE:  acall QDUP
        acall zerosense
        jz cmove2
        acall OVER
        acall PLUS
        acall SWOP
        acall XDO
cmove1: acall DUP
        acall CFETCH
        acall II
        acall CSTORE
        acall ONEPLUS
        acall loopsense
        jz cmove1
        acall UNLOOP
        acall DUP
cmove2: acall DROP
        ajmp DROP

;X CMOVE>   c-addr1 c-addr2 u —  move from top
; as defined in the ANSI optional String word set
;       ?DUP IF
;       1- ROT OVER +  \ addr2 u-1 addr1+u-1
;     ROT ROT OVER +  \ addr1+u-1 addr2 addr2+u-1
;       DO
;       DUP C@ I CI 1-
;       -1 +LOOP DUP
;       THEN 2DROP ;
        .drw link
        .set link,*+1
        .db  0,6,"CMOVE>"
CMOVEUP: acall QDUP
        acall zerosense
        jz cmovu2
        acall ONEMINUS
        acall ROT
        acall OVER
        acall PLUS
        acall ROT
        acall ROT
        acall OVER
        acall PLUS
        acall XDO
cmovu1: acall DUP
        acall CFETCH
        acall II
        acall CSTORE
        acall ONEMINUS
        acall LIT
        .drw -1
        acall plusloopsense
        jz cmovu1
        acall UNLOOP
        acall DUP
cmovu2: acall DROP
        ajmp DROP

;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.
        .drw link
        .set link,*+1
        .db  0,4,"SKIP"
SKIP:   mov r4,dpl      ; stash char temporarily
        movx a,@r0      ; get count in r3:r2
        mov r2,a
        inc r0
        movx a,@r0
        mov r3,a
        inc r0
        movx a,@r0      ; get addr in DPTR
        mov dpl,a
        inc r0
        movx a,@r0
        mov dph,a
        inc r3          ; adj r3,r2 for djnz loop
        inc r2
        sjmp skiptest
skiploop: movx a,@dptr  ; get char
        xrl a,r4        ; compare with desired
        jnz skipmis     ; exit if mismatch
        inc dptr
skiptest: djnz r2,skiploop
        djnz r3,skiploop
        ; count exhausted; r3:r2=0000,
        ;       adrs points past last char
skipmis: ; either mismatch, or count exhausted
        mov a,dph       ; push updated addr
        movx @r0,a
        dec r0
        mov a,dpl
        movx @r0,a
        dec r2          ; adjust r3,r2 back
        dec r3          ;     to a normal count,
        mov dph,r3 ; put in TOS,
        mov dpl,r2
        inc dptr        ; adjust for extra decr
        ret

;Z SCAN     c-addr u c — c-addr' u'
;Z          find matching char
        .drw link
        .set link,*+1
        .db  0,4,"SCAN"
SCAN:   mov r4,dpl      ; stash char temporarily
        movx a,@r0      ; get count in r3:r2
        mov r2,a
        inc r0
        movx a,@r0
```

```
            mov r3,a
            inc r0
            movx a,@r0      ; get addr in DPTR
            mov dpl,a
            inc r0
            movx a,@r0
            mov dph,a
            inc r3          ; adj r3,r2 for djnz loop
            inc r2
            sjmp scantest
scanloop:   movx a,@dptr    ; get char
            xrl a,r4        ; compare with desired
            jz scanmis      ; exit if match
            inc dptr
scantest:   djnz r2,scanloop
            djnz r3,scanloop
            ; count exhausted; r3:r2=0000
            ; adrs points past last char
scanmis:    ; either match, or count exhausted
            mov a,dph ; push updated addr
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            dec r2          ; adjust r3,r2 back
            dec r3          ; to a normal count,
            mov dph,r3      ; put in TOS,
            mov dpl,r2
            inc dptr        ; adjust for extra decr
            ret

;Z S=       c-addr1 c-addr2 u — n string compare
;Z          n<0: s1<s2, n=0: s1=s2, n>0: s1>s2
; Omitted in 8051 version.

;Z N=       c-addr1 c-addr2 u — n string:name cmp
;Z          n<0: s1<s2, n=0: s1=s2, n>0: s1>s2
;          ?DUP IF
;          OVER + SWAP DO
;          DUP C@ I IC@ -
;          ?DUP IF NIP UNLOOP EXIT THEN
;          1+ LOOP DUP
;          THEN 2DROP 0 ;
; Harvard model: c-addr1=>Data, c-addr2=>Code.
            .drw link
            .set link,*+1
            .db 0,2,"N="
NEQUAL:     push dr7
            push dr6
            mov r2,dpl ; count
            mov r3,dph
            movx a,@r0      ; get Code addr in r5:r4
            mov r4,a
            inc r0
            movx a,@r0
            mov r5,a
            inc r0
            movx a,@r0      ; get Data addr in r7:r6
            mov r6,a
            inc r0
            movx a,@r0
            mov r7,a
            inc r0
            inc r3          ; adjust for djnz loop
            inc r2
            sjmp Nequtest
Nequloop:   mov dph,r5      ; get Code char
            mov dpl,r4
            clr a
            movc a,@a+dptr
            mov r1,a
            inc dptr
            mov r5,dph
            mov r4,dpl
            mov dph,r7      ; get Data char
            mov dpl,r6
            movx a,@dptr
            inc dptr
            mov r7,dph
            mov r6,dpl
            clr c           ; Data-Code
            subb a,r1
            jnz Nequfail
Nequtest:   djnz r2,Nequloop
            djnz r3,Nequloop
            mov dph,r3      ; strings match, r3=0,
            mov dpl,r3      ; so make TOS=0
            sjmp Nequdone
Nequfail:   subb a,acc      ; -1 if cy set, 0 if clr
            mov dph,a       ; (Data<Code) (Data>Code)
            orl a,#1        ; TOS = FFFF or 0001
            mov dpl,a
Nequdone:   pop dr6
            pop dr7
            ret
```

```
;
;=============================================
; CamelForth for the Intel 8051
; (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
;
; CAMEL51D.AZM: CPU and Model Dependencies
;           Source code is for the A51 assembler.
;           Forth words are documented as follows:
;*          NAME        stack — stack       description
;           Word names in upper case are from the ANS
;           Forth Core word set.  Names in lower case
;           "internal" implementation words & extensions.
;
; Subroutine-Threaded Forth model for Intel 8051
;           cell size is   16 bits (2 bytes)
;           char size is   8 bits (1 byte)
;           address unit is 8 bits (1 byte), i.e.,
;           addresses are byte-aligned.
;
;=============================================

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

;C ALIGN    —           align HERE
            .drw link
            .set link,*+1
            .db 0,5,"ALIGN"
ALIGN:      ret         ; noop!

;C ALIGNED  addr — a-addr      align given addr
            .drw link
            .set link,*+1
            .db 0,7,"ALIGNED"
ALIGNED:    ret         ; noop!

;Z CELL     — n          size of one cell
            .drw link
            .set link,*+1
            .db 0,4,"CELL"
CELL:       acall DOCON
            .drw 2

;C CELL+    a-addr1 — a-addr2    add cell size
;          2 + ;
            .drw link
            .set link,*+1
            .db 0,5,"CELL+"
CELLPLUS:   inc dptr
            inc dptr
            ret

;C CELLS    n1 — n2      cells->adrs units
            .drw link
            .set link,*+1
            .db 0,5,"CELLS"
CELLS:      ajmp twostar

;C CHAR+    c-addr1 — c-addr2    add char size
            .drw link
            .set link,*+1
            .db 0,5,"CHAR+"
CHARPLUS:   inc dptr
            ret

;C CHARS    n1 — n2      chars->adrs units
            .drw link
            .set link,*+1
            .db 0,5,"CHARS"
CHARS:      ret

;C >BODY    xt — a-addr  adrs of param field
;          3 + ;         Z80 (3 byte CALL)
            .drw link
            .set link,*+1
            .db 0,5,">BODY"
TOBODY:     inc dptr
            inc dptr
            inc dptr
            ret

; Note that I@ and II use lo,hi byte order (same
; as 8086 and Z80), but the 8051 LCALL and LJMP
; addresses are stored hi,lo.  This difference
; is encapsulated within ,XT ICF and ,CF .

;X COMPILE, xt —         append execution token
; I called this word ,XT before I discovered that
; it is defined in the ANSI standard as COMPILE,.
; On a DTC Forth this simply appends xt (like , )
; but on an STC Forth this must append 'CALL xt'.
;           012 IC, >< I, ;  12h = 8051 Lcall instruction
            .drw link
            .set link,*+1
            .db 0,8,"COMPILE,"
COMMAXT:    acall LIT
            .drw h'12
            lcall ICCOMMA
            acall SWAPBYTES
            ljmp ICOMMA

;Z ICF      adrs cfa —   set code action of a word
;           012 OVER ICI          store 'LCALL adrs' instr
;           1+ SWAP >< SWAP II              ; 8051
Harvard VERSION
; Depending on the implementation this could
; append CALL adrs or JUMP adrs.
            .drw link
            .set link,*+1
            .db 0,3,"ICF"
STORECF:    acall LIT
            .drw h'12
            acall OVER
            acall ICSTORE
            acall ONEPLUS
            acall SWOP
            acall SWAPBYTES
            acall SWOP
            ajmp ISTORE

;Z ,CF      adrs —       append a code field
;           012 IC, >< I, ;          8051 Harvard
VERSION
            .drw link
            .set link,*+1
            .db 0,3,",CF"
COMMACF:    ajmp COMMAXT

;Z ICOLON   —            change code field to docolon
;           -5 IALLOT ;  8051 Harvard VERSION
; This should be used immediately after CREATE.
; This is made a distinct word, because on an STC
; Forth, colon definitions have no code field.
            .drw link
            .set link,*+1
            .db 0,6,"ICOLON"
STORCOLON: acall LIT
            .drw -5
            ljmp IALLOT

; ***
; This is approximately the end of the first 2K
; block.  CALLs and JMPs crossing this boundary
; must use the Long form.
; ***

;Z ,EXIT    —            append hi-level EXIT action
;           022 IC, ;    8051 VERSION
; This is made a distinct word, because on an STC
; Forth, it appends a RET instruction, not an xt.
            .drw link
            .set link,*+1
            .db 0,5,",EXIT"
CEXIT:      lcall LIT
            .drw h'22
            ljmp ICCOMMA

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

;Z ,BRANCH xt —          append a branch instruction
; xt is the branch operator to use, e.g. qbranch
; or (loop).  It does NOT append the destination
; address.  On the 8051 this compiles
;           LCALL xt    jz-opcode
; unless xt=0 in which case the LCALL is omitted
; and an 'sjmp' instruction is compiled.
;           ?DUP IF ,XT 080 ELSE 080 THEN IC, ;
            .drw link
            .set link,*+1
            .db 0,7,",BRANCH"
COMMABRANCH: lcall QDUP
            lcall zerosense
            jz combr1
            lcall COMMAXT       ; LCALL sense-routine
            lcall LIT
            .drw h'60           ; jz opcode
            ajmp ICCOMMA
combr1:     lcall LIT
            .drw h'80           ; sjmp opcode
            ajmp ICCOMMA

; high level code may use 'branch' as an argument
```

```
; to ,BRANCH:
            .equ branch,0

;Z ,DEST     dest —      append a branch address
; This appends the given destination address to
; the branch instruction.  On the 8051 this is a
; one-byte relative address.
;           HERE 1+ - IC, ;
            .drw link
            .set link,*+1
            .db  0,5,",DEST"
COMMADEST:  lcall HERE
            lcall ONEPLUS
            lcall MINUS
            ajmp ICCOMMA

;Z IDEST     dest adrs — change a branch dest'n
; Changes the destination address found at 'adrs'
; to the given 'dest'.  On the 8051 this is a
; one-byte relative address.
;           TUCK 1+ - SWAP IC! ;
            .drw link
            .set link,*+1
            .db  0,5,"IDEST"
STOREDEST:  lcall TUCK
            lcall ONEPLUS
            lcall MINUS
            lcall SWOP
            ljmp ICSTORE

;Z ,UNLOOP —          append an UNloop instruction
; Used after a LOOP or +LOOP is compiled.
; Required on the 8051 because the loop branch
; must be followed by UNLOOP.  No-op on Z80.
;           ['] UNLOOP ,XT ;
            .drw link
            .set link,*+1
            .db  0,7,",UNLOOP"
COMMAUNLOOP: lcall LIT
            .drw UNLOOP
            ljmp COMMAXT

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

;Z IDP       — a-addr   ROM dictionary pointer
;           20 USER IDP
            .drw link
            .set link,*+1
            .db  0,3,"IDP"
IDP:  lcall douser
            .drw 20

;Z (IS")     — c-addr u  run-time code for S"
;           R> ICOUNT 2DUP + ALIGNED >R ;
; Harvard model, for string stored in Code space
; e.g. as used by ."
            .drw link
            .set link,*+1
            .db 0,5,"(IS",h'22,")"
XISQUOTE:   lcall RFROM
            acall COUNT
            acall TWODUP
            lcall PLUS
            ; lcall ALIGNED
            lcall TOR   ; do NOT ljmp TOR!
            ret

;Z (S")      — c-addr u  run-time code for S"
;           R@ I@       get Data address
;           R> CELL+ DUP IC@ CHAR+          — Dadr
Radr+2 n+1
;           2DUP + ALIGNED >R   — Dadr ladr n+1
;           >R OVER R> I>D      — Dadr
;           COUNT ;
; Harvard model, for string stored in Code space
; which is copied to Data space.
            .drw link
            .set link,*+1
            .db 0,4,"(S",h'22,")"
XSQUOTE:    lcall RFETCH
            lcall IFETCH
            lcall RFROM
            lcall CELLPLUS
            lcall DUP
            lcall ICFETCH
            lcall CHARPLUS
            acall TWODUP
            lcall PLUS
            ; lcall ALIGNED
            lcall TOR
            lcall TOR
            lcall OVER
            lcall RFROM
            lcall ITOD
            ajmp COUNT

;C IS"       —           compile in-line string
;           COMPILE (IS")  [ HEX ]
;           22 IWORD
;           IC@ 1+ ALIGNED IALLOT ; IMMEDIATE
; Harvard model: string is stored in Code space
            .drw link
            .set link,*+1
            .db IMMED,3,"IS",h'22
ISQUOTE:    lcall LIT
            .drw XISQUOTE
            lcall COMMAXT
            lcall LIT
            .drw h'22
            acall IWORD
            lcall ICFETCH
            lcall ONEPLUS
            ; lcall ALIGNED
            ajmp IALLOT

;C S"        —           compile in-line string
;           COMPILE (S")  [ HEX ]
;           HERE I,     data address
;           22 IWORD
;           IC@ 1+ ALIGNED
;           DUP ALLOT IALLOT ; IMMEDIATE
; Harvard model: string is stored in Code space
            .drw link
            .set link,*+1
            .db IMMED,2,"S",h'22
SQUOTE:     lcall LIT
            .drw XSQUOTE
            lcall COMMAXT
            acall HERE
            acall ICOMMA
            lcall LIT
            .drw h'22
            acall IWORD
            lcall ICFETCH
            lcall ONEPLUS
            ; lcall ALIGNED
            lcall DUP
            acall ALLOT
            ajmp IALLOT

;C ."        —           compile string to print
;           POSTPONE IS"  POSTPONE ITYPE ;
IMMEDIATE
            .drw link
            .set link,*+1
            .db IMMED,2,".",h'22
DOTQUOTE:   acall ISQUOTE
            lcall LIT
            .drw ITYPE
            ljmp COMMAXT

;Z ICOUNT  c-addr1 — c-addr2 u  counted->adr/len
;           DUP CHAR+ SWAP IC@ ;          from Code
space
            .drw link
            .set link,*+1
            .db  0,6,"ICOUNT"
ICOUNT:     lcall DUP
            lcall CHARPLUS
            lcall SWOP
            ljmp ICFETCH

;Z ITYPE     c-addr +n —             type line to term'l
;           ?DUP IF     from Code space
;           OVER + SWAP DO I IC@ EMIT LOOP
;           ELSE DROP THEN ;
            .drw link
            .set link,*+1
            .db 0,5,"ITYPE"
ITYPE:      lcall QDUP
            lcall zerosense
            jz ITYP4
            lcall OVER
            lcall PLUS
            lcall SWOP
            lcall XDO
ITYP3:      lcall II
            lcall ICFETCH
            lcall EMIT
            lcall loopsense
            jz ITYP3
            lcall UNLOOP
            sjmp ITYP5
ITYP4:      lcall DROP
ITYP5:      ret

;Z IWORD   c — c-addr WORD to Code space
;           WORD
;           DUP IHERE OVER C@ CHAR+ D>I ;
            .drw link
            .set link,*+1
            .db 0,5,"IWORD"
IWORD:      lcall WORD
            lcall DUP
            acall IHERE
            lcall OVER
            lcall CFETCH
            lcall CHARPLUS
            ljmp DTOI

; The following additional words support the
; "Harvard" model, with separate address spaces
; for Instructions (Code) and Data.  ANSI
; requires DP to manage the Data space, so a
; separate Instruction Dictionary Pointer, IDP,
; is added to manage the Code space.  Also added:
;           I@ IC@ II ICI               (in the primitives)
;           IHERE IALLOT I, IC,
;           ITYPE ICOUNT WORD>I
; It should be possible to convert the Harvard
; implementation to a combined-code-and-data
; system, by equating these words to their
; Data-space counterparts.

;Z IHERE     — addr      return Code dictionary ptr
;           IDP @ ;
            .drw link
            .set link,*+1
            .db 0,5,"IHERE"
IHERE:      acall IDP
            ljmp FETCH

;Z IALLOT    n —         allocate n bytes in Code dict
;           IDP +! ;
            .drw link
            .set link,*+1
            .db 0,6,"IALLOT"
IALLOT:     acall IDP
            ljmp PLUSSTORE

;Z I,        x —         append cell to Code dict
;           IHERE II 1 CELLS IALLOT ;
            .drw link
            .set link,*+1
            .db 0,2,"I,"
ICOMMA:     acall IHERE
            lcall ISTORE
            lcall lit
            .drw 2
            sjmp IALLOT

;Z IC,  x —  append char to Code dict
;           IHERE IC! 1 CHARS IALLOT ;
            .drw link
            .set link,*+1
            .db 0,3,"IC,"
ICCOMMA:    acall IHERE
            lcall ICSTORE
            lcall lit
            .drw 1
            sjmp IALLOT
```

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Volume Number 1:

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

Volume Number 2:

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

Volume Number 3:

- **Issues 20 to 25**
- Designing an 8035 SBC
- Using Apple Graphics from CP/M
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K
- Extending Turbo Pascal: series
- Unsoldering: The Arcane Art
- Analog Data Acquisition & Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC
- NEW-DOS: series
- Variability in the BDS C Standard Library
- The SCSI Interface: series
- Using Turbo Pascal ISAM Files
- The Ampro Little Board Column: series
- C Column: series
- The Z Column: series
- The SCSI Interface: Introduction to SCSI
- Editing the CP/M Operating System
- INDEXER: Turbo Pascal Program to Create an Index
- Selecting & Building a System
- Introduction to Assemble Code for CP/M
- Ampro 186 Column
- ZTime-1: A Real Time Clock for the Ampro Z-80 Little Board

Volume Number 4:

- **Issues 26 to 31**
- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside Ampro Computers
- NEW-DOS: The CCP Commands (continued)
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS
- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation: Disassembling Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis & Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program in Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying the CP/M Disk Parameter Block for Foreign Disk Formats
- Starting Your Own BBS
- Build an A/D Converter for the Ampro Little Board
- HD64180: Setting the Wait States & RAM Refresh using PRT & DMA
- Using SCSI for Real Time Control
- Open Letter to STD Bus Manufacturers
- Patching Turbo Pascal
- Choosing a Language for Machine Control
- Better Software Filter Design
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 1
- Using the Hitachi hd64180: Embedded Processor Design
- 68000: Why use a new OS and the 68000?
- Detecting the 8087 Math Chip
- Floppy Disk Track Structure
- Double Density Floppy Controller
- ZCPR3 IOP for the Ampro Little Board
- 3200 Hackers' Language
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 2
- Non-Preemptive Multitasking
- Software Timers for the 68000
- Lilliput Z-Node
- Using SCSI for Generalized I/O
- Communicating with Floppy Disks: Disk Parameters & their variations
- XBIOS: A Replacement BIOS for the SB180
- K-OS ONE and the SAGE: Demystifying Operating Systems
- Remote: Designing a Remote System Program
- The ZCPR3 Corner: ARUNZ Documentation

Issue Number 32:

- 15 copies now available -

Issue Number 33:

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

Issue Number 34:

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

Issue Number 35:

- All This & Modula-2: A Pascal-like alternative with scope and parameter passing.
- A Short Course in Source Code Generation: Disassembling 8088 software to produce modifiable assem. source code.
- Real Computing: The NS32032.
- S-100: EPROM Burner project for S-100 hardware hackers.
- Advanced CP/M: An up-to-date DOS, plus details on file structure and formats.
- REL-Style Assembly Language for CP/M and Z-System. Part 1: Selecting your assembler, linker and debugger.

Issue Number 36:

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

Issue Number 37:

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

Issue Number 38:

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

Issue Number 39:

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

Issue Number 40:

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

Issue Number 41:

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

Issue Number 42:

- Dynamic Memory Allocation: Allocating memory at runtime with examples in Forth.
- Using BYE with NZCOM.
- C and the MS-DOS Screen Character Attributes.
- Forth Column: Lists and object oriented Forth.
- The Z-System Corner: Genie, BDS Z and Z-System Fundamentals.
- 68705 Embedded Controller Application: An example of a single-chip microcontroller application.
- Advanced CP/M: PluPerfect Writer and using BDS C with REL files.
- Real Computing: The NS 32000.

Issue Number 43:

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

Issue Number 44:

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

Issue Number 45:

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

Issue Number 46:

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

Issue Number 47:

- Controlling Stepper Motors with the 68HC11F
- Z-System Corner: ZMATE Macro Language
- Using 8031 Interrupts
- T-1: What it is & Why You Need to Know
- ZCPR3 & Modula, Too
- Tips on Using LCDs: Interfacing to the 68HC705
- Real Computing: Debugging, NS32 Multitasking & Distributed Systems

The Computer Journal Back Issues

- Long Distance Printer Driver: correction
- ROBO-SOG 90

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

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

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

Issue Number 51:
- Introducing the YASBEC
- Floppy Disk Alignment w/RTXEB, Pt 3
- High Speed Modems on Eight Bit Systems
- A Z8 Talker and Host
- Local Area Networks—Ethernet
- UNIX Connectivity on the Cheap
- PC Hard Disk Partition Table
- A Short Introduction to Forth
- Stepped Inference in Embedded Control
- Real Computing, the 32CG160, Swordfish,
- 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
- Getting Started in Assembly Language
- The NZCOM IOP
- Z-BEST Software

Issue Number 54:
- 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)
- 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
- 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
- Introduction to Forth
- DR. S-100
- Z AT Last!

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

Issue Number 59:
- Moving Forth
- Center Fold IMSAI MPU-A
- Developing Forth Applications
- Real Computing
- 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
- 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
- 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
- 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
- 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
- Small Systems
- Mr. Kaypro
- IDE Drives Part III

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

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

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

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

Issue Number 69:
- Small System Support
- Center Fold: S-100 IDE
- Z-System Corner II
- Real Computing
- PC/XT Corner
- DR. S-100
- Moving Forth Part 6
- Mr. Kaypro

Issue Number 70:
- Small System Support
- Center Fold: Jupiter ACE
- Z-System Corner II
- Real Computing
- PC/XT Corner: Stepper Motors
- DR. S-100
- Multiprocessing Part 5
- European Beat

Issue Number 71:
- Computing Hero of 1994
- Small System Support
- Center Fold: Hayes 80-103A
- Power Supply Basics
- Real Computing
- PC/XT Corner: Stepper Motors
- DR. S-100
- Moving Forth Part 7
- Mr. Kaypro

SPECIAL DISCOUNT

15% on cost of Back Issues when buying from 1 to Current Issue.
10% on 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

Items: _____

_____

_____ Back Issues Total _____

_____ Shipping 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 or PC/XT?

The Computer Corner

By Bill Kibler

As I was going through my old mail, I came across a couple of catalogs from DYNACOMP. They had been sent to me some time back, one with a note on it saying "your readers might like to know that all the programs in this catalog are still available."

So I went through the catalog #29 and the newer one #38. The old one did have lots of CP/M, NorthStar, Apple, Atari, TRS80, and just about anything (even 8") listed. The newest was mostly MSDOS and MAC, plus a few Apple programs. I noticed they carry the Toolworks C/80 Version 3.1 for $49.95. That got me curious if this was true and so I called the 800-828-6772 number and asked. Sure enough they are still able and willing to pull out anything from the warehouse and ship, including NorthStar. The person said he still uses his Altair. I quickly sent them a complementary copy with ad rates and figured everyone should call them and get one of their old catalogs.

Now some of the old programs are public domain, but they also have many of their own items as well. They may be one of the few remaining original dealers still selling programs. They do have a little bit of everything and so if your looking for programs to run on the old machines here they are. I also noticed they have the CP/M collections, but what was unusual was the Atari, Antic, C64, and Piconet libraries. I am not sure anybody else is still offering those collections. What a find.

PLC Hacking

I have been testing PLC transfers and operations, mainly with a GE 90-30. This is their small end unit and works ok. I say ok, since their performance at moving data into and out of their ladder area is a bit slow. What our application needs is to be able to move a fairly large amount of data into the ladder from a non-standard remote platform.

Now all PLC vendors have some method or other for doing this. We used the Omron version, and have looked at the PLC-Direct as well. My analysis so far is the "Basic" expansion cards all seems to have the same problem, slow. What the problem appears to be is the way in which they talk to the regular ladders memory.

I assume the designers wanted a pretty secure and crash proof design and as such have chosen to use some form of serial communications on the back plane to transfer data. If the back plane was memory mapped and thus any application could read or write the memory location, errant programs might turn all outputs on and thus cause major problems. A slower more difficult process probably seemed safer and easier to limit.

Those limits also make applications like ours almost impossible. A number of Standard Bus vendors have PLC expansion cards that do sit on the memory Bus and from what I gather they don't seem to have any problems. Understand that these applications have several CPU's with each running their own program. Having one of the CPU's go astray is possible, but would normally be found in the startup and development stage.

To me the problem is more of the proprietary concept that many vendors have about their products. If too much about how it works is known, others would make cards and products that might fit into and on their product. The idea is lost sales, but in our case that design limit might mean we look elsewhere for the complete system.

PC Invasion

What I am starting to see is the invasion of the PC based systems into the PLC market. No longer is a manufacturer wanting just a standalone PLC, instead they want PLC functions in a member of the entire computing network of machines. The idea is being pushed by just in time production, where only what is needed is made and shipped as requested. The tying of machines together helps sales requests become actual PLC ladder changes. But that requires PLCs to talk and understand accounting data.

The answer has been putting PLC adapter cards into PC machines all talking over a common RAM area. A network based program talks to the accounting program and puts requests into the PLC's RAM area. The ladder sees these requests and knows to produce ten RED widgets next as they were just ordered. If the PLC was controlling the paint robot all is well, if the PLC was actually a machining station then you got problems.

All humor aside, the idea is not new, but just that the hardware is catching up to the desires of the companies. The power of PC's and cost of interfacing is dropped so that very powerful and cost effective solutions are now possible. Refinement and perfection of the software is what is needed next. Our application in fact might be helped by these larger pushes. Since our clients are a few years behind cutting edge, our options are limited. The more they see others having suc-

cessful projects, the better our chances are for using other designs, like PC based PC104's.

PC104

PC104 format is really starting to take off. I haven't had a chance to talk to Z-World and see how their Z180 based PC104 product is doing, but I hope well for them and us. The PC104 is a small square card that has the PC bus ISA standard interface on it, in the form of a header strip. Thus it is a mini PC clone on a very small card. All designs, programming and concepts of usage can be tested and developed using regular PC clone boxes. Then when the project is ready, it gets moved to only the type and amount of very small hardware needed to run the project.

What you gain is lowered development cost and smaller overall hardware cost. The packaging can be very small and yet the power of a 486 machines is possible in a four inch cube. For me, the idea is to use the PC104 Z180 platform for some 8085 projects. I should be able to move the base code from the STD BUS platform, to the PC104 with little to no program changes. A few I/O addresses and some interrupt labels would need changing, but mostly the code would not be effected. If at some later time, the Z180 is too slow, I replace it with a 386 CPU and run my code through a 8080 to 8086 converter and try again. I suppose you could use one of the Z80 emulators on a 486, but then for a single application, code conversion seems more appropriate.

The Hooks

What this boils down to is making sure you have a well rounded understanding of PC's and platforms in general. As the next few years start to unfold, we are going to find a larger mix of platforms in use, not less. We are seeing clients who want to upgrade, but have limited funds. In years past, you might just have thrown several programmers at the project and produced a new version on a new platform.

Now the user wants just some speed assist or the number of units increased, maybe several units tied in one network. The code, platforms, and interfaces of old often took a long time to develop, programmers are gone who knew how it worked, as well as fixing the old hardware platforms is a major problem. That is where I see Z-World hooking in with their PC104 Z180s. They can save the code mostly intact and yet put it on a new hardware platform that will be around for some time.

For programmers and hardware hackers, it means having skills and knowledge about these old systems and designs, as well as understanding the new ones. I work often now with, STD BUS, talking to embedded 8051's, talking to PLC's that may be feeding summary data to PC's on a network. When a problem develops you need skills and understanding of all the processes in the system. No longer can a technician or programmer just know one programming language or hardware platform. My last work involved Tandem mainframes, talking to PC's over a LAN, that passed data to 68000's on an PC expansion bus running assembly language.

READ ON

For readers of *TCJ* I see what we are doing as vital to you and your projects. If you look at what other magazines are doing, they have staked out some small part of the market. We on the other hand try and focus on giving you the skills to fix any problems that might arise on any platform. I am seeing the "platform independent" term everywhere these days and feel good that we were pushing it before it became the in thing.

However we look at things, one point does surface, knowledge of PC based designs is getting very important. My PC/XT person, Frank Sergeant, has been very busy paying bills and finishing his master's studies. He needs help! So I am again soliciting more articles on PC/XT projects. One project in the works is an article on "watch dog timer." The author is doing some polishing up that I suggested (mainly a little explanation of what and why you need a watch dog timer) and should be available in a later issue.

Now again I must belay all fears that *TCJ* will become a PC only magazine. This of course will happen if everybody drops their subscription, but assuming our readers continue paying, I have plenty of articles on small systems to keep us reading. Like this issue on embedded systems, I would like to have a few specials that do fill in the gaps about how PC/XT's work and can be used for other projects. To do this however I need your articles. Please consider however when sending articles to me that my time is very limited and I often fall very far behind in catching up with you. Things can sometimes sit for several months before I can send you a note. Please be patient with me.

Lastly on the topic of articles, a few words about what I want and hopefully what you are looking for in an article from *TCJ*. A main feature of *TCJ* articles is platform specific and independence. To explain that last sentence is to say that our readers are interested in learning about the platform (or software) in greater detail, but they also want to see how that knowledge can be used on their own platform. A good example of this has been our series on the IDE drives. Here we have talked about how it is implemented on the PC/XT platform and what the protocol is all about. Then we presented hardware for interfacing to Z80 machines which allowed others to adapt for non-Z80 machines. So it goes like this: history, theory, details, and adaption of the project.

Last Word

It seems I haven't done a "what is happening" in computers column for sometime, so this was it. To review would be to say, look for sales of bigger machines to slack off as people tend to keep what they have, more inter-hooking to develop, and some big vendors to step into the only growing part of computers - embedded control where 8051's and 68HC11's will fight it out for dominance. As usual time will tell if I am near or far from the market. Till then however, keep hacking. BDK.

TCJ CLASSIFIED

CLASSIFIED RATES!
$5.00 PER LISTING!

TCJ Classified ads are on a prepaid basis only. The cost is $5.00 per ad entry. Support wanted is a free service to subscribers who need to find old or missing documentation or software. Please limit your requests to one type of system.

Commercial Advertising Rates:

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

Send your items to:
The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535

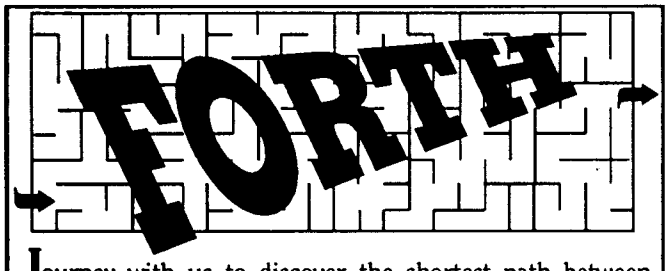
Historically Brewed. The magazine of the Historical Computer Society. Read about the people and machines which changed our world. Buy, sell and trade "antique" computers. Subscriptions $18, or try an issue for $3. HCS, 2962 Park Street #1, Jacksonville, FL 32205

HELP! Looking for someone to install SPELL-CHECK in my obsolete but functional (16 bit?) Televideo 806H (?) wordprocessor running CP/M86. It's presently configured for WordStar Ver. 3.30, 1979. Please call Bob at (916) 791-1914, Granite Bay (Roseville) CA.

Wanted. Books/Manuals/Info on a Structured Design SD20/24 PAL development system. Has built in monitor, but unable to read pre-programmed PALs. Anyone know if these people still in business? Contact Bill at *TCJ* (800)424-8825.

TCJ ADS WORK!

Classified ads in *TCJ*
get results, FAST!
Need to sell that special older system - TRY *TCJ*.
World Wide Coverage
with Readers interested in what **YOU** have to sell.
Provide a support service,
our readers are looking for assistance with their older systems - all the time.
The best deal in magzines,
TCJ Classified
it works!



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.

DIBs Electronic Design

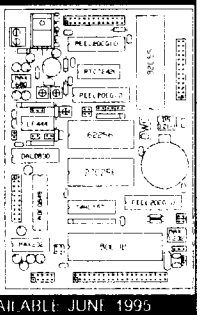
Dave Baldwin

6619 Westbrook Dr.
Citrus Heights, CA 95621

Voice/Fax (916) 722-3877
DIBs BBS (916) 722-5799

VERSATILE 80C32 BASED SINGLE BOARD COMPUTER

The DC8032-1 is the first in a series of versatile single board computers based on the 80C32/52 family of micro-controllers. The DC8032-1 and DC8032-2, available later this year, has been designed to provide most, if not all, of the functions required of a dedicated controller. Standard features include 32K of RAM & EPROM, A/D & D/A, real time clock, 36 digital I/O lines, watch dog timer and a centronics parallel printer port. Extended BASIC with 16 additional commands and a debug minitor with 12 commands is also included.



SPECIAL PRICING FOR STUDENTS

MODEL DC8032-1 AVAILABLE JUNE 1995
$200.00 ASSEMBLED, $175.00 KIT + $10.00 S&H
SEND CASHIERS CHECK, MO, OR C.O.D. TO
D.C. MICROS 1843 SUMNER CT.
LAS CRUCES, NM 88001 PH: (505) 524-4029