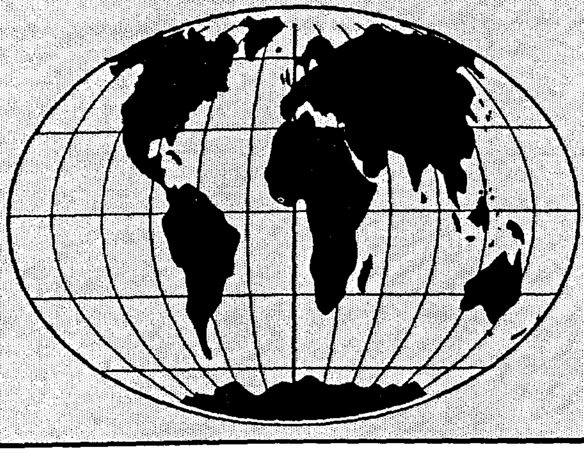
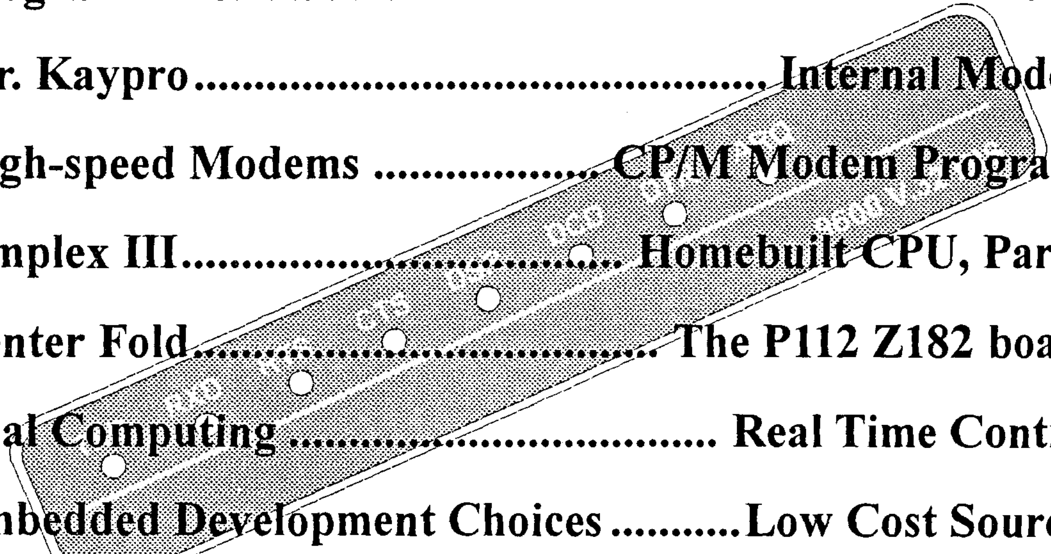
Small Scale Computing Since 1983 Supporting the *Trailing Edge of Technology*



# The Computer Journal

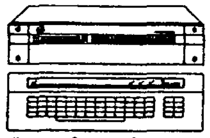
Issue Number 79 Fall 1996 US $4.00



ISSN # 0748-9331 *Hands-on Hardware and Software*

ENGINEER ALERT

## AWESOME Z80 COMPUTER



Perfect for Project or Product

**INCLUDES**

KEYBOARD (17"x7"x2.5")
- Durable Plastic, 66 Keys
- 2 Connectors for Joysticks

ENCLOSURE (18.5"x11"x4")
- Contains Power Supply & Motherboard
- Front: ON/OFF & Reset Switches, LEDs
- Back: 7 Connectors & 4 I/O Panels
- Metal Case, Effective RF Shielding

POWER SUPPLY
- 40 Watts AC/DC Switching
- +5V@2.5A; +12V@2.0A; -12V@0.1A

Beautiful Charcoal Grey Design!

Special Computer Journal Price!

MOTHERBOARD
- 4 MHz Z80A
- 64K Dynamic RAM
- 4K/8K EPROM
- TMS 9918 Video Display (16K)
- AY-3-8910 Sound Generator
- 4 Slots for I/O Cards

ON BOARD I/O
*Audio Output
*Video Output
*CH 3/4 RF Modulator Output
*External RF Input
*Keyboard Interface
*Printer Interface (Parallel)
*Serial Interface (RS-422)
*4 Card Slots

INCREDIBLE PRICE/FEATURE RATIO

| UNITS | PRICE |
|---|---|
| 1 | $250.00 |
| 2 | $200.00 |
| 4 | $150.00 |
| 8 | $125.00 |

KART COMPUTERS
P.O. Box 144
Millbury, Ma. 01527
(508) 755-9778

**SUPER SPECIAL**
INTRODUCTORY
1 UNIT **PRICE**
**$125.00**

## Hiding in Plain Sight...

Some of the most interesting, challenging programming is being done outside the prevailing paradigms. It's been this way for years, and some companies regard its SPEED, COMPACTNESS, EFFICIENCY and VERSATILITY as their private trade-secret weapon.

It has penetrated most of the FORTUNE 500, it's a veteran in AEROSPACE, it's in SPARC WORKSTATIONS, and it's how "plug and play" is implemented in the newest POWER PCs. In fact, it's lurking around a lot of corners.

It's FORTH. Surprised? Call now to subscribe* and learn more about today's Forth.

***Forth Dimensions***
**510-89-FORTH Fax: 510-535-1295**

*Ask for your free copy of "10 Ways to Simplify Programming"

## *Kibler Electronics*

Serving the
Industrial Electronics Community
since 1978

Specializing In
Hardware Design and
Software Programming

*Previous Projects include:*
PLC ladder programming (15,000 lines)
8051 Remote I/O using MODBUS
6805 Instrumentation Controller
68000 Real Time Embedded Operations
NETBIOS programming and Debugging
Forth Projects and Development
HTML Design and programming
Articles, Training, and Documentation

Bill Kibler
*Kibler Electronics*
P.O. Box 535
Lincoln, CA 95648-0535
(916) 645-1670

***e-mail: kibler@psyber.com***
***http://www.psyber.com/~kibler***

## 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)
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) 965-7046 BBS

# TCJ The Computer Journal

Issue Number 79, Fall 1996

**The Computer Journal**

**Founder**
Art Carlson

**Previous Publishers**
Bill D. Kibler
Chris McEwen

**Editor/Publisher**
Dave Baldwin

**Technical Consultant**
Bill D. Kibler

**Contributing Editors**
Herb Johnson
Charles Stafford
Brad Rodriguez
Ronald W. Anderson
Helmut Jungkunz
Frank Sergeant
Richard Rodman
Tilmann Reh

*The Computer Journal* is published six times a year and mailed from *The Computer Journal*, P. O. Box 3900, Citrus Heights, CA 95611, (916) 722-4970.

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

Entire contents copyright © 1996 by *The Computer Journal* and respective authors. All rights reserved. Reproduction in any form prohibited without express written permission of the publisher.

**Subscription rates** within the US: $24 one year (6 issues), $44 two years (12 issues). Send subscription, renewals, address changes, or advertising inquires to: *The Computer Journal*, P.O. Box 3900, Citrus Heights,CA 95611-3900.

# Editor's Column

## And in this issue...

Well, I'm late again, but I'm still here. David McGlone's Z-Letter has ceased publishing, and Chuck Stafford and Herb Johnson have stopped doing regular columns to pursue other interests although we may see occasional articles from them. TCJ has taken over Chuck's Kaypro business and the TCJ Kaypro catalog is on the TCJ Store page.

I want to thank those who responded to the last flyer. All those renewals made this issue possible. TCJ counts on your renewals to get each issue printed. Check the date on your address label to make sure you're not going to miss anything.

I'm going to try to get the next issue out by the end of January. That will mean sending in renewals over the Christmas season, always a bad time, but that's what it takes to get the next issue printed.

Because TCJ focuses on projects that one person can 'do-it-themselves', that means we're kind of at the low end of computing, both in technology and cost. After all, there's no way you're going to spend $500 or more to get software for a hobby project board that cost less than $100 (less than $25?). There are, however, quite a number of small businesses that operate pretty much the same way. (I've done work for a few of them.) No money for tools, hardware or software.

My goal is to try to get articles to help both the hobbiest and those stuck in low/no budget situations along. Part of that is making sure that the authors have actually used and tested the projects they're writing about. Another part is providing source code and the schematics so you can 'do-it-yourself'. And software is made available to help you such as the 6502 assembler Doug Beattie provided with his 6502 DIY board article. You can download the software from the TCJ Web page or the TCJ/DIBs BBS. This is as close to 'free' as I can make it.

Towards that end, I'm always looking for articles, suggestions, and requests. Let me know what you're looking for. Your project may end up as a TCJ article. And check the TCJ Back Issues too. It may have already been done.

Dave Baldwin
Editor/Publisher

**The Computer Journal**
800-424-8825 / 916-722-4970
Fax: 916-722-7480
BBS: 916-722-5799
Web Page: "http://www.psyber.com/~tcj/"
Email: tcj@psyber.com

This must be the **Modem** issue! Frank Sergeant leads off with **The PC Serial Port in Forth**. Next is David Goodenough with **The AT Modem Commands** in Program This! This is a very complete article. Every time I would think of something else that should be in it, it would be there when I re-read it. Good job.

Mr. Kaypro tells us how to install an Internal Modem in your Kaypro and Terry Hazen talks about **High Speed Modems and CP/M** which is about using 14.4k modems with CP/M modem programs.

Dave Brooks has two articles this time. Part 2 of his **Simplex III** series is here and the **Centerfold** is his P112 Z182-based CP/M board.

**Real Computing** by Rick Rodman covers Real-time control in his home environment and includes some more sample code for network communication. In **Embedded Development Choices**, Bill Kibler responds to a reader's letter that requested some info on low cost development.

Ron Anderson cleans up a few details in **Small System Support** and Bill Kibler tells about this summer and things in progress. As always, the TCJ feature and listings are at the back including the **TCJ Store**, the **Support Groups**, and the **Back Issues** listings.

**Future Issues**

I have a number of articles lined up for the next few issues already. Lots of DIY stuff with schematics and source code.

Tilmann Reh (who designed the GIDE kits and many other projects) has sent me an article on his Eprom Simulator. Hal Bowers is sending an article on his B/P Bios for the Ampro Z80 Little Board, the Micromint SB180, and Dave Brooks P112.

I also have an article in progress on a homebuilt 6809 board from Frank Wilson with source code for both the 6809 and for the PC programs for downloading and communications.

We're going to start covering a number of common I/O devices that come up again and again.

We'll cover interfacing the standard LCD character displays with source for at least the Z80 and the 8031. Maybe I can get someone to provide code for some other CPU's.

We're going to revisit stepper motors also. There have been a number of articles about simple stepper interfaces. We'll show you a more sophisticated system and the power supplies required to really get them moving.

# READER to READER

Letters and News

All Readers

MINI Articles

From: Herbert R Johnson
<hjohnson@pluto.njcc.com>
To: tcj@psyber.com

The Doctor is retired but still sees patients...

---

Although I'm not writing my "Dr.S-100" column, I'm still working the S-100 turf via the Internet and correspondence. The Usenet's comp.os.cpm maillist is more active than ever, with several messages a day. That's quite a lot of traffic for an OS that is about 18 years old! Many people are still looking for S-100 stuff and support, or asking about the S-100 bus. I recommend looking over the CP/M FAQ (note to editor -where is this archived? - Herb) and The Computer Journal's Web page, as well as TCJ's "paper archive" of back issues of my column and many other resources.

Most S-100 interest these days is in Compupro equipment. The "Cadillac" of personal and industrial computers of the early 1980's, Compupro produced 68000, 80286 and 80386 systems that outperformed IBM PC's of the era. Now that these are surplus, many people who coveted these now want one cheaply; and folks who once developed on these system are letting them go to a good home (sometimes via my basement) rather than landfill.

Despite the Internet, I still get some paper mail. Some time ago, I contacted Rlee H. Peters, a retired AF officer and collector of S-100 stuff, who wrote a letter on S-100 stuff in David McGlone's Z-Letter. Rlee kindly sent me his list of stuff, and recently a followup letter on my TCJ columns. His efforts are encouraging to all of us, and interested readers should contact him directly. One reason I keep a hand in the S-100 world is that it brings me in touch with good people from around the world.

Herb Johnson, aka "Dr. S-100"
hjohnson@pluto.njcc.com

Letters to the Doctor - from Rlee Peters

---

"I was just rereading your column in TCJ #78, where I noticed that one of your customers was looking for 8" 10 Meg Bernoulli disks. I upgraded to the 5-inch 150 Meg version and have 16 of the 8-inch disks left. [Note: I have some of these too - Herb]. Three are still in plastic, two loaded with CP/M 80 files, and the rest new or formatted. I also have an S-100 interface and IBM PC bootable interface, and a cleaning disk. No documentation.

"The customer who wants to put an 8" [soft sectored] drive on his North Star is well advised to use his [Morrow] DJ-2D controller for that; I have used this combo for many years... UNFORTUNATELY you can boot one or the other or swap easily but not copy 5 to 8 easily unless you use the software my brother wrote to do that. I just finished rewriting the [Morrow] E4 BIOS to talk to the North Star disk but have not debugged it yet. The other approach would be to change to the Morrow DJDMA board which supports 5" hard sector, 5" soft sector and 8" soft sector, using the E4 BIOS. I have this in my North Star now with a Morrow HDDMA board with a 20 Meg harddrive....I have docs, boot disks and BIOS code. This setup only runs CP/M. If he wants to run NS DOS there is a CP/M program to run it...and a program to copy files between them, [in an article I have copies of].

"For the guy with the Compupro 68K boards, I have the documentation on them and DR CP/M 68K..."

Rlee has a number of S-100 cards, some in quantity, including Morrow Wunderbuss motherboard cards, Morrow 256K memory cards; "and a supply of 8" double sided drives, some even half height. Got lots of kinds of boards!...Your column is always interesting, keep up the good work!"

(signed) Rlee H Peters,
1600A N Sierra View, Ridgecrest CA
93555-2438 LT Col USAFA
(Ret) GS-12 (Ret)

---

From: Harold Bower
<HalBower@msn.com>
To: tcj@psyber.com

Dave,

Just received your note to TCJ subscribers, and will be sending you an early renewal because I don't want to see the last bastion get in too much trouble. You have probably also heard that David McGlone is folding his Z-Letter and business.

This past Spring, I took the most recent issue of TCJ with me to the Trenton Computer Fest in New Jersey, and your newly-adopted banner created a big splash.."Supporting the Trailing Edge of Technology". I showed off a modified YASBEC laptop which was modified to include "High-Density" floppy support by replacing the Western Digital 1772 controller with a National DP8473, and a monochrome

VGA LCD kit from Earth Computer Technologies. The entire system runs from 12 vdc and had a 1.76MB floppy, 45 MB Conner SCSI drive and 256k of RAM. The operating system was an adapted version of my B/P Bios and the Banked ZsDos2. The system sees much use here at home, normally on my lap in the living room easy chair.

This past Summer, David Brooks released his P112 Z182-based system on the footprint of a 3.5" floppy. I have adapted the B/P System to it, designed and built a SCSI controller, and have it running as well. It is quite an impressive little board with several "modern" features which some might like to toy with such as Flash ROM, Battery-backed-up RAM for storing parameters, etc. It might even be possible to adapt one of the Earth Tech LCD systems into it as well (already bought one, and should start working on it after Christmas).

Several years ago when TCJ was in the process of moving to your location, I had sent the first installment on an article on the B/P Bios system (with banked ZsDos and Command Processor), but it apparently got lost in the process. Would you still be interested? If so, I will try to locate the sources if you can still accept WordStar 4 text, since that is all I use on the 8-bitters (WordStar 7 on the clones).

Keep up the good work.

Hal

*And from another of Hal's messages:*

For sale, I have the Banked and Portable (B/P) Bios which consists of source for selected hardware platforms, along with a large banked version of ZSDOS (ZsDos2) which features Z-System style command line parsing, and hashed directory capability for *speed*, and a banked extension and enhancement of ZCPR 3.4 which features many commands normally placed in RCPs. Versions are available for:

MicroMint SB-180
MicroMint SB180FX
YASBEC
Ampro Little Board 100 (with
Terry Hazen's MDISK)
D-X Designs Pty Ltd P112

The price is $69.00 + $3.00 S&H. Deliverables consist of a 150+ page manual, and one or two disks depending on the desired disk configuration and version of the target computer. Wherever possible, I try to provide bootable disks, with pre-configured banked and non-banked images. Support utilities are also provided to support the new features such as a rather extensive SCSI Hard drive diagnostic (drives up to 1GB are handled), and sample source for interfacing.

E-Mail: HalBower@msn.com

Address:
Harold F. Bower
7914 Redglobe Ct.
Severn, MD 21144-1048

Phone (no calls after 10PM Eastern time, please)
(410) 551-5922

---

From: William Cook
<wcook@nni.com>
To: tcj@psyber.com
Subject: XT's

I stumbled across your news letter and found it very infomative. I made it required reading at my company for new recruits. (We sell electronics) as many would not remember the bad old days of computing. Do you have a page that keeps updating, if so let me know, I'd like to read more of your work.

The article in ref was on XT's.

Thanks, Bill Cook

*And, thank you, Bill. I'm trying to figure out how to make TCJ required reading everywhere.*

---

Hi Dave,

This is Ian Blythe, I have received the trial issue of TCJ, and I will be sending you a subscription as soon as I can resolve my financial situation (4/5ths of my immediate family have birthdays in May/June... and the clutch is slipping in my car... :(

I'd like to take advantage of the discount on ordering backnumbers with a subscription. I am particularly interested in the IDE articles, I would like to build an IDE interface for my homebrew 6809 system. I know that the GIDE stuff is for the Z80, but if any of the previous issues covered IDE in a more general way (ie an interface guide for _any_ MPU) I would appreciate it if you could point me in the right direction.

I would be happy to write an article for TCJ, something on the lines of: Student days, no money, lots of SS50 stuff and S100, want a computer, no money (still) so build it yourself, more interesting and fun!

"Find application note by Motorola on 6883 SAM, add in a 6522 VIA and 6551, pop in a 20L10 PAL to run the clock and decoding, page in 4 EPROMs, and 64K DRAM and you've got a standalone 6809 computer. Type in Assist09 and modify it to suit, find stuff for Flex OS, put in an FDC controller, put Flex bootup into EPROM, and modify assist09. Type FLEX and the system boots into a full 64K DRAM disk system. Build in paged expansion bus, plan EPROM programmer, RAM disk, CRT interface, but then get a job, bought a motorbike, got a family, sold the bike and bought a car, built an IBM PC clone...changed country and 20 years later I come back to my Flex system."

Would this be of interest?

best regards,

Ian
--
Senior Technical Writer,
SGS-THOMSON Microelectronics

*If it's DIY with source code and schematics, TCJ is interested.*

# PC/XT Corner

By Frank Sergeant

Regular Feature

PC Serial Port

In Forth

Among other things, I maintain a medical accounting package written in Clipper. The first major development work I did on it was to add electronic claims handling. This involved controlling serial ports and modems on IBM PC clones, usually '486 or Pentium-based. This article discusses how I handle the serial port from Pygmy Forth.

## Application Summary

I wrote the data collection and extraction routines for the electronic claims in Clipper, a dBASE-like, C-like language. I used Clipper for this because the rest of the application was written in Clipper. But, since Clipper can shell to DOS to run external programs, I took advantage of this to write the transmit module completely in Forth.

When it is time to send claims to the clearing house, the Clipper program extracts the data and writes it to a text file, then runs TRANSMIT.COM. TRANSMIT.COM is a version of Pygmy Forth in disguise. The Clipper program passes the file name and phone number to Pygmy. Pygmy dials the phone, transmits the file to the clearing house record by record, captures the returned report, hangs up the phone, and returns control to the Clipper program.

## Why Use Forth?

For me, the real question is why continue to use Clipper? The answers to that are inertia and fear. I would prefer the application be entirely in Forth. Unfortunately, when I took over the application, it was a monster. I have been afraid of losing too many "business rules" which are captured in the current application, but not documented clearly. I suppose it might have been (and might now be) better to do a complete rewrite in Forth. Since my income from this project doesn't seem to justify such an effort, I'll limp along with Clipper for now, gradually cleaning up the code as I go, and look for independent modules that can be turned over to Forth.

In hindsight, I am very pleased to have used Forth for this module. One of Forth's strong points is the speed of testing. This is due to the direct, straight forward, interactive access to the machine that Forth provides plus the small resource requirements of Forth. In Forth, I can test something quickly from the keyboard and then poke around in memory if I need to examine something. A recompile of one or three blocks of Forth source takes a second or so. A small recompile and link for Clipper takes perhaps 30 to 50 times as long. By that time, I might have lost my train of thought. Since I decoupled the Forth module from Clipper by having Forth transmit a text file that Clipper had previously created, I could make the Clipper program create the file once, then bring up Forth and stay in Forth while testing, without needing to run the Clipper program each time.

## The PC Serial Port

Fortunately, the basic serial port is fairly standard between all versions of IBM PCs. There is a high-level of compatibility from the oldest PC/XT to the newest Pentium-based machine. The original serial chip was the 8250. Subsequent serial port chips tend to look exactly like an 8250 except they often have additional capabilities which old software simply ignores. The major enhancement over the original 8250 is the addition of a FIFO. This on-chip FIFO (first in, first out) is a buffer for storing incoming serial characters. This reduces the chance of losing characters if the software is tied up when a new character arrives. If the FIFO is not turned on, the newer serial chips behave as if they were original 8250s. Some of the older enhanced chips have unreliable FIFOs which should not be turned on. I'll show the code for turning the FIFOs on and off and for determining whether to trust the FIFO.

The serial chip is controlled by writing bit patterns to its various control registers and reading its status registers. In particular, the status registers indicate when an incoming serial character is available and when the transmitter is ready to accept a character to transmit.

## Modems versus Serial Ports

The modem is connected to a serial port. The program talks to the modem via the serial port. (The modem, of course, talks to another modem via the telephone line.) Some PC modems are "external." They sit in a box connected to the PC via a serial cable. Other modems are "internal." These are on a card plugged into the PC's motherboard. These cards have built-in serial ports. Either way, to talk to the modem, the program must control a serial port.

## Interrupts

The serial chip could be handled by polling, without requiring the use of any interrupts. Software to do this stays in a loop reading a status register on the serial chip to determine when a new incoming character is available or when it is ok to write a new outgoing character to the serial port.

This can be a very workable method in some situations. The problem is that often other things must be done by the software; it can't spend all its time checking on the serial port. Interrupts help solve this problem by allowing the software to do its other work until the serial port needs attention. Then, the serial chip interrupts the CPU and invokes a special interrupt handler (software routine) that "services" the serial port. You tell the serial port which events, if any, should cause an interrupt by writing bit patterns to a control register in the serial chip.

I usually set the serial port to generate an interrupt only when an incoming character is ready. When transmitting a character, the software polls the serial chip until the chip is ready to accept the character. In chips without a FIFO, if a second character is received before you read the current one, the second character overwrites the current character. The whole point of using an interrupt on serial input is to prevent the loss of incoming data that might occur if too many bytes arrive at the serial chip before your software can read them all (because your software is busy elsewhere). FIFOs also help prevent lost data because the FIFO buffer can hold a number of characters (such as 16) before losing any. You can tell the serial chip, for example, to generate an interrupt whenever the FIFO is half full. It also generates an interrupt when one or more characters are ready but no new characters have been recieved for four character periods. The FIFO has the added advantage of reducing the overhead due to the interrupt. It is more efficient to read 8 or 10 characters per interrupt than to read a single character per interrupt. I set the interrupt point at half full (8 characters) so there is still a bit of room in the FIFO if more characters come in while responding to the interrupt. Before returning, the interrupt routine "drains" the FIFO of all the characters that are available.

## Quick In and Out

The interrupt handler should run as quickly possible. At 9600 bps (bits per second), a new character arrives approximately every millisecond (1,000,000 microseconds/sec divided by 960 bytes/sec = 1042 microseconds per byte. I divide by 960 because there are typically 10 bits per character: a start bit, 8 data bits, and a stop bit, thus 9600 bits per second is 960 bytes per second.) Imagine the pickle you'd be in if your interrupt handler took longer than 1042 microseconds to handle each character. Well, the problem is worse than that. Other events going on in the PC generate interrupts which require handling. It spells trouble if any of those interrupt routines are real CPU hogs. So, get in quick, do the minimum work you must inside the handler, then get out. Any processing that isn't time critical should be done outside of the interrupt routine.

## FAQs

Various FAQs (Frequently Asked Questions documents) are available on the internet describing the hardware and software aspects of PC serial ports and modems. See Chris Blum's ftp://ftp.phil.uni-sb.de/pub/people/chris/The_Serial_Port and John Navas's http://web.aimnet.com/~jnavas/modem/faq.html. Another good source is a data sheet on the serial chip. Since the newer chips are backward compatible with the older chips, a data sheet for almost any 8250 or 16550 chip will be useful.

I found the Chris Blum FAQ very interesting. He gives the details about how to test whether the FIFO is reliable, etc. Apparently the interrupt handling can be tricky. He offers what he considers a bullet-proof method of handling the interrupts and that is what I based my code on.

## The Serial Port Code

See the listings for the source and shadow blocks. I'll leave it to the shadow blocks for most of the documentation and just discuss some of the more interesting details.

The word `SER-IN?` is analogous to `KEY?` in that it tells whether a character is available from the serial port. Rather than just returning a true or false flag, `SER-IN?` returns a count of how many characters are available. If you care about the exact count, you've got it. Otherwise, just use the result as a true or false flag.

The word `SER-IN` is analagous to `KEY`. It returns the next available character from the serial port. It doesn't read the serial port directly. That is done by the interrupt handler. `SER-IN` reads the `SERIAL` queue into which the interrupt handler has stuffed incoming characters.

`SER-IN` calls `SER-IN?` in a loop until at least one character is available. Then it removes that character from the `SERIAL` queue and returns it on the stack. The loop is not necessarily an endless loop. If no character is available, `SER-IN` will time out eventually with an error message. To avoid a possible time out, always call `SER-IN?` first to make sure a character is available.

`SERIAL` is the name of the queue where the incoming serial characters are stuffed by the interrupt handler. `SERIAL` is created by the word `BYTEQ:` and must have a length that is a power of 2, such as 512 or 4096.

## Serial Port Registers

The serial port registers are accessed by the 80x86 CPU's input and output instructions. `'SDATA` holds the address of the serial port's base I/O address. All the register addresses are relative to that base I/O address. `'SDATA` must be set properly for the serial port you want to use. The word `COM` takes care of this by reading the BIOS data area to find the proper I/O base address. Saying `1 COM` or `2 COM` sets `'SDATA` and some related variables to the correct values for the serial port you have specified. `COM` makes some assumptions as to the interrupt number and level. Normally COM1: uses interrupt 12 and IRQ level 4, while COM2: uses interrupt 11 and IRQ level 3. If that should not be the case with your hardware, after running `COM` you can store different values into `SINT#` and `SIRQ#` to handle special cases.

After typing `1 COM` or `2 COM` you can type `.STA` to show the status of the various registers.

## PIC

The PIC (programmable interrupt controller) chip is also heavily involved in hardware interrupts. Its two registers `PIC_CTRL` and `PIC_MASK` are defined on the same block as the serial port registers. The PIC_MASK controls which IRQ (interrupt) levels are allowed to be active. The serial port usually uses IRQ level 4 or 3. The word IRQ_MASK produces the proper mask byte for use within ENABLE_PIC and DISABLE_PIC for enabling or disabling the chosen serial port. The words PORT-ON and PORT-OFF turn on or off specified bits without affecting the other bits. This makes it easy to selectively disable, say, IRQ4 (for serial port #1) without disturbing the interrupts in use by the rest of the system.

After an interrupt occurs, you must write an "end of interrupt" value to PIC_CTRL to reset the PIC to allow it to handle future interrupts.

### Initializing the Serial Port

Now that COM can determine the base address and IRQ level and now that names exist for the various serial port registers, the serial port can be initialized to the chosen bit rate, parity, etc. See BPS, DATABITS, FIFO-ON, FIFO-OFF, and NO-PARITY. Here is an example of initializing the serial port:

```
1 COM  9600 BPS  8 DATABITS  NO-PARITY  FIFO-ON
```

Note that 115200 BPS is especially interesting on a 16-bit Forth because 115200 is too big for the 16-bit data stack. BPS handles this special case by noting that when you try to put decimal 115200 on the 16-bit data stack, the value becomes 49664 (because 115200 mod 65536 is 49664).

### Defining the Interrupt Handler

There are at least three ways to define the serial interrupt handler code.

(1) Define a handler hard-coded for each serial port base address and IRQ level combination, then pick the right one to install into the interrupt vector table. This is fast at run-time but somewhat inconvenient, since you don't usually know which serial port will be used at the time the handler is defined.

(2) Define a general purpose handler that reads the values of the 'SDATA, SINT#, and SIRQ# variables each time it runs so it will know how to address the proper serial port. This is flexible, but adds a run-time penalty which slows down the interrupt handler.

(3) Use a macro to define a custom hard-coded handler. The macro itself uses the 'SDATA, SINT#, and SIRQ# variables as it defines the hard-coded handler. Thereafter, when the handler runs, no time needs to be spent checking those variables.

I took the third approach in this project. The word HANDLER, is the assembly language macro that lays down the proper machine code. It is run only *after* 1 COM or 2 COM has selected the serial port. To keep HANDLER, to a moderate size, it invokes sub-macros such as DISABLE_PIC,, EOI,, STI,, and DRAIN,. (Note that I usually put a comma as the final character of the name of an assembly language macro to indicate that it commas code into the dictionary.)

The word BUILD-HANDLER passes a string to EVALUATE to compile the CODE word SERIAL-HANDLER at run-time. The point of putting this off until run-time is to delay its compilation until the serial port to be used is known and so can be hard-coded into the serial interrupt handler. If SERIAL-HANDLER were defined earlier, it would need various tests or variable accesses which would slow it down.

### Installing the Interrupt Handler

After the interrupt handler has been defined, it must be installed into the interrupt vector table, the serial chip must be initialized to generate an interrupt at the proper time (when an incoming character is available), and the PIC must be initialized to pass this interrupt on to the CPU. The word INSTALL-SINT does this. The related word UNINSTALL-SINT disables the serial interrupt and restores the original vector in the interrupt table.

INT-VECTOR@ and INT-VECTOR! call DOS to fetch or store to the interrupt vector table. START puts it all together to specify a serial port and build and install the interrupt handler. The example in the code is for serial port 2 and 9600 bps. Change it according to your needs.

### Testing it with DUMB

The word DUMB provides a dumb terminal for testing the serial port and modem. It is a loop that does two things. Whenever a character comes in on the serial port, DUMB displays that character on the screen. Whenever a key is pressed, DUMB sends that character to the serial port. The exception is the Esc key. This terminates DUMB and returns to Forth.

To try it out, modify START to suit your hardware and load the code and type DUMB. The simplest test of the modem is to type AT followed by the enter key while in the dumb terminal. If the modem is listening, it will respond with OK. Once you are talking to the modem, you can try fancier AT commands to turn the modem speaker on or off, to dial a number, etc.

Note, DUMB makes a very dumb terminal. For real use as a terminal program, you need to modify it to handle various terminal control codes. See DUMB2 for an improvement that drops incoming line feed characters and converts incoming carriage return characters to a carriage return/line feed pair.

### Summary

These serial port words hide the nasty hardware details and provide convenient access to the serial port and modem. This approach works for applications written entirely in Forth and also for mixed language applications where Forth is called upon to do the low-level work.

### Source Code Listing

The next three pages contain the source code in a modified format to fit in the magazine. Shadow is in the left column and source is in the right. The code is available in standard format as BLOCKS79.ZIP on the TCJ Web page and on the TCJ/DIBs BBS.

**<< SHADOW** | **SOURCE >>**

```
( serial port variables )

  SPORT#   active serial port number

   SIRQ#   hardware interrupt line used for
           active serial port

   SINT#   equivalent software int number

  'SDATA   base I/O address for the port
           ( e.g. $03F8 for com1)


( Serial port & PIC registers )
DATA     serial data register
IER      interrupt enable register
DIV-LSB  least significant byte, baud rate divisor
DIV-MSB  most significant byte, baud rate divisor
IIR      interrupt id register
LCR      line control register, its bit 7 chooses
           whether divisor or data is addressed
MCR      modem control register
LSR      line status register
MSR      modem status register
FCR      fifo control register

PIC_CTRL   programmable interrupt controller
PIC_MASK      registers

( select a serial port and create the queue)

COM  fetches the base I/O address from the BIOS
     data area, makes a reasonable guess as to
     the proper interrupt to use, and sets up
     the serial port variables.  E.g.

                1 COM    to select COM1:  or
                2 COM    to select COM2:

SERIAL  is the name of a 4096 byte queue.  Note,
        the length must be a power of two, or
        the cheap mod mask trick will not work.

( Serial input )

RESET-SER-IN  empties the SERIAL queue
SER-IN?       returns true if at least one
              character is waiting

TIMEOUT       error handling if SER-IN can't
              retrieve a character

SER-IN        fetch a byte from the SERIAL queue.
              Wait a little while, if necessary,
              but don't hang forever.


( Serial output       >SERIAL  )
SER-OUT  write a byte directly to the serial port
  hardware, but wait until the transmit data
  register is empty.  Don't wait forever, though.

>SERIAL  point the main Forth input and output
  words to the serial port.  Note >SERIAL is
  currently defined to revector only EMIT.  See
  the alternative below EXIT which also revectors
  KEY? and KEY.  The point of revectoring to the
  serial port is to allow all the usual I/O words
  (EMIT TYPE U.R etc) and the words that call them
  to work unchanged with the serial port.

( Set baud rate)

>DIVISOR  choose divisior registers
>DATA     choose data register

DIV@  fetch 16-bit divisor
DIV!  store 16-bit divisor to establish baud rate
```

```
( serial port variables )

VARIABLE   SPORT#
           ( 1 or 2 for com1 or com2)

VARIABLE    SIRQ#
           ( usually IRQ4 for com1, IRQ3 for com2)

VARIABLE    SINT#
           ( usually 12 for com1 or 11 for com2)

VARIABLE   'SDATA
           ( usually $03F8 for com1)

( Serial port & PIC registers )

: DATA     ( - port)    'SDATA @    ;
: IER      ( - port)  'SDATA @ 1+   ;
: DIV-LSB  ( - port)         DATA   ;
: DIV-MSB  ( - port)          IER   ;
: IIR      ( - port)  'SDATA @ 2 + ;
: LCR      ( - port)  'SDATA @ 3 + ;
: MCR      ( - port ) 'SDATA @ 4 + ;
: LSR      ( - port ) 'SDATA @ 5 + ;
: MSR      ( - port ) 'SDATA @ 6 + ;
: FCR      ( - port )         IIR   ;

$0020 CONSTANT PIC_CTRL
$0021 CONSTANT PIC_MASK

( select a serial port and create the queue)

: COM ( port# -)
  1 4 CLAMP ( port#)
  DUP SPORT# !   11 OVER 1 AND + SINT# !
                  3 OVER 1 AND + SIRQ# !
             $40 SWAP 1- 2* L@  'SDATA !   ;



4096 BYTEQ: SERIAL      ( define the queue)
2 COM                   ( pick the serial port)

( Serial input )
: RESET-SER-IN ( -)    SERIAL QRESET  ;
: SER-IN? ( - #items_waiting)  SERIAL Q?  ;

DEFER TIMEOUT
: (TIMEOUT ( -)
  -1 ABORT"  timeout in SER-IN "  ;
' (TIMEOUT IS TIMEOUT

: SER-IN ( - c)
  10000 FOR  SER-IN?
          IF POP DROP SERIAL Q@ EXIT  THEN
        NEXT  TIMEOUT  ;

( Serial output       >SERIAL  )
: SER-OUT ( c -)
  10000 FOR  LSR PC@ $20 AND
          ( transmitter holding reg empty?)
          IF ( c) DATA  PC! POP DROP EXIT   THEN
  NEXT ( c) DROP ."  timeout in SER-OUT "  ;

: >SERIAL ( -) ['] SER-OUT IS EMIT  ;
EXIT     : >SERIAL  ( -)
                ['] SER-IN? IS KEY?
                ['] SER-IN IS KEY
                ['] SER-OUT IS EMIT   ;


( Set baud rate)

: >DIVISOR ( -)  LCR $80 PORT-ON  ;
: >DATA    ( -)  LCR $80 PORT-OFF ;

: DIV@ ( - u)
  >DIVISOR DIV-LSB PC@  DIV-MSB PC@ $100 * +
  >DATA  ;
```

## << SHADOW | SOURCE >>

```
BPS  calculate & set divisor for given baud rate
     (bits per second).  It works for 110 300 600
     1200 4800 9600 19200 38400 57600 and 115200.
     Note special trick for 115200 since 115200
     won't fit in a 16-bit number.

eg   1200 BPS     57600 BPS    115200 BPS
```

```
: DIV! ( u -)  >DIVISOR
  DUP $100 U/ DIV-MSB PC!  $FF AND DIV-LSB PC!
  >DATA ;

: BPS ( bps -) DUP 49664 - IF  49664 1  ROT UM/MOD
  ELSE 1 ( 115,200) THEN   DIV! ( U.)  DROP  ;
```

---

```
( Set databits and parity)

Bits 1,0 of the Line Contol Register control the
 number of data bits (5, 6, 7, or 8).

Bits 5,4,3 of the Line Control Register determine
 the parity.  First we clear all 3 bits, then we
 set the proper ones for the requested parity.
 E.g.  8 DATABITS  NO-PARITY
       7 DATABITS  EVEN-PARITY

$87 to the FCR turns on the fifo and clears the
 tx & rx buffers and sets the trigger point to
 8 characters.  ($07 sets the trigger point to
 1 character.)
```

```
( Set databits and parity)
: FIFO-OFF ( -)  0 FCR PC!  ;
: FIFO-ON  ( -)  $87 FCR PC! NOP
  IIR PC@ $C0 AND $C0 - IF ( bad) FIFO-OFF THEN ;

: DATABITS ( #bits -)   5 8 CLAMP 5 -
   LCR 3 PORT-OFF    LCR SWAP PORT-ON  ;

: PARITY!  ( mask -)  LCR $38 PORT-OFF
                      LCR SWAP PORT-ON  ;
: NO-PARITY     ( -)    0 PARITY!  ;
: ODD-PARITY    ( -)  $08 PARITY!  ;
: EVEN-PARITY   ( -)  $18 PARITY!  ;
EXIT
: MARK-PARITY  ( -)  $28 PARITY!  ;
: SPACE-PARITY ( -)  $38 PARITY!  ;
```

---

```
( 8259 interrupt controller I/O port addresses)

IRQ_MASK  the hardware interrupt we want to
          enable for the chosen serial port

EOI  send "end of interrupt" message to PIC

ENABLE_PIC  the cleared bits in PIC_MASK register
            indicate which interrupts will be
            passed to the CPU

DISABLE_PIC  the set bits indicate which will not
             be passed to the CPU
```

```
( 8259 interrupt controller)

: IRQ_MASK ( - mask)
  1  SIRQ# @ FOR  2*  NEXT   ;
  ( above provides the OR mask; it must
    be inverted for ANDing)

: EOI ( -)  $20 PIC_CTRL PC!  ;
  ( send end-of-interrupt)

: ENABLE_PIC ( -)
  PIC_MASK IRQ_MASK PORT-OFF ( enable irq#) ;

: DISABLE_PIC ( -)
  PIC_MASK IRQ_MASK PORT-ON  ( disable irq#) ;
```

---

```
( Serial port control lines)

+DTR   turn on the DTR line
       (i.e. make it about +9 volts).

-DTR   turn off the DTR line
       (i.e. make it about -9 volts).

Similar words control the RTS and OUT2 lines
```

```
( Serial port control lines)

: DTR  ( - port bit) MCR ( ie port)  1  ;
: RTS  ( - port bit) MCR ( ie port)  2  ;
: OUT2 ( - port bit) MCR ( ie port)  8  ;
: +DTR  ( -) DTR ( port bit)  PORT-ON   ;
: -DTR  ( -) DTR ( port bit)  PORT-OFF  ;
: +RTS  ( -) RTS ( port bit)  PORT-ON   ;
: -RTS  ( -) RTS ( port bit)  PORT-OFF  ;
: +OUT2 ( -) OUT2 ( port bit)  PORT-ON  ;
: -OUT2 ( -) OUT2 ( port bit)  PORT-OFF ;

: LOOPBACK ( -)  MCR $10 PORT-ON  ;
: NOLOOPBACK ( -)  MCR $10 PORT-OFF  ;
```

---

```
( Macros for building an interrupt handler)

These macros are used to build a serial interrupt
handler _after_ the serial port is set up
(e.g. 2 COM ).  Then, at run time of the handler,
we won't need to access variables to find the
port addresses, etc.  Instead, they will be
hard-coded into the interrupt handler.  Thus,
the variables must be set up before the macro
executes.

        EOI,  send "end of interrupt" to the PIC
 ENABLE_PIC,  unmasks the correct serial irq line
DISABLE_PIC,  masks the correct serial irq line
```

```
( Macros for building an interrupt handler)

: EOI,  ( -)
  $20 #, AL MOV,  PIC_CTRL #, AL OUT,  ;

: ENABLE_PIC, ( -)
  PIC_MASK #, AL IN,
  IRQ_MASK $FF XOR #, AL AND,
  PIC_MASK #, AL OUT,  ;

: DISABLE_PIC, ( -)
  PIC_MASK #, AL IN,
  IRQ_MASK #, AL OR,
  PIC_MASK #, AL OUT, ;
```

---

```
DRAIN,
  while character(s) are waiting, read them and
  stuff them into the proper serial queue.
```

```
( Macros for building an interrupt handler)
: DRAIN, ( -)
  BEGIN,  LSR #, DX MOV, AL IN, 1 #, AL AND,
   0=, NOT, WHILE,
     DATA #, DX MOV,   AL IN,  ( read input char)
     SERIAL #, BX MOV, 0 [BX] CX MOV, ( mask)
     4 [BX] CX AND, ( tail)
     4 [BX] W-PTR INC,  ( incr tail)
     CX BX ADD,      ( add tail to start of buffer)
     AL 6 [BX] MOV, ( store char into buffer)
  REPEAT,   ;
```

**<< SHADOW** | **SOURCE >>**

```
( Macros for building an interrupt handler)

CLEAR-IID,
  while bit 0 of IIR is zero, take the most
  likely steps to clear the pending interrupt
  flags.  Note, this should not be necessary,
  since we do not enable any ints except for
  data received and reading the DATA port
  should clear all the interrupt flags
  associated with received data.  However,
  others have warned that this step is needed
  because some of the other interrupt flags
  still get set from time to time.
```

```
( Macros for building an interrupt handler)

: CLEAR-IID, ( -)
  BEGIN,  IIR #, DX MOV, AL IN, 1 #, AL AND,
   0=, WHILE,
     MSR #, DX MOV,   AL IN,
      ( in case delta flags caused the int)
     LSR #, DX MOV,   AL IN,
      ( in case OE etc caused the int)
     DRAIN,
      ( in case data ready caused the int)
  REPEAT,   ;
```

```
( lay down all the code for the interrupt handler)

HANDLER,
  This lays down the assembly code for the
  interrupt handler.  It must not be done until
  after the 'SDATA SIRQ# etc variables are set,
  e.g. with 2 COM, as the macros need to know
  port and interrupt values.

  It goes through a rather elaborate procedure to
  attempt to guarantee the irq line will never be
  stuck on.  See the Serial Port FAQ for the
  reasoning behind this.  Probably the handler
  does not need to be this elaborate if only a
  single port is attached to the irq.
```

```
( lay down all the code for the interrupt handler)

: HANDLER, ( -)
  DS PUSH, DX PUSH, CX PUSH,
  BX PUSH, AX PUSH,  ( ie save the registers)
  CS PUSH, DS POP,
   ( set DS in case int is called from DOS, etc.)
  DISABLE_PIC,  EOI,  STI,
  DRAIN,  CLEAR-IID,  CLI,  ENABLE_PIC,
  AX POP, BX POP,
  CX POP, DX POP, DS POP, ( ie restore registers)
  IRET, ;
```

```
( install serial interrupt routine )

OLD-SVECTOR
  place to save old serial interrupt vector

INSTALL-SINT
  save the old vector, install vector pointing to
  our own handler, use IRQ_MASK to clear the bit
  in the 8259 chip to allow our hardware
  interrupt, set certain of the serial port
  output lines, tell the serial chip to interrupt
  only on datain, clear any accidentally pending
  interrupt, and make sure the SERIAL buffer is
  empty.
```

```
( install serial interrupt routine )

VARIABLE OLD-SVECTOR 2 ALLOT
0 0 OLD-SVECTOR 2!

: INSTALL-SINT ( a -)
  INTS-OFF  +DTR 10 MS -DTR 10 MS +DTR
  200 MS   SINT# @ INT-VECTOR@   OLD-SVECTOR 2!
  CS@  SWAP  SINT# @
  ( seg offset int#) INT-VECTOR!   ENABLE_PIC
  $0B MCR PC!  ( set DTR, RTS, & OUT2 high)
  1 IER PC!  ( enable only data-in int)
  LSR PC@ MSR PC@ 2DROP  DATA PC@ IIR PC@ 2DROP
  EOI  RESET-SER-IN  INTS-ON  ;
```

```
( uninstall serial interrupt routine )

UNINSTALL-SINT
  disable the serial interrupt at the 8259 and
  restore the saved vector (not that it would
  do any good with the int disabled at the 8259)
```

```
( uninstall serial interrupt routine )

: UNINSTALL-SINT ( -) OLD-SVECTOR 2@ OR
  IF  INTS-OFF  DISABLE_PIC
      OLD-SVECTOR 2@  SINT# @  INT-VECTOR!
      0 0 OLD-SVECTOR 2!  INTS-ON
  ELSE CR ." Nothing to uninstall! " CR  THEN  ;
  ( might also need to drop DTR or
    RTS in some cases)
```

```
( Create and install interrupt handler)

BUILD-HANDLER
  This is used to define the serial interrupt
  handler code at run-time.  It must not be done
  until the com port has been established.  It
  builds a custom handler for the specific com
  port that is active when BUILD-HANDLER is
  executed.

START
  Establish the correct com port and define and
  install an interrupt handler for it.
```

```
( Create and install interrupt handler)

: BUILD-HANDLER ( -)
  " CODE SERIAL-HANDLER HANDLER,  END-CODE
    ' SERIAL-HANDLER  INSTALL-SINT  "  ( a)
   COUNT  ( a #) EVALUATE  ;


: START ( -)
  2 COM  FIFO-ON 9600 BPS   BUILD-HANDLER    ;
```

```
( DUMB terminal)

DUMB
  example of using the serial routines to make a
  dumb terminal.  It prints to the screen any
  characters that come in on the serial port.
  It sends to the serial port any keys pressed
  on the keyboard.   Except, pressing the Esc
  key stops DUMB.
```

```
( DUMB terminal)

: DUMB ( -)  ( abort by pressing ESC key )  CR
  BEGIN
    SER-IN?
       IF  SER-IN EMIT  THEN
    KEY?
       IF KEY DUP 27 =
           IF DROP CR ." now in Pygmy " CR EXIT
           ELSE SER-OUT
           THEN
       THEN
  AGAIN  ;
```

# Program This! The AT Modem Commands

By David Goodenough

Special Feature

All Reader

Modem control

AT +MS=11,1,300,28800 S48=7 S36=4

Huh?

The **AT command set** for modems can be a rather daunting challenge, when you're trying to set up your new modem with a communication package. Many good programs will come with lists of init strings for all the various known modems, i.e. these are AT commands that set up your particular modem for the application in question.

**What exactly goes into one of these AT commands?**

The first thing to do is cover why they're even called AT commands in the first place. This is because they invariably start with the two letters 'A' and 'T'. The reason for these two is shrouded in antiquity, but is as valid today as it ever was. It is possible to communicate with your modem at many different speeds (BPS rates). There is a certain amount of smarts in the modem, and when it is waiting for a command, it doesn't actually know if you're going to send one at 2400 BPS, 19200 BPS or 57600 BPS, or any other speed for that matter. So, it watches the serial line, and when it sees the 'A', it analyses the bit pattern, and is thus able to determine the BPS rate. In the good old days of parity, it would then also scan the 'T', and by looking at what it had received for both letters, it could determine the parity in use, if any.

Thus, by the time AT has been typed, the modem is awake, and knows what speed the rest of the command will be coming at. It will then remain in this mode, until it sees a carriage return (Hex 0d), at which point it will try to execute the command.

The AT is also a mnemonic, in that it gets the modem's ATtention, and this is the simple explanation that is given in most documentation, since it covers things very nicely and simply.

Following the AT can come all sorts of commands. There are quite a few that are common to just about all modems, I'll cover these first. Also note that I've placed spaces in these commands for readability. The commands are always designed in such a way that they can be crunched together with no spaces. A single command that requires multiple letters (e.g. AT &W0) should not have the &W0 split up, however you can place spaces before the & and after the 0 with no ill effects.

**AT Z:** one of the most commonly used commands, this resets the modem to a stored profile. There are often two extensions to this: AT Z0 and AT Z1, which can select from one of two stored profiles, with the addition that it is sometimes possible to arrange it so that an AT Z will use either of the two profiles as it's default. **AT &F** is frequently used to 'restore' the factory defaults which is the initial profile set up by the manufacturer.

After the AT Z has completed the modem responds with either OK, or the single digit '0'. This response means that the modem executed the command correctly, about the only other response that will be seen for most commands is ERROR (or '4'), meaning the modem couldn't make sense of what you asked. AT Z4 will typically elicit an ERROR response, since the only valid numbers that can be used with the AT Z command are 0 and 1.

The usual state for modems is to echo commands, i.e. the modem sends back what you typed at it, which means to get to see what you typed. This can be disabled with **AT E0** (echo off), and enabled with AT E1 (echo on). Whether you get and OK or a 0 is determined by the **AT V** setting: AT V1 (verbose on) gives full words, while AT V0 (verbose off) gives just the numbers. Finally, the responses can be completely disabled with the **AT Q** command: AT Q0 (quiet off, i.e. send responses) is the default, AT Q1 (quiet on) silences the modem. This means that AT Q1 E0 will completely silence a modem: you'll see no echo of your commands, and the modem won't respond.

**AT &W, AT &W0, AT &W1:** these place the current modem setup into one of the stored profiles that can be used with AT Z; while **AT &Y0** and **AT &Y1** select which of the profiles will be used for the default AT Z. Note that the &W and &Y commands are common to most modern modems, but are not always to be found with older modems.

Finally most modems use **AT &V** to show the current settings, as well as the stored profiles, although I have come across one modem that used an AT In command to do the same thing.

While on the subject, **AT I0** thru AT I<whatever> are often provided to allow the modem to be identified. The output from these commands, and the number of them that exist tends to be a bit variable, for example on the Cardinal MVP 288 XF that I own, AT I3, I4, and I6 are perhaps the most useful, showing the firmware revision, identifier string, and data pump revision respectively. The firmware revision is important, since it is possibly to upgrade this modem via its flash ROM.

So far, we can reset the modem, and figure out what sort of modem it is. However, what about doing something useful, like actually dialing.

**AT D ...**

is the dial command, this is universal across all modems that support the AT command set. In it's simplest case, just follow the AT D with the number:

**AT D5551212**

and the modem will go off hook, dial the number, and try to establish a connection. It will come back with one of several responses, depending on what happened. The four most common are:

**BUSY** - it detected a busy signal;

**NO DIALTONE** - it didn't hear dialtone when it went off hook;

**NO CARRIER** - it dialed correctly, but didn't make a connection to another modem;

**CONNECT nnn** - it got through, and established a connection at nnn BPS.

Further up, I commented that single commands should not have any spaces in them, the D command is the one exception to this. It allows additional punctuation to be added, which is silently ignored. This is most likely done for the sake of readability:

**AT D 1 (510) 555-1212**

is a little more readable than:

**AT D15105551212**

There are a couple of common sub-commands to the D command. Putting in a **T** or a **P** will make the modem use tone or pulse dialing respectively, and a comma ',' can be used for a delay, this delay is usually two seconds per comma. This might be used when you're using a phone system where you have to dial 9 to get an outside line:

**AT DT 9, 555-1212**

will dial 9 to get the outside line, wait 2 seconds, and then dial the remainder of the number. Likewise:

**AT DT 0 (510) 555-1212,,(408) 555-1212 1234**

Dials a number with a leading 0, which means this is a calling card call. It then waits two comma's worth (four seconds) for the calling card tone to sound, and then dials the card number.

Lastly, using just an L as the dial command will cause the modem to dial the last number again:

**AT D L**

It is worth noting that my Cardinal has quite an extensive collection in addition to the four enumerated above, for instance **W** will wait for dial tone again, while **&** waits for the 'AT&T "Bong" tone' [sic] i.e. the calling card tone. This means that dialing a calling card call from an inside line could be done as:

**AT D 9 W 0 (510) 555-1212 & (408) 555-1212 1234**

A final pair of command that go hand in hand with the D command are **AT M** and **AT L**. AT M0 and AT M1 disable and enable the modem's speaker respectively, providing the option to listen to the call in progress or not. Use AT M0 if you want to remain friends with everyone in the dorm. When AT M1 has the speaker on, AT L sets the volume: AT L0 is quietest, thru AT L3 which is loudest.

Now that we've got online, what commands have an effect now? There are a couple more that have an impact on the call:

**AT &Dn**

and

**AT &Cn**

which modify the modem's treatment of the DTR line, and how it outputs to the DCD line.

It is worth noting at this point that DTR gets its name from the days when modems were directly connected to dumb terminals. DTR means Data Terminal Ready, and is used to allow the terminal to tell the modem that it's on line. The two most useful variations of AT &D are AT &D0, which causes the modem to completely ignore DTR, and AT &D2 which causes the modem to hangup and re-enter command mode if DTR is made inactive. AT &D2 is useful, since 99.9% of communications programs deactivate DTR when they exit, which means that you can't accidentally leave yourself online after exiting your comm program, running up a huge online bill.

Likewise, DCD stands for Data Carrier Detect, and is a signal from the modem to the terminal that it is connected to another modem. AT &C1 is the most common setting, this causes the modem to make DCD actually reflect the state of carrier, so that your comm program can monitor the line, and determine if you're connected or not. AT &C0 causes the modem to keep DCD permanently active, this might be used if the comm program can't work at all unless it sees DCD.

Returning to the AT &D command for a moment, the question needs asking "how do I hang up if the modem doesn't respond to DTR?" The mechanism to allow this is quite clever. While online, if the modem detects a delay of 1 second, followed by three '+' characters, followed by another delay of 1 second, it will return to command mode, but it also stays online. At this point, the **AT H0** command will cause it to hang up. AT H0 has a counterpart: AT H1, which causes the modem to go off hook, and then return to command mode. About the only use for this might be to busy out the phone line if the modem wants to perform a long set of initialisation commands, and it doesn't want a call arriving in the middle of them.

The net result of all this is that if your comm program delays for one second, sends three '+' characters, delays for another second, and then sends AT H0, it will make the modem terminate the call, and go on hook.

The last major area that needs covering are the modem "S" registers. These are numeric values that can be set to alter the behaviour of the modem.

The general syntax is:

**AT Sr=v**

where **r** is the register number, and **v** is the value to place in it. It is also possible to inspect them by using the:

**AT Sr?** command.

There are a number of these that are common across just about all modems, I'll cover these. However take note that there are also many of these that vary based on the modem manufacturer, you'll need to consult your modem's documentation to get details of them.

**S0** is the number of rings to wait before answering. If this is set to 0, auto answer is disabled.

**S1** is a read only register (i.e. you can't set it with AT S1=v), and reports the number of rings seen so far. When S1 becomes equal to S0, the modem answers the phone.

**S2** is the "escape" character, entered as a decimal number. I noted above that a delay, three '+'s and a delay return to command mode - S2 lets you change the '+' character. The default is 43, which encodes a '+' character, but it could easily be changed to 45 for a '-' character, or 63 for an '=', or whatever else.

**S3** is the command termination character in decimal. The default is 13, meaning use a carriage return.

**S4** is the line separator character, this is output along with the command termination character to separate lines of output from the modem. Default is 10, for a line feed.

**S5** is the delete character, i.e. the one you can use to erase mistakes in the command. The default is 8, for backspace.

**S7** is the delay for carrier detect in seconds, this tells the modem how long it should wait after dialing to try to establish carrier with the other modem. With the advent of the complex negotiation systems of high speed modems, it's a good idea to give this a fairly high value. I use 60 seconds, to give it a whole minute to place the call and connect.

**S11** is the tone dial speed in milliseconds, i.e. the duration of the tones used to dial, as well as the gap between tones. A safe value is 75, although this may work as low as 55, or even 50.

This covers the most common commands. I have not dealt with such things as data compression, error correction, flow control and the likes, since these tend to vary between modems. Check your documentation for details, although it is worth noting that the default setup is often close to optimal, and needs very little alteration.

Feature

Kaypro Owners

Internal Modem

# Mr. Kaypro

by Charles B. Stafford

**WHEREIN**

We continue the transmogrification of a basic luggable computing box into an indispensable personal assistant. Since all good personal assistants must be able to communicate, we must, therefore, implant a communications device (aka a modem). The goal is to use a standard RS-232 new/surplus/used 1200/2400/9600 baud external modem and mount it semi-permanently internally in whatever flavor of Darth Vader's lunchbox you have.

**THE CHALLENGE**

then, for your sometimes errant Scribe, is to devise a process that will work with whatever you happen to pick up at the local swap/flea market or thrift store, or, heaven help us, buy from your local retailer or mail-order house.

The challenge for you, is to pick and choose; first, the hardware you're going to work with; and second, the particular techniques and procedures from the plethora about to be described, that will work for you.

**THE BATTLE PLAN**

The concept of what we're going to do is simple, take a readily available external modem, strip off the box, physically mount it somewhere, hook it up so that it doesn't completely disable our serial port, AND do it easily and elegantly. As some much more erudite WIZARD once said, "the concept is a piece of cake, but, the devil is in the details." TAKE HEART, however, when we did the original project, we used three different modems, an Everex 2400, a Hayes 2400, and a no-name turbo 9600, and it all came out the same in most details. The real key, is to use a modem that will work when connected to the serial port in the conventional manner, with an external cable. If it will work connected externally, then it will work when we put it inside.

**HOW WE GET AWAY WITH IT**

We have to deal with two voltages; 120V ac,the power supply; and +12V to -12V, the RS-232 signals. I suppose that a real purist would first do the PC-XT power supply transplant (prior article) and use it to supply regulated +5V, +12V, and -12V directly to the proper places on the modem circuit board, BUT, this project is designed so that even I, Joe Fumblethumbs, can do it with a minimum of fuss and bother, and a reasonable chance of success. So, here's what we're going to do: most modems use a "wall-wart" (plug-in transformer for you new readers), or an inline transformer to get from 120Vac down to somewhere around 8-9Vac and then use a voltage doubler, as well as a bridge rectifier circuit and regulators to get the correct voltages. This means we can use the same trick we used to stabilize the display with a small modification, a power switch, to put the wall-wart inside.

With the power taken care of, (don't worry, I know I didn't give you any details, that comes later) all we have to face is the RS-232 connection. What we are dealing with here are the most common RS-232 drivers; the 1488, the transmitter and the 1489, the receiver. Both the modem and the Kaypro use them. The difficulty arises when we tie the output of a 1488 (the modem's pin 3) to the input of a 1489 (the Kaypro's pin 3) AND the output of ANOTHER 1488 (the Serial port pin 3) is also present. The reason that this presents a challenge, is that the output of a 1488 can either look like ground or an open, powered or not, and is completely unpredictable. The solution to this dilemma is as elegant as it is sneaky. We just put a 6.8k resistor in series with the 1488's output line (see Figure 1). If the other 1488's junction is trying to look like ground, this will make it look like any other 6.8k impedance, and as a matter of fact the rated input impedance of a 1489 is; trumpets please; 3-7k ohms!!! The cur- rents involved are vary low so the voltage drop as far as the signal is concerned is negligible, but if we were using long transmission lines, this probably wouldn't work. Again details later.

**Construction**

First the physical considerations; stripping and mounting the modem circuit board, locating and mounting a power switch for the modem, and last but not least, installing the power.

Most external modems up to now have been made to be repairable, that is to say, you can open the "box" and close it again the way the manufacturer did with no damage. Normally when this is done and the box disposed of, you are left with a populated printed circuit board that has LEDs on one end and connectors on the other. The most convenient spot to mount it is on the side of the drive cage next to the monitor, unless you've done the hard drive conversion, which uses this spot and then you're stuck with the second easiest spot, the shield below the motherboard. In either case, double stick foam pads, (the thick kind) will do the job, or in the case of the side of the drive cage, standoffs can be used, metal or plastic, provided you use flat head screws inside the drive cage, so as not to interfere with the drives, or "pop-rivet" 6-32 threaded inserts can be epoxied to the side of the drive cage and used as standoffs. In order of ease of installation the foam pads are first with the "pop-rivets a close second, beause you can mount the inserts on the back of the modem board, flange out, daub a little epoxy on the flanges and stick it on the side of the drive cage, holding it with masking tape until the epoxy is set (24 hours?). Prior to mounting, however, (here comes the devil) you might want to consider whether or not you want the the LEDs exposed. You could drill individual holes in the front panel, to "let the lights shine through" but most modem software uses FYI (for your information) messages to tell you the same information, and laying out all those holes could lead to a gigantic headache. They were left concealed on the prototype.

## ACTION TIME

It's time, now to get out the screw-driver and remove first the power plug from the outlet and then the ten screws that hold the hood on, and then the hood. There are two regular screws on the back and two fasteners for each serial port and two for the parallel port. Remove all of these, the connectors on the mother- board and the two screws at the front of the mother-board, and then the mother-board it self. Now, you're ready to mount the modem as described above.

## 120VAC

Two out of the three modems that were experimented with for this project used wall-transformer power supplies, but the third used an inline transformer. They can be handled in similar fashions, using the same trick that was first used to provide an independent 12V supply to the video board with one exception, for this application a power switch is needed. On the prototype a 120Vac rated miniature toggle switch was mounted on the back panel by drilling a 1/4 inch hole above the main power switch. If you've already moved the reset button, you could use its former location and avoid drilling a new hole. In either case, it's time to find an old but good extension cord that the wall transformer will plug into and cut off the socket end with about a foot of wire. Using automotive spade type crimp connectors (just like Kaypro did) connect one side of that foot of wire to the "NEUTRAL" side of the main power switch (white wire), and the other to the "HOT" side of the main power switch (black wire), after going through the new modem power switch. Then plug in the wall transformer and mount it on the bottom of the case using some of those double-stick foam pads. An inline transformer can be mounted the same way, but its ac line will have to be connected direct to the "NEUTRAL" and "HOT" leads through the switch. You can now switch power on and off to the modem, so on to:

## The Signal Lines

On the modem circuit board somewhere there is a serial connector, either a DB-25 (the most common) or a DB-9 in addition to the power connector and the modular telephone connector(s). Since it would be nice to be able to replace the modem easily, should the DRAGONs prevail and munch the poor beast, use of the connector is dictated. Here you must make a choice, you could use a solder-cup type connector and solder wires with resistors inline to the appropriate pins, or you could do it the way the prototype was done using a "patch-box". A "patch-box" consists of male and female connectors on opposite ends of a small circuit board enclosed in a plastic case. The circuit board has interrupted traces between the connectors with solder pads at the breaks, so that cross connections could be made to produce, for instance, a null modem device or in this case putting resistors in-line with certain pins (see Figure 2). In fact, two patch- boxes were used on the prototype, one connected to the modem and one more, identical to the first, connected to the serial port on the back of the machine.

The patch-boxes were configured so that pins 3, 5, and 8 have 6.8K ohm 1/8th watt resistors in-line and pins 1, 2, 4, 7, and 20 have the jumpers supplied with the patch box installed "straight through".

The next challenge is "how do we get from here to the serial port (the data port on the '84 machines, the only serial port on the '83 machines). The most elegant way would be to etch a small circuit and use two wire wrap sockets so that the 1488 and 1489, which are next to each other could be removed the circuit board plugged in and a cable routed down to the modem. Alternatively, a piece of prototype board could be used with the same results. Both of these solutions require, a fair amount of fuss, mess, and bother, so here's a "quick & dirty" way to do it. RADIO SHACK sells "Micro Test Clip Leads" which come in a package of two, and consist of a length of wire with a teensy little test clip on each end, that once clipped on, hangs on like a teenager to a telephone. It only takes seven to do the job, remove one clip from each, strip the end 1/16th of an inch and solder it into the appropriate solder cup of a solder-cup type connector. Remember, pins 1, 2, 3, 4, 5, 8, and 20 are the ones to use, pins 1, and 7 are both grounds so you only need to use pin 1. With an indelible pen mark the pin numbers on the sides of the test clips.

Before re-assembly is started, there is one more detail, (remember the devil?) to be attended to, the telephone connector. There are several ways to handle this, including just hanging a piece of telephone wire out through one of the cooling slots at the back of the computer. The most sanitary way is to procure a panel mount modular connector and put another hole in the back panel, and use a short piece of telephone line cord to connect it to the modem. A compromise solution is to beg/borrow/steal/buy a wall surface mount telephone junction box (they're about 2 inches square and about 7/8th of an inch high) and mount it on the back panel on the outside so that the "line" side is over one of the cooling slots, and run a short piece of line cord with a modular connector on one end to the modem. If you wish to have a connection for a telephone as well, you could duplicate your preferred solution for connection to the telephone socket on the modem, oe use a two into one splitter connector (Radio Shack or your favorite hardware store).

NOW, it's time for re-assembly. Plug one of the patch boxes onto the modem, and plug the modular telephone connector(s) and the power connector into the appropriate sockets. Plug the solder-cup type connector with all the numbered leads and micro test clips hanging off of it onto the patch box and route the leads either between the mother-board and the back panel or around the side of the mother-board, and re-install the mother-board, with a screw into each standoff, two on the back panel, and the two fasteners for each port.

At this point, it is appropriate to recheck all connections below the mother-board, making sure that you haven't created any inadvertant shorts.

Here's where the identity of your machine becomes critical. There are essentially three varieties of mother-boards, the '83 K-II/IV, the '83 K-10, and the '84 series. Identification is relatively easy, the '83 K-II/IV has only one DB-25 Serial port, the '83 K-10 hard-drive cable (50 conductor flat) connects in the center of the mother-board, and everything else is an '84 series machine. (Robies follow the ID pattern, but you'll have to devise your own physical mounting scheme). Once you've identified your computer, pick the corresponding connection data below and connect all the micro test clips to the proper places. Care is essential, because some of the signals you're working with are 12V and if you get them connected to a 5V device, an IC could be fatally injured by the DRAGON's bite.

| Micro Test | K-II/IV | | '83 K-10 | | "84 Series | |
|---|---|---|---|---|---|---|
| Clip Pin | U | Pin | U | Pin | U | Pin |
| 1 | ground at the power connector pin 4 | | | | | |
| 2 | 68 | 3 | 17 | 3 | 4 | 11 |
| 3 | 69 | 1 | 4 | 1 | 5 | 10 |
| 4 | 68 | 11 | 17 | 6 | 4 | 8 |
| 5 | 69 | 13 | 4 | 4 | 5 | 4 |
| 8 | 69 | 10 | 4 | 13 | 5 | 13 |
| 20 | 68 | 8 | 17 | 8 | 4 | 6 |

Recheck the connections and re-install the hood with its ten screws, and it's time for checkout.

## Software

Both MDM740 and MEX 114 were successfully tested with the prototype installation. During the checkout they werte both used in the "Terminal" mode by typing in "at" and return. The modem should echo the characters "at" back to the screen and reply "OK" after the return. This assumes a Hayes compatable modem. For other cases, consult your modem manual. If all goes well up to this point, plug in the telephone line and call a local bulletin board and try a more extensive test. TCJ's number, by the way, is (916) 722-5799.

## Roadhouse

A small tip, not related to computers. IF you happen to be traveling in Northern Califirnia, in Marin county, (that's the one where all the Alternative Intelligences are, you know, the ones featured in the movie "Serial"). On California Highway One, The Pacific Coast Highway, at the intersection of the Francis Drake Highway, which goes out to Point Reyes, there is a small roadside reataurant called The Roadhouse, which serves an exquisite oyster chowder with huge plump locally grown oysters. They have other food as well as the best of the California wines, but if you are an oyster fan, this place is not to be missed! The prices are reasonable, too.

## MATERIALS LIST

| ITEM | SUPPLIER |
|---|---|
| Your K-II/IV/10/2/4 | your closet(?) |
| a modem of your choice | swap/flea market |
| 2 Patch Boxes | HSC/Jameco/Local |
| 6 6.8k ohm resistors | " |
| 1 solder-cup type male DB-25 connector | " |
| 7 Micro Test-clip leads | Radio Shack |
| 4 standoffs/pop-rivet nuts | HSC/Jameco/Local |

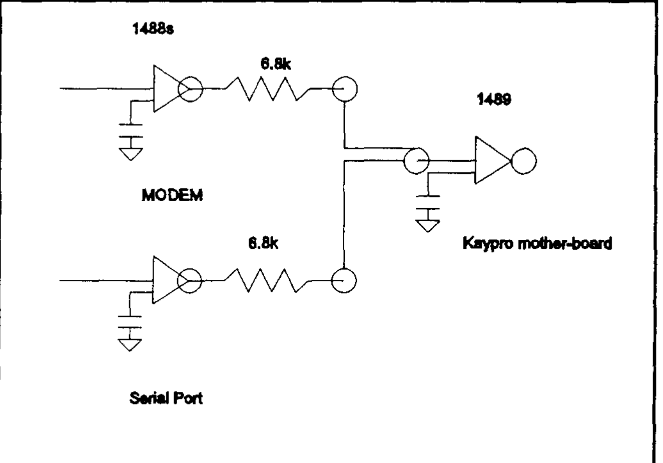


Figure 1

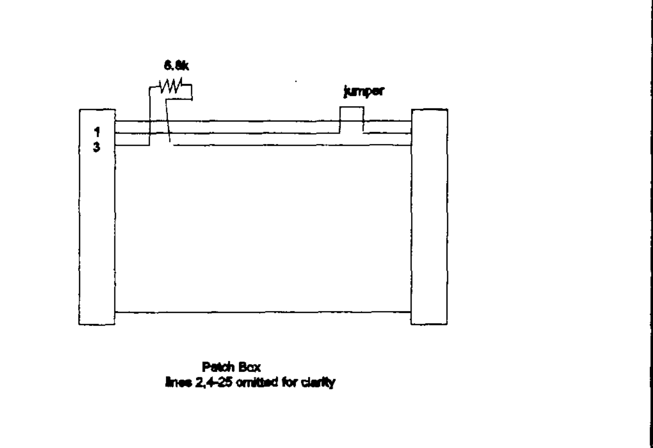


Figure 2

# High Speed Modems and CP/M

by Terry Hazen

Special

CP/M Users

High Speed Modems

**Using 14.4k Modems with CP/M Modem Programs**

If your CP/M system has a modem serial port capable of at least 19.2k, you should be able to soup up your modem performance by upgrading your IMP or ZMP modem program overlays and adding a 14.4k modem. If you don't have a 14.4k modem yet, remember that you get what you pay for. Cheap ones don't always work as advertised. The modem control codes in the examples are for the US Robotics Sportster or Courier 14.4k (or 28.8k) modems. Consult your own modem manual for any differences.

IMP245 and ZMP15 are two standard CP/M modem programs written at a time when 2400 baud was considered high speed. Both programs and a wide selection of their overlays are widely available on most CP/M BBS systems as well as on the Walnut Creek CP/M CDROM. If you already have an IMP245 or ZMP15 overlay for your system, it's easy to modify it for use with a 14.4k modem. You should also be able to adapt BYE and your MEX or QTERM overlays using the same techniques. See the end of the article for information on downloading the Yasbec SBC (Z180) versions of the 14.4k IMP and ZMP overlay files.

I'll limit my discussion here to modem port speeds of 19.2k. With higher port speeds, you encounter other issues such as hardware handshaking and character buffering. The Yasbec integral ASIC serial ports, for example, don't have buffering and may or may not be able to talk to a modem at 38.4k without dropping characters. If your modem port can run at 38.4k with handshaking and buffering, though, and you want to try running a 28.8k modem at 38.4k, don't be afraid to give it a shot.

For 14.4k connections, set your modem serial port to 19.2k, 8 bits, no parity and one stop bit (8N1) before you run IMP or ZMP, as neither overlay specifically initializes the modem port. Don't try to run a 14.4k modem at a serial port speed of 14.4k! If you find you can use a higher modem port speed without dropping characters, it will improve efficiency somewhat. USR's ATI4 command will display the current USR modem's settings and will also quickly show you if you're dropping characters or not.

If you aren't familiar with 14.4k modems, one change from 2400 baud operation is that the 14.4k modem must be initialized differently since it operates differently. That means that we can no longer use the 1200 and 2400 baud modem initialization strings built into IMP. High speed modems also have some new features that require proper initialization.

1200 and 2400 baud modems require that the modem port speed be the same as the modem connection speed. When you connect at a different speed, your modem overlay has to change the modem port speed to match. No longer. The modem port speed stays fixed and the modem determines the connection speed and provides buffering so that the data always flows between modem port and modem at the fixed rate. For the USR modem, the command '&B1' tells the modem to maintain a fixed modem port speed.

Another new modem feature is ARQ error control, which is enabled in the USR modem with the command '&M4' and disabled with the command '&M0.' Error control is only useful when connecting to another high speed modem. ARQ error control should be the default for the best 14.4k connections.

Few 2400 baud modems, however, support ARQ error control. 14.4k modems will negotiate with the remote modem on connection and if that modem doesn't support ARQ, yours will fall back to disable it. But that takes time and can interfere with the display of the remote system's logon screen, so it works best to always turn ARQ off before calling a 2400 baud system.

Data compression is another new feature which works only with another compatible high speed modem and is used to provide compression during data transfers. Most BBS files are already compressed and little can be gained from data compression when transferring already compressed files. Data compression is most useful when you're transferring plain text files. '&K1' sets auto enable/disable.

**The High speed IMP245 Overlay**

I2YN14-2.Z80 is an IMP245 overlay file for use with the Yasbec and USR modems for connections at up to 14.4k. You can adapt it for use with your system by replacing the computer-specific modem code with the equivalent code from the current overlay for your system.

The 14.4k IMP overlay uses two high speed modem initialization strings at STRNG1 (ARQ error control and STRNGA (no ARQ error control) that are specific for the USR

Sportster or Courier 14.4k and 28.8k modems. If you're using a different modem, remember to check your manual and modify these commands as required. If you choose, you can save the modem initialization strings to the modem non-volatile RAM NVRAM0 and NVRAM1 using the &Wn commands and change the strings in the overlay to initialize the modem using the shorter ATZn commands.

Modem initialization can be done in many ways. Don't be afraid to read your manual and experiment. The USR initialization string I use for IMP and ZMP is:

**AT** - Attention command
**&F0** - Load generic factory template

Then I modify the &F0 template:

**&M4** - Enable ARQ error control (&F0 default)
(&M0 disables ARQ)
**&K1** - Auto enable/disable data compression (&F0 default)
**Q0** - Display result codes (&F0 default)
**V0** - Display numeric result codes for IMP
(V1 - Display verbal result codes for ZMP)
**X4** - Display full result codes
**&A0** - Disable extended result codes for IMP
(&A3 - Display all protocol indicators for ZMP)
**&B1** - Fixed modem serial port speed (&F0 default)

The HS2400 equate in the overlay now controls the modem ARQ error control default. Set HS2400 to YES if you mostly use 14.4k connections and you want ARQ error control (&M4) to be the default and set it to NO if if you mostly use 2400 baud connections and you want ARQ disabled (&M0) as the default.

The IMP overlay uses IMP's SET command to toggle and display the ARQ error control setting, replacing it's original function of setting the baud rate. Use the SET command 'SET N' from the IMP command line to disable ARQ error control (&M0) before calling a 2400 baud modem. ARQ can be also reset from the IMP command line 'SET Y' before calling a 14.4k modem. The current ARQ state will be displayed during operation and on connection.

When a modem makes a connection, it sends the computer a connection result code, either a number or a phrase depending on how you've configured the modem. The modem program can then identify the connection speed and switch from calling mode to terminal mode, ready to continue. Since IMP and ZMP were writtten before high speed modems, neither one can recognize the two-digit high speed result codes. The 14.4k IMP overlay gets around this problem by including a patch for IMP.COM that replaces IMP's 300 baud test with a test for any two-digit result code, signifying connections at rates greater than 1200 baud.

IMP's file transfer time displays are based on MSPEED, the serial modem port speed byte. The overlay starts with MSPEED=9 (19.2k,) the closest MSPEED to 14.4k. Since the 14.4k connection speed is slower than the 19.2k MSPEED value, the transfer times IMP displays for 14.4k connections will not reflect the actual connection speed and should only be used as a general guide.

If your modem connects at 2400 baud, the overlay will change MSPEED to 6 (2400 baud) so that the displayed file transfer times will match the 2400 baud connection speed. A subsequent 14.4k connection will change MSPEED back to 9.

**The High speed ZMP15 Overlay**

ZMO-YN14.Z80 is a ZMP15 overlay file for use with the Yasbec and USR modems for connections at up to 14.4k. Like the IMP overlay, you can adapt it for use with your system by replacing the computer-specific code with the code from the current overlay for your system. ZMP supports the fast and efficient ZMODEM file transfer protocol, which is compatible with the PC BBS version. It's also very useful with CP/M remote systems that support ZMODEM.

In order to take advantage of the ZMODEM protocol at 14.4k, however, the original ZMP overlay routine MRD has to be modified, as the 100ms software timer in the routine won't work properly at high speeds. You can use either of two approaches.

The first requires your system to have a 100ms interrupt-driven or hardware timer or down counter available. The Yasbec ZMO-YN14.Z80 overlay uses this hardware approach, as the Yasbec n/BIOS and B/P Bios both have 100ms interrupt-driven down counters as part of the bios code. The n/BIOS TIME call provides the address of a 100ms interrupt-driven down counter in the bios. The down counter is set and checked at the MRD label in the overlay. Since the B/P Bios down counter isn't directly user-accessible, B/P Bios must be slightly modified to provide user access to the MTM down counter located in the FDC-xx.Z80 module. One way to do that is to modify the TIME routine in the TIM-xx.Z80 module so that a call to TIME returns the address of MTM in register BC, since that register is not presently used or preserved. Setting the BPBIO equate in the ZMP overlay to YES provides for B/P Bios MTM access when B/P Bios has been modified as described.

If your system doesn't have a hardware timer available, you can take the software approach used in the Yasbec ZMOYN14A.Z80 overlay, which modifies the MRD routine the way Simeon Cran did in his MYZ80 ZMP overlay. Instead of calling MIRDY (renamed to AUXIST in the Yasbec ZMP overlay file) to check the modem input character status, then waiting 100 ms before checking again, it calls MIRDY many more times and does no waiting at all. Simeon says that the number of times MIRDY is called probably needs to be adjusted according to the speed of the system, but it doesn't matter too much if it calls too many times. He used 4000 in his MYZ80 overlay and that value also works fine on my 18 Mhz Yasbec system.

If you use the software timing approach, you'll probably find that when you first try to dial a number, ZMP will abort the call before completing the connection, since all the character waiting has been eliminated. I fixed that problem on my 18 Mhz Yasbec system from the 'C' configuration menu by selecting 'M' for modem configuration and changing the Redial timeout delay from the default of 40 to 750. That allowed four rings before aborting the call (the default 40 with a hardware timer allowed 7 rings.) You'll probably have to adjust the value for the clock speed on your own system.

Unlike IMP, ZMP has no provision for a modem initialization string in the overlay. Instead, you enter a modem initialization string for your modem from the 'C' configuration menu by selecting 'S' for Set modem parameters. As far as I can tell, the initialization string is not sent to the modem unless you directly specify it from the 'L' menu. See the IMP overlay section for modem initialization information. Before calling a 2400 baud remote system, be sure to send the USR modem the string 'AT&M0' from the main screen in order to turn off ARQ error control.

Like IMP, ZMP doesn't recognize the two-digit high speed connection result codes. Since I couldn't patch the ZMP code to take care of that, ZMP requires some user action after placing a call. When ZMP makes a connection and displays the result code, you'll need to press ESC (ignore the resulting 'Call Aborted' message - it hasn't been.) You'll now be back in terminal mode, ready to continue when the remote computer signs on. Not elegant, but it works.

Like IMP, ZMP's file transfer time displays are based on MSPEED, the modem serial port speed byte. Since this is not the actual connection speed, the transfer time displays are based on MSPEED, the modem serial port speed byte. Since this is not the actual connection speed, the transfer times ZMP displays for 14.4k connections will not reflect the actual connection speed and should only be used as a general guide.

The ZMP overlay assumes that you're running ZCPR3 and contains the routines required to obtain the terminal cursor addressing and highlighting control codes from the Z3TCAP terminal capabilities module in the ZCPR3 environment. If you aren't using ZCPR3, order NZCOM right away! Alternatively, change these routines to suit your specific terminal.

## Obtaining the Overlay Files

You can download the full modem overlay files in crunched form as I2YN14-2.ZZ0 (for IMP245) and ZMO-YN14.ZZ0 and ZMOYN14A.ZZ0 (for ZMP15) from any of the following BBS systems:

| | | |
|---|---|---|
| Adam's RiBBS (WA) | (206)481-1371 | (2400-PBBS) |
| DHN* (PA) | (215)535-0344 | (2400-HBBS) |
| ZeeMachine (CA) | (408)245-1420 | (ISDN-Maximus) |
| TCJ/DIBs BBS (CA) | (916)722-5799 | (14.4k-Wildcat) |

Special Feature

Advanced

TTL Computer

# Simplex III

Part 2

by Dave Brooks

## Simplex-III Architecture

This is the second in a series of articles describing Simplex-III, a home-designed CPU. The machine was built in the late 1970's from discrete TTL logic. In the last issue, the historical background was reviewed, and the reasons given why the machine evolved as it did. This article describes the general architecture, which borrowed heavily from the British GEC 2050, a machine on which I had several years' experience.

Since the GEC 2050 (and hence Simplex-III) used a register-to-memory architecture, it follows that every micro-cycle will normally run one memory cycle. This set the speed of the machine, and logically pointed toward a very close memory/CPU coupling, where the main internal CPU clocks are so timed as to serve also as the RAS/CAS memory clocks. The DRAMs used were 1uS cycle, so the basic machine cycle was made a little longer: 1.6uS. It should be noted that the very old DRAMs used had a different RAS/CAS timing relationship than is now usual.

Like the GEC 2050, Simplex-III is a big-endian machine, the least-significant byte of a multi-byte object being at the highest address. The "address" of such an object is defined as the address of its least-significant byte.

One irregularity is that Simplex-III advances the instruction pointer (S register) before executing the instruction. This only affects jumps: the address of an instruction is defined as that of the byte immediately below the instruction in memory. If I had ever written an Assembler for Simplex, this could have been hidden from programmers. This nasty was left in, as it simplified the micro-code considerably.

Simplex-III had 5 programmer-accessible registers, implemented in a 16-byte TTL RAM. These registers are listed in Table 1. The "address" listed is the location in the RAM (or "scratchpad").

Operands in memory were accessed by taking a 16-bit index register (one of S, X1, X2, X3), and adding an 8-bit offset byte (zero-extended to 16 bits). In the terminology of the day, the X registers were termed "index" registers, although modern parlance might name them "bases" in terms of their function. The X registers are useable either as address-bases, or as general workspace.

All instructions are 2 bytes: a function byte and a data byte. The latter may represent either an address offset or literal data, zero-filled to 16 bits as necessary. The function-byte was regarded as the "least significant" of the byte-pair, hence it occupied the higher memory address.

Register operands were one of A, X1, X2, X3. When an index (X) register is the operand, the data length is fixed at 2 bytes. The "effective length" of A could be set from 1 to 8 bytes, by loading an internal register with the desired length. This length then persisted until changed again.

## Bit Numbering

Consistent with the big-endian architecture, the most significant bit of a register is Bit-0. Bit groups are referred to as, for example, I[0:7], meaning Bits 0 through 7 of Register I, with bit 0 most significant.

## Organisation

The register layout is shown in Fig. 1, the principal hardware components being listed in Table 2. Internal data paths are 8 bits wide, with longer operands being processed by repeated cycles. Address pointers auto-index at each microcode state (or "box"). The store address auto-decrements, while the scratchpad pointer auto-increments. Multi-byte objects are processed by repeating the current microcode state as needed. This is much more efficient than using explicit loops in the microcode.

All the programmer-visible registers (except C) are in the scratchpad RAM (named "SPAD" in the drawings). This register bank is duplicated, for interrupt and base level operation. Consequently an environment switch between interrupt and base level takes essentially zero time, the register-bank being switched as the current instruction completes.

"R" is the "anti-race register", needed as the SRAM was not edge-triggered: it could not do read-modify-writes. The "R" register was edge-triggered, and could serve as an additional temporary data store.

RFA is the "refresh address" (5 bits in the 1kb DRAMs I was using). It is multiplexed on to the address bus when the CPU does not require the bus for data transfers.

IPL denotes "initial program load", ie a boot ROM. In fact this ROM code was never implemented: boot code was loaded by hand from the panel switches.

## Clocking

The master clock was a 9.47MHz crystal, which happened to be in the junk box. The clock signals are shown in Fig. 2. A full micro-cycle is 16 cycles of the crystal, or 1.68uS. This cycle corresponded to a read/modify/write cycle of the DRAM. The clocking used a 4-phase model, with the 4 primary clocks named after Greek letters. The clock nomenclature is listed in Table 3.

## DRAM Refresh

This was built into the CPU, in that every microcycle which did not actually need the DRAM data bus, automatically ran a refresh cycle. Since all instructions include at least two such cycles (while S is fetched and incremented), refresh is always guaranteed. With the refresh logic operating independently of the microcode (it was in effect, interrupted to run a "real" data transfer on the bus), refresh continued during CPU halts, even at the lowest microcode level.

This had another useful property: a 5-bit count was available on the backplane. This was used to drive multiplexers on each board, which scanned out the states of their internal registers, as sequential streams. These were demultiplexed on the monitor card, to drive the panel LEDs. This monitor card was in the front position in the card-frame, with a window in front of it. The LEDs are mounted directly on the card.

Hence every refresh cycle is also a monitor update cycle. Now when the machine is stopped (even at the microcycle level), every bus cycle becomes a refresh, and hence the display is constantly updated.

## Next issue

This article has described the overall architecture of Simplex-III. The next issue will describe the machine instructions, and present some program fragments, to illustrate how they were used.

**Programmer-Visible Registers**

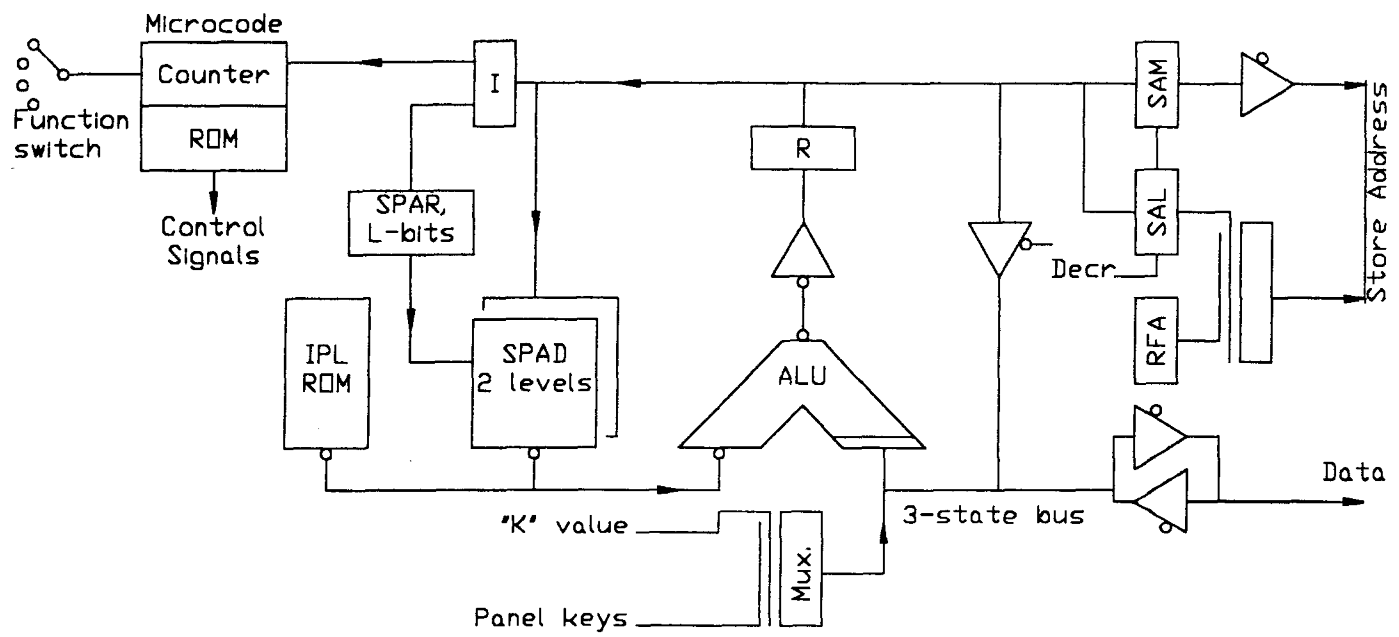
| Name | Bytes | Address | Function |
|---|---|---|---|
| S | 2 | 0..1 | Sequence (instruction pointer) |
| X1 | 2 | 2..3 | Index / data register |
| X2 | 2 | 4..5 | ditto |
| X3 | 2 | 6..7 | ditto |
| A | 1..8 | 8..F | Workspace (accumulator) |

**Major hardware items:**

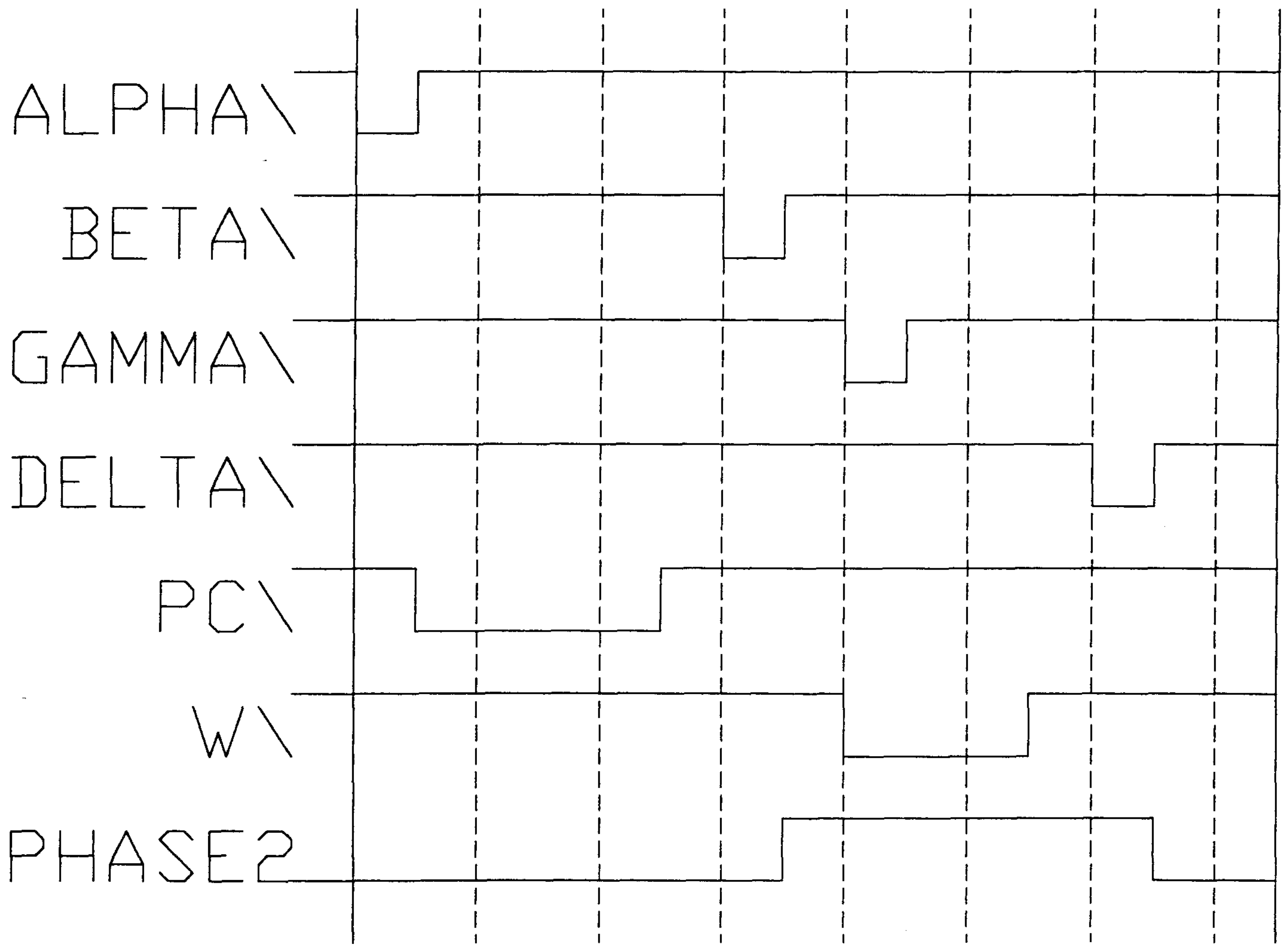
| | |
|---|---|
| Microcode counter | 5-bit microcode state address |
| Microcode ROM | 32 x 16-bit ROM (diode matrix) |
| IPL ROM ROM | 32 byte bootstrap loader code |
| SPAR | Scratchpad address register (4 bits) |
| SPAD | Scratchpad (programmer-accessible registers), broken into |
| A | Accumulate (length 1..8 bytes selectable) |
| X1 | Index register (for store addresses) |
| X2 | ditto |
| X3 | ditto |
| S | Instruction pointer |
| I | Current-instruction register |
| ALU | Arithmetic-logic unit |
| R | ALU Result register / anti-race latch |
| SAL, SAM | Store address register (2 bytes) |
| RFA | Refresh address (5 bits, for 1kb DRAM) |
| C | Condition codes (Z, N, CA & length for A) |

**Outputs from the clock generator:**

| | |
|---|---|
| ALPHA\ | Preset carry, clear R, count microcycles, register addr. |
| BETA\ | Load R, condition-bits, carry |
| GAMMA\ | Strobe in results |
| DELTA\ | Count memory address, step microcode |
| PC\ | Precharge for DRAM |
| W\ | Write for DRAM & IO |
| PHASE2 | Sets IO buffers to output |



Microcode
Counter
ROM
Function switch
Control Signals
I
SPAR, L-bits
IPL ROM
SPAD 2 levels
R
ALU
"K" value
Panel keys
Mux.
3-state bus
Decr.
SAM
SAL
RFA
Store Address
Data



ALPHA\
BETA\
GAMMA\
DELTA\
PC\
W\
PHASE2

# *TCJ* Center Fold
# P112 board

by David Brooks

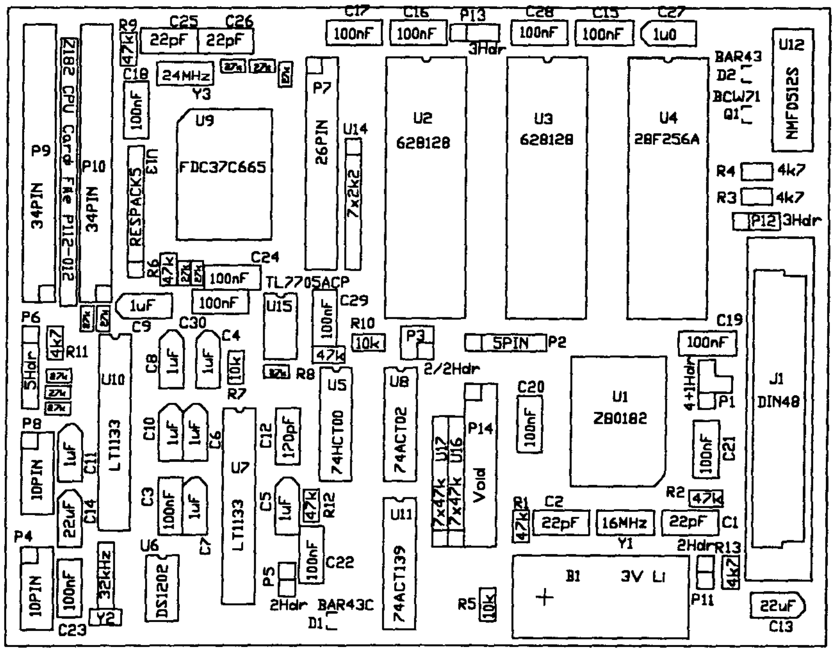
Special Feature

All Users

CP/M System

## THE "P112" BOARD FEATURES

Dimensions: 130 x 100mm (5.1 x 3.9 inch)
Support for 5.25 and 3.5 inch diskette drives (up to 4 drives, mixed types)
Z80182 CPU at 16MHz (12.228, 18.432 or 24.576MHz optional)
32kB flash ROM, in-board reprogrammable
64kB SRAM, upgradeable to 1MB
5V-only power supply (150mA plus drives)
Real-time clock/RAM, with on-board battery
5 (yes, five) serial IO ports, 2 as PC-AT compatible connectors, 3 as TTL outputs
Parallel port, IBM compatible, with bidirectional ability
Bus expansion/logic analyser socket
Software included:
 Shareware DOS+ and CCP+ (replace CP/M)
 Shareware PPIP (replaces PIP)
 Shareware UUENCODE & UUDECODE
 BIOS support for diskettes, parallel & RS232 serial ports, ROM monitor, including debugger



This project originated in a thread of about a year ago, on the 'comp.os.cpm' Usenet newsgroup. A number of people observed that their trusty old 8-bit systems were starting to show their age, and regretted that no current designs were available to run CP/M [tm] or similar software. My own CP/M platform was an ageing "Little Big Board" (Pulsar Electronics, Australia), in a STD bus format, running a pair of 5-inch drives. It too, was due for replacement.

It happened that this expressed need chimed with a long-standing dream of my own. Having designed from scratch a Z-80 powered commercial system (has anyone heard of the "Index 2000?") back in the late 70's, I had a yen to rework that idea using modern parts. I just needed an excuse. A first look showed that it should be feasible to put the whole thing in the same form factor as a 3-inch drive, and bolt it straight to the drive.

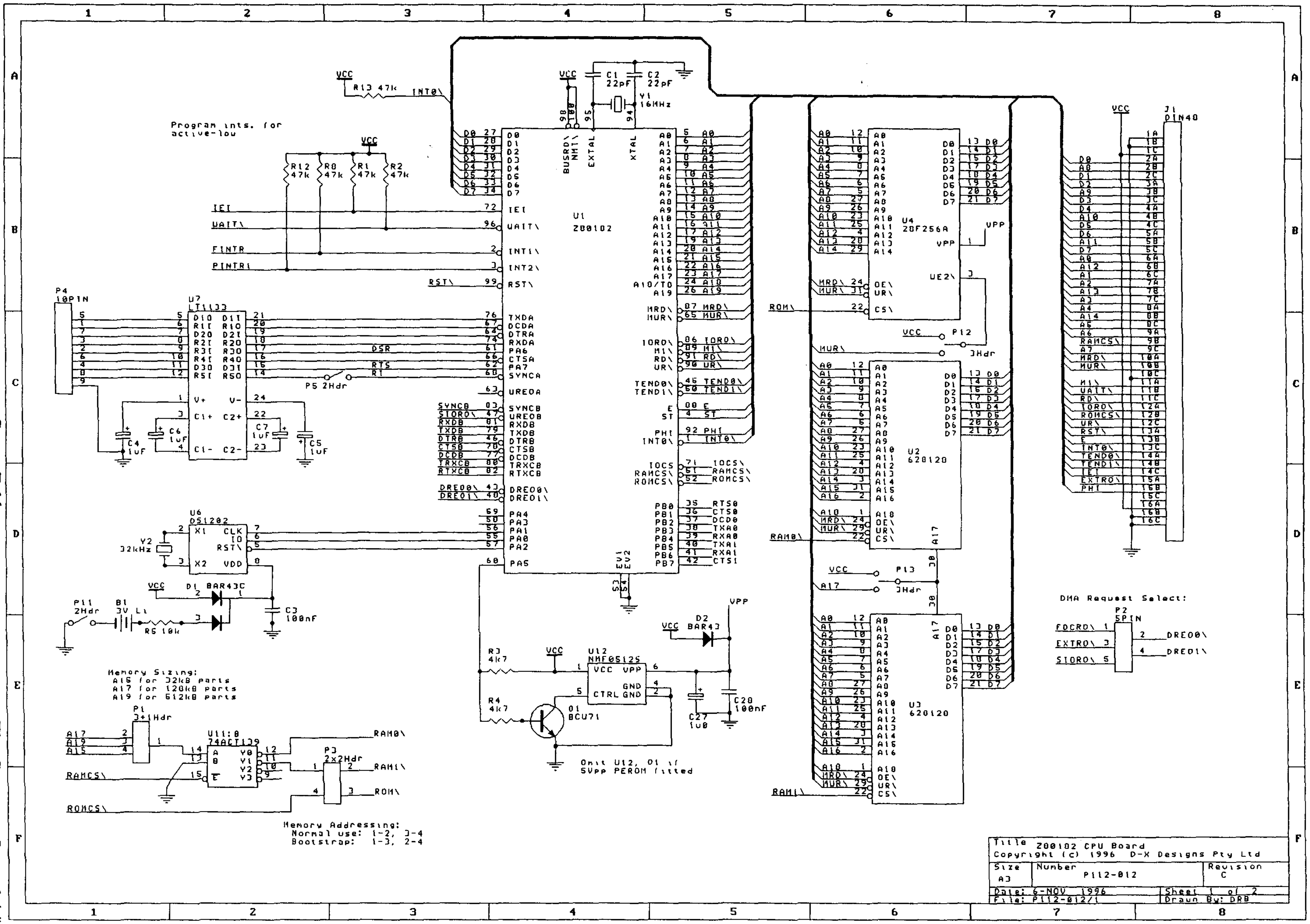
Discussions on-line hammered out a rough consensus. My original idea had been a little too high-tech: I had planned to use surface-mount devices throughout, with glue logic in FPGA's (I use these techniques in my working life). This was far more than the prospective users wanted: they were looking for through-hole technology to assemble themselves.

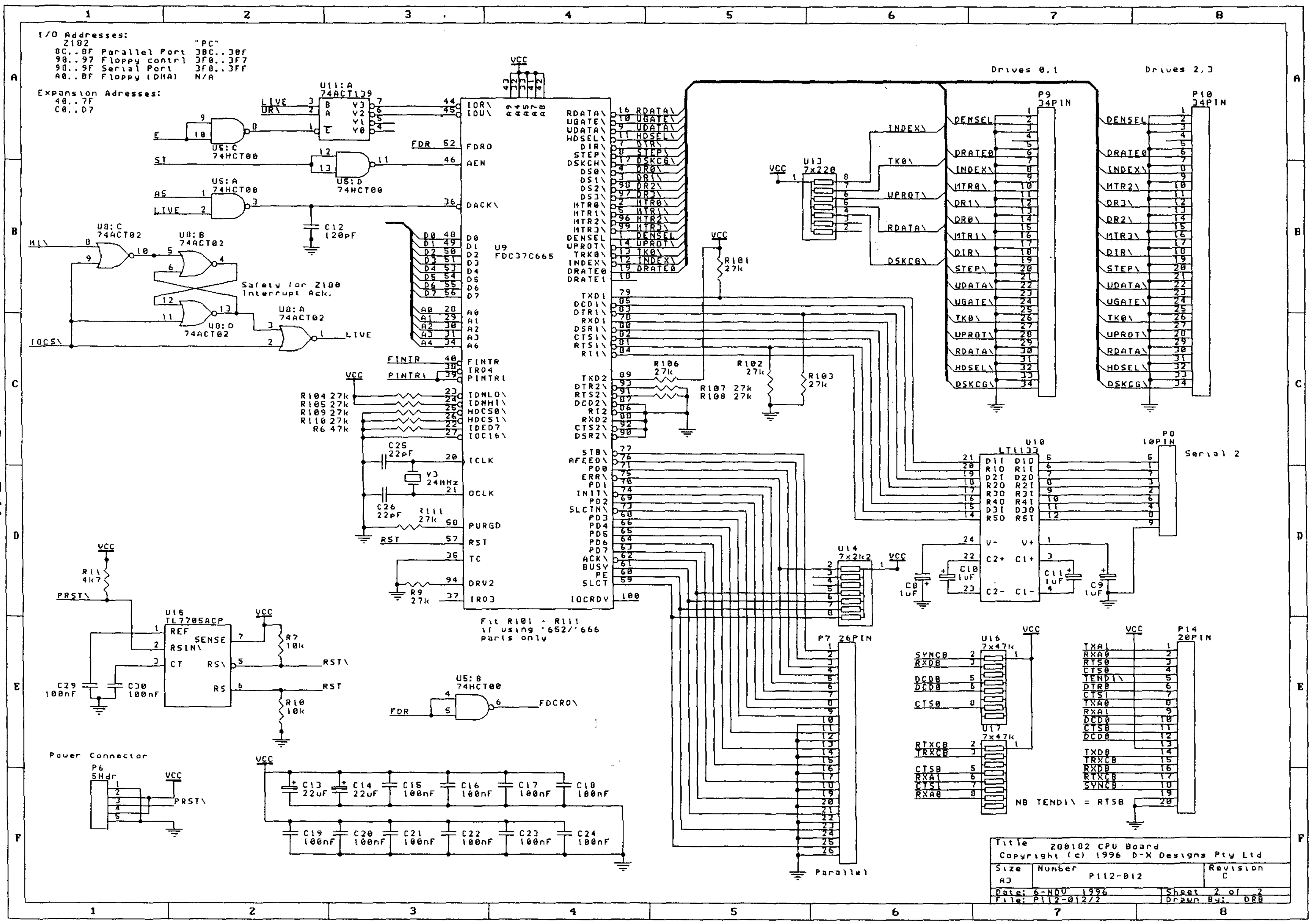
With the idea officially up and running, it was allocated the next sequence number on my Company's project register - 112. That of course, is the origin of the name "P112", if anyone wondered.

It was early decided to provide a PC-like parallel printer port, as serial printers are now rare birds. With two serial interfaces besides, this would be adequate for most purposes. A bus-expansion connector was an obvious necessity, both for add-on projects and for initial testing. Since the 3-inch drive form factor is not mechanically compatible with any standard bus, there seemed little point in more than a simple trace-out of the CPU pins. It was anticipated that the majority of applications would use the board stand-alone, anyway.

There then ensued a long search to find parts with a reasonable life expectancy. User feedback had shown that the original 64kB memory space was inadequate; some form of mapping was needed. Since this logic could not be put in a FPGA, the Z180 core architecture became attractive, as this function is included. The Zilog Z182 rapidly emerged as the best CPU choice, and includes sufficient serial IO for the purpose. Enough spare parallel IO pins were also available to emulate a basic printer port.

The disk controller proved a long hunt. All the old standards are now on "death row", being due to be dumped from


Program ints. for active-low
U1 Z80182
U4 28F256A
U2 628128
U3 628128
U7 LT1133
U6 DS1202
U11:B 74ACT139
U12 NMF0512S
Q1 BCU71
J1 DIN40
P4 10PIN
Memory Sizing:
A16 for 32kB parts
A17 for 128kB parts
A19 for 512kB parts
Memory Addressing:
Normal use: 1-2, 3-4
Bootstrap: 1-3, 2-4
Omit U12, Q1 if 5Vpp PEROM fitted
DMA Request Select:
Title Z80182 CPU Board
Copyright (c) 1996 D-X Designs Pty Ltd
Size A3
Number P112-012
Revision C
Date: 6-NOV 1996
Sheet 1 of 2
File: P112-012/1
Drawn By: DRB


I/O Addresses:
Z182 "PC"
8C..8F Parallel Port 3BC..3BF
90..97 Floppy contrl 3F0..3F7
98..9F Serial Port 3F8..3FF
A0..BF Floppy (DMA) N/A
Expansion Adresses:
40..7F
C0..D7
U11:A 74ACT139
U6:C 74HCT00
U5:D 74HCT00
U5:A 74HCT00
U8:C 74ACT02
U8:B 74ACT02
U8:D 74ACT02
U8:A 74ACT02
Safety for Z180 Interrupt Ack.
LIVE
IOCS\
C12 120pF
U9 FDC37C665
VCC
Drives 0,1
Drives 2,3
P9 34PIN
P10 34PIN
U13 7x220
R101 27k
R102 27k
R103 27k
R106 27k
R107 27k
R108 27k
R104 27k
R105 27k
R109 27k
R110 27k
R6 47k
C25 22pF
C26 22pF
Y3 24MHz
R111 27k
R9 27k
Fit R101 - R111 if using '652/'666 parts only
U10 LT1133
P8 10PIN
Serial 2
C8 1uF
C10 1uF
C11 1uF
C9 1uF
U14 7x2k2
P7 26PIN
Parallel
R11 4k7
PRST\
U15 TL7705ACP
R7 10k
R10 10k
C29 100nF
C30 100nF
U5:B 74HCT00
FDCRO\
Power Connector
P6 5Hdr
C13 22uF
C14 22uF
C15 100nF
C16 100nF
C17 100nF
C18 100nF
C19 100nF
C20 100nF
C21 100nF
C22 100nF
C23 100nF
C24 100nF
U16 7x47k
U17 7x47k
P14 20PIN
NB TEND1\ = RTSB
Title Z80182 CPU Board
Copyright (c) 1996 D-X Designs Pty Ltd
Size A3
Number P112-012
Revision C
Date: 6-NOV-1996
Sheet 2 of 2
File: P112-012/2
Drawn By: DRB

production in the near future. Eventually I selected the SMC "Super IO" line. These are designed as multi-function IO devices for use in PC's: providing full-featured disk, serial and parallel ports. Of course, these parts are designed to work in a ISA environment, and some glue would be necessary to adapt the Z180 bus. At this point, I was glad to read Claude Palm's article in TCJ (Nov/Dec 1995), describing his problems with Z80 interrupt cycles and DMA devices. Being forewarned to that problem doubtless saved several sleepless nights.

It would have been attractive to use standard 1MB SIMM modules, however the amount of glue logic required made this impractical. Z80 "DRAM support" is in practice limited to providing refresh addresses (with insufficient bits for modern parts). Address multiplexing and clock timing require additional hardware.

The board was designed to take a single ROM part, and 1 or 2 static RAMs. This enables a 64kB system to be built with two 32kB SRAMs, or a larger system using 128kB (or, for the wealthy, 512kB) parts.

The SMC part also offers "IDE support", however this is little more than address decoding. The big part of running IDE drives on a 8-bit machine (as Tilmann Reh has shown earlier in TCJ) is matching the IDE 16-bit bus to the 8-bit CPU. While this can be done in slow-time using software and two IO ports, I intended the disk to operate under DMA control.

The logic for this was prohibitive (FPGAs not being allowed), so IDE was not provided. An add-on board could be done later (as of this writing, this has not yet been done, although Harold F Bower <HalBower@msn.com> has built a SCSI controller board, along with much other improved software).

I resisted pressure to use a 2-layer board: with the CPU and IO chip only available in 100-pin flatpacks, there would be some very close signal paths just where power/ground noise would likely be worst. The decision has been justified in the reliability of the 4-layer board as built.

As few people care to install such parts at home, I have made the boards available part-assembled, with the flatpack chips and 3 surface-mounted diodes pre-installed. The remaining components are standard through-hole types.

Boot code was put in a flash ROM, although cheap glass EPROMs can be fitted. I now use flash ROM exclusively in my regular work, and the time saved fully justifies any extra cost. The ability to re-program in place is a real benefit.

## SOME THINGS LEARNED

As with all Z80 devices, there are several clock cycles for each bus transaction. This means the CPU clock is not much use to trigger a logic analyser. On the Z182, the "E" output provides exactly the information required: trigger the analyser on a negative edge at this pin, and the required information is available on the bus pins. This "clock" gives exactly one edge per transaction, as required.

Signal timing to the SMC chip can be quite critical. In particular, signals such as AEN\ and the addresses are latched on the falling edge of RD\ or WR\. If AEN\ for example, is generated by gating from the Z180's IOCS\ line, this timing is not met. See the schematics for a solution.

When driving the SMC chip's printer interface, poll for ready before (not after) sending data. After sending, the port status should not be read for about 40uS, else it will lock up and never become ready. The software overhead in fetching the next data byte will normally guarantee the delay.

## BRINGING IT TO LIFE

Initial tests were done using an old debugger module which I now use as a "standard" to bring up any new Z80 based hardware. Providing facilities similar to MS-DOS [tm] Debug, but in a hardware-only setting, this module enabled me to walk through exercising the basic hardware functions. The flash ROMs were programmed by plugging into another, working system.

The only real hardware "gotcha" encountered was in the timing of signals to the SMC chip. (Moral: RTFM, most carefully) This area was re-designed, and now works reliably.

With the hardware operating, it was time to install software. The disk driver software was written to be controlled by a table formatted identically to a PC BIOS table. This meant that such tables could immediately be "borrowed" from my PC to set things up for various drives. The use of PC-compatible formats also meant that first-cut disks could be built on a PC.

The first step was to port the old Little Big Board system (my own rewrite of the original Pulsar BIOS). With an old 5-inch drive connected to the P112, an old LBB disk was copied, and the BIOS sectors overwritten for a first system disk. After the usual bug fixes, this worked. CP/M was now live on the P112.

The first BIOS included format tables for 5 and 3-inch disks. A crude "format" program was written, to run under CP/M, and to accommodate both disk sizes. This built the first 3-inch disk.

With the system booted from the 5-inch disk, standard utilities could now copy program files between the disks, and soon a true 3-inch version was running.

To avoid copyright problems, the distribution disks have CP/M replaced by shareware equivalents. Harold Bower <HalBower@msn.com> has also written an enhanced BIOS, and improved my orginal disk format. These improvements are available via my web-site.

---

*Dave Brooks is the head of D-X DESIGNS PTY LTD, 7 Buchan Close, SPEARWOOD, Western Australia 6163*
*Tel/fax: +61 9 434 4280*
*Email: daveb@iinet.net.au*
*Web page: http://www.iinet.net.au/~daveb*

*With over 25 years digital design experience, D-X Designs Pty Ltd provides a custom design service for digital equipment. We specialise in embedded applications, with experience in 8051, 8085, Z-80 and 80186 processors. Software capability includes Assembler, Pascal, Fortran C and C++. Our second speciality is design for Xilinx FPGAs, for which we have a full tool-set.*

# Real Computing

By Rick Rodman

32-Bit Systems

All Readers

Real-time Control

**REAL COMPUTING**

According to Greek legend, Damocles had to eat an entire meal with a sword dangling above his head, suspended by a single human hair. It was not conducive to good digestion. This is thought to be one of the first demonstrations of the importance of reducing system overhead.

**The Distributed Real-Time Control System**

Home automation is one of my favorite computer applications. Some approaches I've tried have included S-100 boards driving relays and VIC-20 and C-64 machines with X-10 interfaces. The problem always seems to come down to where to put a central controller. If it's in my bedroom, it makes too much noise for me to sleep at night. If it's in the furnace room, it's not very handy to use.

My latest design uses a controller machine which is left on all the time, connected to the LAN so that other machines can access it. That way, once I run Ethernet up to my bedroom, I'll be able to turn on my laptop and issue a request, without having to permanently dedicate my laptop to that purpose.

Actual real-time control is performed by a machine which, as I mentioned, must be powered up all the time. At present, this machine is an IBM PS/2 model 30, which is actually a type of XT. It operates an X-10 Powerhouse CP-290 interface. It has a 3COM Ethernet card and a Watson telephone voice card.

I've described the CP-290 before, and have written simple routines to interact with it. I also have a TW-523, but it appears to require fairly constant CPU attention, which I don't think I can give it at present. The TW-523 has a capability of interrogating modules to determine their current state. However, I think this is a basic X-10 feature which I will also be able to do with a CP-290.

The software on the Model 30 listens for requests on its Ethernet card. On my Sun, a simple C program makes requests through the net based on command-line parameters. Soon, this C program will be invoked by a Perl script activated by a button on a HTML script, allowing device status and control from a Web page.

Where does the system run? It runs, in part, on each machine. In fact, it could have more pieces running on other machines besides those two.

As I discussed last time, I had been working with TinyTCP for this project. However, after much labor, I had to set it aside - even TinyTCP just had too much overhead - and started afresh to design something even simpler. If Mr. Baldwin permits, I will accompany this article with "snippets" showing key code parts. I think this could be of interest to other folks who are using XTs for embedded projects and want to incorporate networking with an absolute minimum of code.

Before I go too much further, about the heading: I call this setup the Real-Time Control System, but there's real-time, and then there's Real-Time. The software on the Model 30 is basically a big polling loop. There aren't any interrupts. I get around to the Ethernet when I get around to it, and the CP-290 operates at a leisurely 600 baud. Some folks will say that Real-Time means you've got to have a multitasking kernel with priority-based preemptive scheduling, interprocess communication, automatic deadlock detection, and so on. I say, real-time just means "fast enough", and if the Model 30 runs out of gas, there are three 286s in the storage room. But I like the Model 30 because it's small.

Basically, I started by snipping small parts from the 3C501 driver of an old version of KA9Q. I have three of these boards, each of which I've acquired for a dollar or less. No matter what people say about them, they work.

The first thing to get working, after a simple packet receive routine (I didn't bother with using interrupts either), is an ARP (Address Resolution Protocol) routine. Every Ethernet board has a 6-byte unique ID burned in its PROM. In Novell IPX, they use the Ethernet ID directly, but in IP, there has to be a mapping performed. One machine who has never spoken to another must broadcast an ARP request to determine the Ethernet address that corresponds to the second machine's IP address. ARP is described in RFC 826.

At the outset of this project I decided to use no data structures beyond simple arrays of shorts. Thus the constants IPH_SRC_IP_L, and so on, are offsets into an array. The entire message is passed to each layer, so all underlying addresses are always available. Listing 1 shows my routine for distributing incoming packets, and listing 2 shows my ARP routine. Notice that it does as little as necessary.

Once you've got ARP working, you can just go on and do TCP or UDP (as did TinyTCP's authors). However, it's really nice to be able to "ping" your machine as a check of your work. To do this requires that you implement a sliver of ICMP (Internet Communications Management Protocol). And I do mean a sliver - just a ping responder, nothing else. Listing 3 shows my entire IP layer. ICMP, TCP, and UDP are layers above IP. Listing 4 shows the ping responder. No matter how simple you make it, you still have to compute two checksums for the response message. I don't waste time *checking* any checksums, though. Since ICMP is replying to a sender, and the entire message is always passed, a reply can be fixed up by copying the original message and swapping source and destination addresses.

At this point, you're ready for TCP or UDP. I decided I didn't really need TCP, with its guaranteed delivery, transparent stream protocol. I just want to send short commands and get short responses. UDP (User Datagram Protocol) is a better match for this requirement. UDP is very tersely described in RFC 768. Again, there are two checksums to calculate, with one based on a "pseudoheader" that's built but never transmitted, and if anything's wrong your message is ignored. Listing 5 shows the UDP layer. Listing 6 shows the Sun side, which is much simpler since networking is built into SunOS.

You may think that not checking checksums is rather unsafe. Actually, since Ethernet itself has a CRC that will pretty much guarantee good data, I don't worry about it. If a SLIP interface were to be used instead, there'd be a danger of lost data, and checksums should be checked.

Now, I'll be the first to admit that this kind of implementation is not going to work well in multiuser, high-data-rate applications. But then a Model 30 doesn't have CPU power for that kind of work anyway. My whole intent is to have a 24-hour control system running in a cheap XT - if lightning kills it, roll in another one - while other machines can be turned on and off without affecting it. I haven't got the voice board integrated yet, but if things work out, I may merge a bulletin board into it, too!

**Next time**

Next time we add passwords to the BBS to stop pranksters from turning my lights on and off. Just kidding!

**For more information**

Kettle Pond Computing Facility BBS or Fax: +1 703 759 1169
E-mail: ricker@erols.com
Mail: 1150 Kettle Pond Lane, Great Falls VA 22066-1614

**LINUX**

| | |
|---|---|
| InfoMagic 5 CD Set | $21.95 |
| Yggdrasil | $29.95 |
| Linux man Pages | $29.95 |
| The New Book of Linux | $29.95 |

Call for other titles

**www.justcomp.com**
*on the World Wide Web*

***JUST COMPUTERS!***
**(800) 800-1648**
Fax (707) 586-5606 Int'l (707) 586-5600
P.O. Box 751414, Petaluma, CA 94975-1414
E-mail: sales@justcomp.com
Visa/MC/Int'l Orders Gladly Accepted
For catalog, send e-mail : info@justcomp.com
Include "help" on a single line in the message.

```
===== Listing 1: Basic Ethernet poll routine. ============

 paklen = maybe_receive_packet( &recv_buffer[ 0 ], BFRSIZ );
 if( ! paklen ) continue;

 i = ( recv_buffer[ 12 ] << 8 ) + recv_buffer[ 13 ];
 switch( i ) {    /* Process by protocol */
   case 0x0806:     /* ARP */
      check_arp( ( unsigned short * ) &recv_buffer[ 0 ],
           paklen );
      break;
   case 0x0800:     /* IP */
      process_ip(( unsigned short * ) &recv_buffer[ 0 ],
           paklen );
      break;
   default:
      printf( "Unimplemented protocol\n" );
 }

===== Listing 2: ARP routine. ==================

static int check_arp( unsigned short *p_recv, int len ) {
     unsigned short     reply[ 30 ];

     if( *( p_recv + 6 ) != 0x0608 ) return 0;
     if( *( p_recv + 10 ) != 0x0100 ) return 0;

     /* my address = 129.212.32.32 = 81 D4 20 20 */

     if(( *( p_recv + 19 ) != 0xD481 )
          || ( *( p_recv + 20 ) != 0x2020 )) return 0;

     /* put two ethernet addresses */

     memcpy(( unsigned char * ) &reply[ 0 ],
          ( unsigned char * )( p_recv + 3 ), 6 );
     memcpy(( unsigned char * ) &reply[ 3 ], &ethernet_address[ 0 ], 6 );

     reply[ 6 ] = 0x0608;     /* ARP */
     reply[ 7 ] = 0x0100;     /* ethernet */
     reply[ 8 ] = 0x0008;     /* protocol = IP = 0x800 reversed */
     reply[ 9 ] = 0x0406;     /* lengths, hw/prot, reversed */
     reply[ 10 ] = 0x0200;     /* ARP reply */

     memcpy(( unsigned char * ) &reply[ 11 ], &ethernet_address[ 0 ], 6
);
     reply[ 14 ] = 0xD481;
     reply[ 15 ] = 0x2020;
     memcpy(( unsigned char * ) &reply[ 16 ],
          ( unsigned char * )( p_recv + 11 ), 10 );

     send_packet(( unsigned char * ) &reply[ 0 ], 42 );
     return 1;
}

===== Listing 3: The IP layer. ===============

int process_ip( unsigned short *p_buffer, int len ) {
     int          ip_protocol;

     /* As before, the first 7 words are the source and dest
          ethernet addresses */

     if( *p_buffer == 0xFFFF ) return 0;     /* Ignore broadcasts */

     /* If it came to us without being a broadcast, we can assume
          it's for us. */

     ip_protocol = ntohs( *( p_buffer + IPH_TTL_PROT )) & 0xFF;
     switch( ip_protocol ) {
     case 6:          /* TCP */
          return 0;          /* I don't process */
     case 17:     /* UDP */
          return process_udp( p_buffer, len );
     case 1:          /* ICMP */
          return process_icmp( p_buffer, len );
     default:
          printf( "Unknown IP protocol %04x\n", ip_protocol );
     }
     return 0;     /* not processed */
}

===== Listing 4: ICMP (Ping) responder. ==============

static int process_icmp( unsigned short *p_buffer, int len ) {
     unsigned short          reply[ 60 ];
     unsigned short          type_subtype;

     /* type-subtype is unreversed. First byte = 8 for echo request */

     type_subtype = *( p_buffer + IPH_WORDS );

     if( type_subtype != 0x0008 )     /* Not Ping request? */
          return 0;

     /* copy input message to reply */

     if( len > 120 ) len = 120;
     memcpy(( unsigned char * ) &reply[ 0 ],
          ( unsigned char * ) p_buffer, len );

     reply[ IPH_WORDS ] = 0;     /* ping reply */

     /* reverse the addresses */

     reverse_eth_addresses( reply );
     reverse_ip_addresses( reply );

     /* fix ICMP checksum */

     reply[ IPH_WORDS + 1 ] = 0;
     reply[ IPH_WORDS + 1 ] = htons( ~
          checksum( 0, &reply[ IPH_WORDS ],
               len - ( 2 * IPH_WORDS )));

     /* fix IP checksum - checksum of ip header only */

     reply[ IPH_CHECKSUM ] = 0;
     reply[ IPH_CHECKSUM ] = htons( ~
          checksum( 0, &reply[ ETH_WORDS ],
               2 * ( IPH_WORDS - ETH_WORDS )));

     /* send the message */

     return send_packet(( unsigned char * ) reply, len );
}
```

```
===== Listing 5: UDP layer. ==================

int process_udp( unsigned short *p_buffer, int len ) {
     unsigned short    reply[ 60 ],
                       ph[ 6 ],
                       src_port, dst_port, partial_sum;
     int               request_len, reply_len;

     /* Don't bother reversing port addresses */

     src_port = *( p_buffer + UDPH_SRC_PORT );
     dst_port = *( p_buffer + UDPH_DST_PORT );
     request_len = *( p_buffer + UDPH_LENGTH );

     /* Null-terminate the request length */

     * ((( char * )( p_buffer + UDPH_WORDS )) + request_len ) = '\0';

/* Here the command in the buffer is passed to the command
subroutine. The response is placed in the reply buffer. */

     reply_len = strlen(( char * ) &reply[ UDPH_WORDS ] );
     if( reply_len & 1 ) ++reply_len;     /* make it even */
     len = UDPH_WORDS + reply_len;         /* output length */

     reply_len += 8;          /* UDP length includes UDP header */

     /* copy input message to reply */

     memcpy(( unsigned char * ) &reply[ 0 ],
          ( unsigned char * ) p_buffer, 2 * UDPH_WORDS );

     /* reverse UDP ports */

     reply[ UDPH_SRC_PORT ] = dst_port;
     reply[ UDPH_DST_PORT ] = src_port;
     reply[ UDPH_LENGTH ] = htons( reply_len );

     /* reverse the addresses */

     reverse_eth_addresses( reply );
     reverse_ip_addresses( reply );

     /* fix UDP checksum. We have to build a 'pseudo-header'
        and checksum that plus the data. */

     reply[ UDPH_CHECKSUM ] = 0;

     ph[ 0 ] = reply[ IPH_SRC_IP_H ];
     ph[ 1 ] = reply[ IPH_SRC_IP_L ];
     ph[ 2 ] = reply[ IPH_DST_IP_H ];
     ph[ 3 ] = reply[ IPH_DST_IP_L ];
     ph[ 4 ] = ( 17 << 8 );          /* protocol */
     ph[ 5 ] = reply[ UDPH_LENGTH ];

     partial_sum = checksum( 0, &ph[ 0 ], 12 );

     reply[ UDPH_CHECKSUM ] = htons( ~
          checksum( partial_sum, &reply[ IPH_WORDS ],
               len - ( 2 * IPH_WORDS )));

     /* fix IP checksum - it is the checksum of just the header */

     reply[ IPH_CHECKSUM ] = 0;
     reply[ IPH_CHECKSUM ] = htons( ~
          checksum( 0, &reply[ ETH_WORDS ],
               2 * ( IPH_WORDS - ETH_WORDS )));

     /* send the message */

     return send_packet(( unsigned char * ) reply, len );
}
```

```
=== Listing 6: Client code from Sun (excerpted). =====

     soc = socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );
     if( soc < 0 ) return;

     bzero(( char * ) &sin, sizeof( sin ));
     sin.sin_family = AF_INET;
     sin.sin_port = 0x9999;
     sin.sin_addr.s_addr = htonl( INADDR_ANY );

     if( bind( soc, ( struct sockaddr * ) &sin, sizeof( sin )) < 0 ) {
          printf( "bind error\n" );
          close( soc );
          return;
     }

     /* try to send a datagram */

     bzero(( char * ) &sout, sizeof( sout ));
     sout.sin_family = AF_INET;
     sout.sin_port = 0x9999;
     sout.sin_addr.s_addr = inet_addr( "129.212.32.32" );

     nbytes = sendto( soc, p_command, strlen( p_command ),
          0, ( struct sockaddr * ) &sout, sizeof( sout ));
     if( nbytes <= 0 ) {
          close( soc );
          return;
     }

     printf( "awaiting response...\n" );

     addrlen = sizeof( sout );

     nbytes = recvfrom( soc, recvbuf, sizeof( recvbuf ), 0,
          ( struct sockaddr * ) &sout, &addrlen );

     if( nbytes > 0 ) {
          recvbuf[ nbytes ] = '\0';
          printf( "received response: %s\n", recvbuf );
     } else printf( "recvfrom returned %d\n", nbytes );

     close( soc );
}
```

# Embedded Development Choices

By Bill Kibler

Special Feature

Embedded Support

Which Tools

I received this letter asking for help and think other *TCJ* readers might be interested in my answer. The situation involves choosing embedded control software and tools.

**The Letter**

*I have had the opportunity to coach Odyssey of the Mind (OM) and have really enjoyed working with kids and teaching them some rather diverse things. My 3rd daughter built a 5' walking (not rolling) robot when she was 8. To make it work she had to learn ratios, and she hates math, and the use of many hand and power tools.*

*I have a team of 5th graders now working on the ride-on vehicle problem. My son wants to use a computer to control part of their scenery and some of the mechanical tasks associated with it. I got a clarification from World OM allowing them to use interface cards which they did not build, so I have given them a hand built card with an 8255 on it. It has relays for output on port B and an 74LS244 as an input buffer on port A. He is learning to program it in C++ (Borland 3.0) and doing quite well.*

*This brings me to a problem that you might be able to help me with. They also want to use an embedded system to control part of their project. I have a CPU board out of a Prolog STD bus system with an 8085 and 2732 ROMs on it that will work real well. I have a cross assembler to run on my PC and a PROM cooker, but the kids don't really have time to learn assembly language too; their tournament is in March and they have tons of work to do. I would like to find a CHEAP! (shareware?) C cross compiler that would run on the PC under MS-DOS and generate 8080/8085 code. As I have only recently returned to work after being laid off for 8 months, I can't afford to buy much right now. My CP/M system died a few years ago taking with it my Pascal->8080 compiler, so I have no high level language support for my 8 bit stuff, including lots of Z80 boards.*

*If you can help us out, Hunters Glen Elementary Amusin' Cruisin' Team A will be for ever grateful.*

*Arlyn Whitchurch, Thornton, CO.*

**The Reply**

Ok Arlyn, great going. To answer your letter I need to start with my standard reply and go beyond. Our last years Computing Hero, David Jaffe, uses Forth embedded in a Z180. He has also used it very successfully with college students in an engineering packaging project much like your OM work. He finds the students able to pick up the language quickly and move on to the project without getting lost in the language itself. You might want to contact him via Internet (jaffe@roses.stanford.edu ) and read issue #71 for a quick review of his work.

Should you decide not to use one of the many Forth's for ROM and embedded work, your best option is learning assembly. Why assembly, mainly because it is simple and very straight forward and more importantly there are plenty of cross assemblers available for free. Remember too that almost all assemblers work the same, yet high level languages are very different.

As to C, there are no free C cross assemblers specifically for what you are doing. As close as I can come is using Small-C with assembly output for the 8080. There are several commercial versions, the cheapest of which I think starts at over $300. We have over the past few years talked considerably about Small-C and you might want to get issue #64 where we talked in detail about using it. Small-C does work, but I am afraid you or your students might get caught spending more time making the C work for them than getting the robots to run.

**First Steps**

Which ever way you decide to go, the first step is seeking resources for the needed code. We have run numerous articles over the years about working on robotic systems and using various languages. Dave now has a special price for all back issues should you not have them, and I can not stress enough the need to search back through the past for help.

For people with access to PC Clone's and a CDROM device, a number of CDROMs would be my next choice. Walnut Creek CDROM has about the best assortment to choose from. Their CP/M, MSDOS, and SOURCE disks will each produce useful support software. I listed what I found on each separately as there is plenty to keep you busy. Of course I have no guarantees any of them will work or be bug free, but many come with source, so you can play with them.

All the programs on the CDROM are available on the Internet or BBSs'. Try our BBS or web site for a start, but Walnut Creek and Oakland Internet sites also have the same programs from the CDROM. The *TCJ* Home Page has listings of other support locations and a few good nights following those threads should provide more support.

**An Option**

On our BBS is Brad Rodriguez's

Camel Forth. I have used his 8051 version and was very happy with the easy setup needed. I know of several other Forths' that you might seek in place of his. EFORTH is very popular for what you are doing, and the major versions are intended to be assembled using the PC DOS's MASM. MASM is a MACRO assembler and the cross assembly is done using tons of MACROS. Following the code is very hard, but several of us have ported it to normal assemblers. I did one for the 68K and the FIG Internet site should have the others.

The advantage of using these ROM based Forths is the ease with which they can be ported. For Brad's 8051 code I added two lines of code for the serial I/O (I was using RS485 and needed to toggle the TX/RX driver chip) and changed a few equates that pointed to the RAM space. I assembled it, burned the ROM, and was running Forth and checking out the system. At this point your students could start generating and testing code to talk and do the robotic operations.

With this type of system, you can pretty much learn by trail and error using a book like "Mastering Forth" (got my last copy for $3 at a discount book store). It will help if they understand some of the hardware and especially talking to I/O devices. Even if Forth is not used, knowing how ports talk is very important. Before we had embedded Forths, MONITOR programs were the only tools available.

## MONITORS

For years I used ROM based MONITOR programs that allow you to load programs, single step through programs, disassemble code, and make patches in RAM. They seldom have the higher level feel you get from Forth, but once learned you can do as much with them as with Forth. A few tricks are needed, such as having the ROM write a few jump to RAM and RETURN code sections. Often I put whole jump tables in RAM so you can patch around a bad section of code.

You will also find with the 8080 CPU, that it is missing a few key instructions that make life easier and monitors hard to use. With Z80's, you can load the C register with a port address from the command line and then do a read or write on that port. To do the same with an 8080 requires setting up a RAM location with read and write to port operation. You then need to change the port address (written into RAM) before calling that location. These are all things you need to look for in a good monitor program. I saw several monitors listed on the CDROMS.

## High Level Languages

At this point we need to review the normal method you might use with C or the Pascal you used in the past. Remember that we are talking about embedded or ROM based operations here, no disk files sitting out in the real world. So what are the steps that I have done for many years, well first off write the code in some editor program. Next is compile it, still on the host system. Follow that by burning the ROM ( and yes about 1 out of 5 times the ROM will fail for good). Put it in the system, test, find errors, patch around if it also contains a monitor, document errors, start again.

There have been a number of enhancements over the years that reduce the pain of doing the above steps. ROM emulators are by far the best change, as they can make getting the code into the machine a few keystrokes away. Some ROM emulators will allow you to edit the code while the machine is running. I built one myself about ten years ago for my S100 system. My design limited the operation to about 2 feet of cable, but the newer units can be strapped on the machine and a long serial cable used to down load the code. Cost is about $100 and up for the larger RAM/ROM units. Also some EPROM burners have an emulator option.

The latest variation on this theme is tethered systems. I have used a few Forth systems this way and am aware of a few high level C cross compilers that work this way as well. The idea is a lot like using ROM emulators, in that the code is down loaded into the machine and the monitor resides on the host. Often a small program is placed in ROM and all the load memory, single step, debug features are run remotely with embedded commands. You never see what is actually passing back and forth, all you get are screens of registers or program status. They can make the process simpler and faster and with higher level languages, they isolate you somewhat from the hardware. The Forth's start at about $100 to $300 depending on features, while the C systems start in the $500 and go to $5000 and more.

## CP/M on PC's

You mentioned that your old CP/M system died some years back. There are ways to use your PC Clone to run CP/M and re-use all those old programs. Many of *TCJ* readers have stopped fixing their CP/M systems and gone to MYZ80. This public domain program works great and tied with 22DISK can be used to read and run most CP/M programs. 22DISK can read most 5 inch disk formats on your PC and load them on the hard disk. You run MYZ80 and import the programs from the PC's hard disk, into the MYZ80 disk structure.

The are so many features and options that you really need to just get the program and try it. If all your programs are on old 8 inch disks, there are several persons who can convert them to PC disks for you. Alternately, you can get I/O cards that will allow using actual 8 inch drives with 22DISK. I will be trying out an adapter card later to see if 3.5 inch drive cables can be used with regular AT disk controllers and 8 inch drives (I think so).

Once you get MYZ80 up and running (there are other similar emulators on the CDROM's ) you can run your old programs, but you will still be in the compile burn output mode. Several Forth's for CP/M are available, small-C, and even BASIC source code you could put in ROM.

## Last Word

My main concern with any solutions, even MYZ80, is not getting hung up in the tools, but making sure you spend all your time doing the robotic project. I got into computers in order to solve solar energy problems. Needless to say I have not run a solar program in years!

My order of preference would be using Forth in ROM, followed by assembler, and begrudgingly using C. The

Forth would allow me to build the project slowly with working routines. The assembler would be built on top of a monitor that would contain simple robotic commands, such as RF4 (Robotic debug operation, Forward move, 4 steps). I could then write the program that just called all the same routines resident in the monitor program.

I list C only as a last resort after experience with a commercial version. We bought it thinking it would help us get fast C code for an 8051 project. It came with sample code, but the sample was for some math options, not talking with the I/O. The only code provided that could be used for a guideline or base to start from, was the Intel 8051.asm monitor programs (available from many BBS's). What we wanted was to see how they might initialize and do serial I/O with their C code. I have been told by others that you typically do all the normal stuff in assembler and leave the C coding for the more difficult math and text string problems. That being the case we decided to just hack our normal assembler source into the new project, since we knew it worked. We never did have any time to just play with the C side of the package.

Well Arlyn, that pretty much sums up my choices and options for solving your problem. I would like to hear what you actually end up doing, and especially more about the Odyssey program.

## Program Snippets

To help see how you might handle a simple problem with the three language options I presented, I decided to use last issues traffic signal Forth Day Lunch contest as the basis for some code samples. In issue #77, my Computer Corner explained about using the PC parallel port to emulate a set of traffic signals. LEDs were used for the traffic lights, and push buttons acted as vehicle sensors.

We were able to move quickly from reading and writing the parallel port to turning on the lights in various sequences. We hit a stumbling block when it came to the vehicle sensors. Based on the sensors output, you do nothing, or start the change from N/S being Green, to the E/W changing to Green. We first tried combining the testing within the changing routines. That proved very problematic and we ran out of time before a good solution was found.

Since then I have decided the program should start up and drop into the read sensor routine. If sensors are not active, it defaults into the N/S is green, E/W is stop (was specified in the design to do that on power up). Then based on the sensors, you have one of sixteen options that determine what happens next. If N/S is green, and a vehicle arrives at the East sensor, you need to start the change from N/S to E/W option.

To simplify the procedures, only two sub routines are needed, changing from N/S to E/W, and changing from E/W to N/S. All other procedures are just to determine if the change is needed or not. The vehicle in the East lane, has the port returning a 40hex value. So one way of doing the main loop then is a simple case statement on the 16 possible options (16 being all four sensors have cars sitting on them).

The code for the 40hex option would go something like this, see if N/S is green, if yes do change, else must have vehicle waiting to make left turn, start left turn option, else do nothing. Of course some have special considerations like all four have vehicles waiting. What I have provided is how I would read the parallel port, mask the bits I want, and then do a simple case statement. I'll start with the assembler first since it is the simplest.

```
Pseudo Assembler

Main_Loop:              ; all sub programs jmp back to
                        ;  here
    INA  PORT_S         ; get status port value
    AND  A,0F0H         ; only want top bits
    SHR  A,2            ; shift it right 2 places F0 = 6C
    LDB  TABLE          ; load B/C register with pointer
                        ;    to table
    ADD  B,A            ; adds A or offset into table
                        ;  with table
    LPC  (B)            ; load and jump to program
                        ; pointed to by what B points to.
;
TABLE:                  ; contains addresses of each sub
                        ;  program
    DW   ONE            ; will put address of routine
                        ; one here
    DW   TWO            ; if bit two is on, this
                        ;  procedure is run
    DW   THREE          ; same through all 16
                        ;  options......
```

This is a simple assembler case statement. We get the value and change it to an offset into the table of procedure addresses. Most CPU's have some form of indirect program jump. That means if I load a registers that points to some place in memory, the CPU will then fetch or retrieve what is in the location and use it as the location to start reading instructions at next.

High level languages all have case statements and C is no different. You hope the resulting underlying asm code produced by the complier will be very similar to the above. I have looked and found that not to be the case, but understand that considerable changes have taken place in the last few years to improve C's output. This is how I might do it in C.

```
#define lport 0x03F8 /* the parallel port address */

main(void)
{
   int   nStatus;

   while(((nResult = inportb(lport)) & 0xF0) != 0)
/* do this as long as something other than 0 is returned */
   switch (nStatus) {
         case 0x10:  /* one car waiting in North lane */
         {  /* one case code goes here... */
         }
         case 0x20: /* case two ......
      ...... through 16 cases ...
   default:
   { /* got garbage data  just loop */ }
   }
   endwhile
```

The last option is using Forth and three ways are possible. Some Forths come with case statements, most do not. Most can do CODE or assembly in line and thus the first example could be used pretty much as is. Normally some basic loop test repeat operation would be chosen and is what I have provided as an example.

```
HEX    / means all numbers are in HEX format
: SIGNAL ( - ) / means no parameters passed
 DO_INIT / call this to setup flags/ports...
 BEGIN    / start loop we never exit
 378 PC@ / read PC port hex 378 putting data on stack
 0F0 AND  / want only top 4 bits
 DUP       / make stack's 1st and 2nd item the same
10 IF            / test top of stack to see if it is 10 hex
= DROP ONE      / test if yes, if so skip top item and call
                / ONE
ELSE DUP 20 IF = DROP TWO / if not 10 see if 20 hex
ELSE DUP 30 IF = DROP THREE / repeat same for all cases
                ........ / will have 15 ELSE's and
THEN THEN .......THEN / 16 THEN's for closure
                / of IF-THEN cases
AGAIN ; / loop back up to BEGIN and repeat test/cases ...
```

You test this program by typing SIGNAL at the OK prompt when running FPC as we did at Forth Day. If you analyze the way FPC does case statements, you will see that the underlying code is the above.

---

I searched my old copies of these CDROMs. Please contact Walnut Creek CDROM for latest prices and current version number. This list is NOT complete, I passed over many other good files.

**Simtel Disk2\cross assemblers**

| | |
|---|---|
| as##_107.zip | Assemblers for many CPU's |
| asem5111.zip | Cross assembler for the MCS-51 family |
| asref.zip | Reference manual for MOTOASMS cross assemers |
| assemblr.zip | Generic 6502/6803/8085 assembler |
| embedpc.zip | Tools & source for embedded PC applications |
| epasm13.zip | Assembler for Intel 8749 & other EPROM chips |
| motoasms.zip | Motorola 6800/01/04/05/09/11 cross assemblers |
| pcmac.zip | Two-pass symbolic cross assembler, w/linker |
| ps##a12.zip | Psuedo Sam assemblers for many CPU's |
| svasm02.zip | Cross assembler for 6502 and 65C02 |
| tasm276.zip | Table-driven cross assembler, for many CPUs |
| uasm.zip | Cross assembler for 8051/6805/Z8, w/'C' src |
| xasm220.zip | Twelve cross assemblers (65xx, 68xx, 80xx) |

**Simtel Disk1\Forth**

| | |
|---|---|
| eforth.zip | Ting's '86 portable eForth, ROMable, w/asm src |
| f83a/b.zip | 8080 Laxen & Perry Forth83,block oriented |

**Simtel Disk2\ASMUTIL**

| | |
|---|---|
| 80x0393.zip | ASM snippet collection from 80XXX FidoNet echo |
| xlt86.zip | 8080 to 8086 ASM translater, w/ASM source |

**Simtel DISK2\PGMUTIL**

| | |
|---|---|
| mide25.zip | Devlp envirn for Arizona Microchip 16Cxx pgmrs |
| pcrob141.zip | Learn programming by writing robot programs |

**Simtel Disk1\Emulators**

| | |
|---|---|
| 22nce142.zip | Z80 CP/M emulator for MS-DOS systems. SYDEX |
| 68em10.zip | 6800 emulator for DOS, includes a realtime O/S |
| mcx11v15.zip | MC68HC11 MicroController multitask eXecutive |
| myz80111.zip | Simeon Cran's Z80 CP/M Z-System emulator |
| s48v10.zip | Full screen simulator for 8048/49/50 micros |
| sim6822c.zip | Motorola 68HC11 uController simulater |
| v2080j88.zip | Run CP/M-80 programs on system with V20 CPU |
| v20boot.zip | Turbo Pascal source code for V20 CP/M emulator |
| z80mu52b.zip | CP/M (Z80 processor) emulator for MS-DOS |
| zsim24.zip | Z80 emulator + CP/M-80 BIOS to run CP/M |

**C USER Group CDROM**

CUG267 - 8085, 2650 & S6 Cross Assemblers
Cross assemblers for Intel 8080 and 8085, Signetics 2650, and SGS S6 micro processors.

CUG276 - Z80 and 6804 cross assemblers
New cross-assemblers (updated CUG267) for Z80 and 6804 processors.

CUG284 - Portable 8080 System (JUG070)
8080 interpreter in C for embedded 8080 & CP/M-80.

CUG284.01-BASIC.ASM source
Palo Alto Tiny BASIC. Stand-alone BASIC ported to 8080.

CUG292 - ASxxxx C Cross Assemblers
Collection of cross assemblers in C for 6800 (6802/6808), 6801 (hd6303), 6804, 6805, 6809, 6811, 8085 (8080), and z80 (hd64180).

**CP/M CDROM**

The CP/M has assemblers, disassemblers, and Pascal and 'C' compilers for the 8080/85 and Z80 CPU's. Most if not all of them run under CP/M on machines like Kaypro's and other older machines.

# Small System Support

by Ronald W. Anderson

Regular Feature

68xx/68xxx Support

C & Assembly

## C Class Notes 7 - Odds and Ends

There are several features of C that we haven't talked about yet. This time there won't be an example program because there are too many things to try to include in one program. Next time we will wind up the Class sessions on C with a rather concise but useful C program. Meanwhile let's cover a number of small points.

### Structures and Unions

We have discussed variables and arrays of variables several times. There are still other more complex ways to store variables than arrays. First there is a multi-dimensioned array that can be thought of as an array of arrays. For example if I were writing a screen editor a screen might consist of an array of lines, while a line is an array of characters. I might describe this arrangement as:

```
char screen_page[25][80];
```

That is, screen_page is an array of 25 arrays of 80 characters each. We can go beyond this, however. Suppose we are writing a program to hold information for an address/phone -book.

```
// defines a new data type person_info
struct person_info {
        char name[40];
        char address1[50];
        char address2[50];
        char city[25];
        char state[3];
        long zip;
}

// declares an array records[] of structures
// of type person_info
person_info records[100];

record[0].name = "Dave Lesnekowiak";
record[0].address1 = "681 Airport Blvd.";
record[0].address2 = "";
record[0].city = "Ann Arbor";
record[0].state = "MI";
record[0].zip = 48108; //Canadian zips are strings
```

We defined a structure data type person_info. We then declared an array of dimension 100 of data type person_info. Then we accessed each element of records[0] to put information into it. With what you now know about C you can see that it would be easy to write a program to prompt a user to enter name and address information for a number of people, perhaps for a personal phone directory.

We could write a program to search the array for a name or city or zipcode. We could write a program to find a record based on a name match and let us modify it, and lastly we could write a program to go through the array and print a nicely formatted name and address list. Of course we might want to add a phone number item to the person_info definition.

Structures like this (called records in Pascal) are used in database systems and in a lot of other applications. If you have a pointer to a structure you use a little different syntax to access a part. *ptr->zip gets you the zipcode part of the sub record if ptr is pointing at the structure in question. Note that the symbol "->" is a compound one made of the two ASCII characters "-" and ">".

### Unions

A union is a collection of different variable types occupying the same space. One is not required very often but they can be very useful. The tutorial program gives a silly example of the use of a union. It shows different data types occupying the same space.

```
union stuff
{
    char c;
    int i;
    float f;
}
```

We need to realize that char occupies one byte, int 2 and float 4. Why you would want to overlap different types in normal programs is beyond my comprehension unless you had defined a large array and wanted to use the same space later for two smaller ones. there are other and better ways to handle that sort of situation in C. The tutorials and books all say that you have to remember what is stored in a union so you don't, for example store a character in this case, and then try to access the integer. I've used a union like this:

```
union mathstuf
{
   unsigned char byteparts[8];
   unsigned int wordparts[4];
   double value;
}
```

Now you declare a variable of type mathstuf and you can store a double there and then by means of the char or int array access various parts of it. I've used this to access the exponent of a double for example to divide the exponent by 2 as a step in writing a square root function. I also used it to transform an 8 byte real variable representation from an old 6809 BASIC into a standard IEEE double representation when I did some programs to transport old 6809 data files to a PC. In these present days of multi-megabyte memory space, I don't see any reason to try to save a few bytes by overlapping variables. Incidentally if you name the structure or union as shown here you have declared a new data type, in this case "mathstuf". They you can declare variables of the type mathstuf. If you omit the name, called a "tag" by the authors of C, you can put a name directly at the end of the declaration (after the ending curly brace) and you have defined a variable that is a union that contains the types of variables that you just defined, but you can only have that one variable. This is true of both struct's and unions.

**Dos Limitations**

While I realize most of the readers of this will be using C on an antique computer of some sort, I thought it would be well to include this information anyway. It does apply to those antique XT machines. MS-DOS has some memory management limitations. Though the processors since the 386 are able to address a large memory space, MS-DOS still has some hangups with that. The maximum size a data structure can occupy is 64K unless you do some tricky things. The C compilers for the PC have a number of "memory models" that let you have larger data areas or larger programs but the limitation still remains in them that you can't have one data item (array) larger than 64K.

You can only get around this limitation by using what is called a "huge" pointer and allocating memory after the program is running, through the use of a memory allocation called farmalloc(). You tell farmalloc() how many bytes of memory you want and it returns a pointer of type void (it can point to any type of data). You cast the pointer into the appropriate type and you can access the large block of memory as though it were a big array using a long for the array index. I've done just that in the editor PAT to get a 200,000 character edit buffer. If farmalloc() can't allocate enough memory it returns a NULL so you can test to see if the allocation was successful.

You can later deallocate a big block of memory or when you exit the program it is done automatically. To use this function you have to include the header file "alloc.h". NOTE: what I've said in the above paragraphs is true of Borland C and their Turbo C. Microsoft may use some other name for the function that allocates large memory blocks.

**Enumerated Data Type**

An enumerated data type may be declared to be used to make a program clearer:
NOTE: This is a new feature in ANSI C. It did not exist in the original K&R version.

```
enum traffic_light { red, yellow, green }

traffic_light signal;  // defines a variable called
                       // signal of type
                       // traffic_light
signal = green;
```

The names red, yellow, and green are assigned the integer values 0, 1, and 2 respectively unless you initialize them with a different integer value. You could accomplish exactly the same thing as follows:

```
#define red 0
#define yellow 1
#define green 2

int signal
signal = green;
```

This seems to me to be an attempt to duplicate the same feature in Pascal. I can't imagine any respectable C programmer using it much. It has the same shortcoming as the Pascal version:

```
enum days { sunday, monday, tuesday, wednesday,
thursday, friday,
saturday }

days day_of_week;

day_of_week = tuesday;

printf("today is ... %d\n",day_of_week);
```

This will nicely print "2". Since day_of_week is an integer for all practical purposes you can't print the day of the week from this setup. You can define a two-dimensional character array of strings containing the names of the days, and use day as an index to select them, for example, but you might just as well use the day number in the first place. The above program segment will print the number 2, not the day name tuesday.

```
char *days[] = {
"Sunday","Monday","Tuesday","Wednesday",
                  "Thursday","Friday","Saturday" };

day_of_week = Tuesday;

printf("today is %s\n",days[day_of_week]);
```

This will print "Tuesday". Note particularly that the array days is an array of pointers to character type. Note also that days[0] points to "Sunday" etc. These are in alignment with the enum declaration above, or the scheme wouldn't work. You can imagine a similar scheme for printing the names of the months given the month number. In this case the first name in the array has to be "dummy" or "error" since we don't start counting months at month zero.

If this usage looks like an "artifact" to make the program read in a more "English like" manner to you, I agree. I would tend to get the month from the month number rather than fooling around with an enumerated type, but suit yourself.

**Assembler**

I have a fan folded card in front of me, labeled MC6809 8-bit microprocessor Reference Card. It came as part of the package with my 6809 processor board from SWTPc in 1976 or so. The card has been shrinking ever since (or could it be my eyesight is getting worse as I age?). At any rate, I am sending my stained and aged copy to TCJ in the hope that it will be made into a "Centerfold" in the future. I think with a little bit of luck it could be copied it directly (hope-fully making it a bit larger rather than shrinking it further).

This reference lists the 6809 instruction set alphabetically by mnemonic from ABA (Add B to A) to TST. Each entry shows all the applicable addressing modes and the hexadecimal op code for each. It also indicates how many bytes the instruction takes and how many clock cycles it takes to execute, as well as which condition code bits it affects in the condition code register (CCR). For example a TSTA instruction "tests" the contents of the A accumulator and the card shows that it sets or clears the Negative and Zero flags according to the data in the accumulator. It unconditionally clears the overflow bit, and it doesn't affect the carry and half carry bits. We haven't talked much about the condition code register yet. The Half carry is used in doing binary coded decimal arithmetic. The reason we haven't gotten into these just yet is that they work more or less automatically. The branch test instructions use them. DECB followed by BNE ... is an example. DECB decrements the contents of B. When it reaches zero the zero bit of the CCR is set. The BNE instruction looks at the zero bit and branches as long as it is a zero (indicating the contents of B resulting from the DECB are not zero). When DECB results in B containing zero, the zero bit is set to 1 and the BNE test fails.

Maybe it is time to list the normal addressing modes of the 6809 and discuss them one by one. All of these modes are different ways of accessing memory except one, called the inherent mode. The inherent mode deals with addressing that is defined by the mnemonic. For example ABX is the instruction to add the contents of the B accumulator to the X register. No memory is involved. Another example is INCA or INCB, causing incrementing of the value in the register. Let's now look at the other ways of accessing memory.

**Direct:**

This mode accesses the Direct Page as defined by the contents of the DP register in the processor. If not changed DP initializes to zero on power up.

```
   LDA $34
```

Loads accumulator A with the contents of memory location $0034 in this case. If the program contains an instruction to set the DP to $11, for example. LDA $34 would access memory location $1134. The direct page was used a lot in small programs and is useful to keep the code size small, but in a larger program, for example an editor or text processor in which there are many variables, most programmers ignore the direct addressing mode and simply consider everything as needing the next mode:

**Extended:**

```
LDA $1234
```

Loads accumulator A with the contents of address $1234. If DP contains $12, the direct addressing instruction LDA $34 would acomplish the same thing using one less byte of memory for the instruction.

**Indexed:**

```
LEAX #$1234
LDA 0,X
```

This again would load A with the contents of memory location $1234. The usefulness of indexed addressing is in the case of needing to access a sequence of bytes in memory. If you had set up an array of bytes in memory you could access element 4 of the array with:

**Indexed with Offset:**

```
LDA 3,X
```

This mode adds the offset (3) to the address contained in X. It would access address $1237 in this case.

We can also use the contents of one of the accumulators as an offset.

```
LDB #3
LDA B,X
```

This has the same effect as LDA 3,X. Of course we can calculate an array offset and have the result in A or B and then use that to load a value. It is a little more flexible than using a constant offset.

We can also bump along a string of addresses in memory using the:

**Post Increment Indexed mode:**

```
LDA ,X+
```

This instruction in the present case would load A with the value in memory location $1234 and then increment the contents of X to $1235. If we have a text string stored in memory, for example, this is a nice way to go through a loop to output it.

```
MESG  FCC /Hi There/,0

      LEAX MESG,PCR  point x at the message string
      BSR PRTT

PRTT  LDA ,X+
      BEQ DONE
      JSR PUTCHR
      BRA PRTT
DONE  RTS
```

Note that if the offset value is zero you can use either 0,X or just plain ,X. Post increment does not work with an offset. That is, you can't use 4,X+.

If you are bumping along loading a 16 bit value you can use:

```
LDD ,X++
```

This increments X by two bytes as you need to do if you are loading 16 bit values into a register. The assembler doesn't take care of this for you. You must use the instruction properly for yourself.

In addition there are instructions for going backwards in memory. The Post Increment instructions use the value in

X and then increment it. the Pre Decrement instructions decrement the value in X and then use it. The pre Decrement syntax is:

```
,-X  and ,–X
```

**Indirect addressing:**

```
POINTR FDB $1234
```

```
LDA [POINTR]
```

This instruction says to use the contents of the variable POINTR as a pointer (in the same sense as the pointers in C that we have been talking about for several issues now). Again in this case the result would be to load A with the contents of memory location $1234.

**Immediate:**

Lastly we must not forget the immediate mode.

```
LDA #$20
```

This puts the value $20 in the A accumulator.

Everything said above about the X register applies equally well to the Y register, the U register or the S register, though the S register (the system stack pointer) is usually left alone by the programmer since it is automatically used as the subroutine return address stack when you do a JSR or BSR instruction. That is, the processor puts the return address on the stack and uses it when it encounters an RTS instruction.

All of the indexed addressing modes can be made indirect too:

LDA 0,X loads A with the contents of memory location pointed at by the contents of the X register.

LDA [0,X] loads A with the contents of the memory location pointed at by the contents of the memory locations pointed at by the contents of the X register.

This is a second level pointer system equivalent to the double pointer in C that we have not yet talked about. You can think of the contents of X as a pointer to a pointer to the value you are going to load in the A accumulator.

Don't fret over this too much. It is not used very often. In fact if you are writing programs in assembler you can simply not use this mode at all. I find that most assembler programmers tend to use a subset of the instruction set (probably too small a subset).

There is one other indexed mode we have used when we did a position independent program, the program counter relative mode.

```
VARIBL FCB 'A
```

```
LDA VARIBL,PCR
```

The label VARBIL is equal to the address to which the program counter is pointing when VARBIL is defined. That value is a constant. The instruction causes the processor to calculate the offset from the instruction LDA VARIBL,PCR to the place where VARBIL is defined, and to use that offset to get to the variable. Because of this, it doesn't matter where the program is loaded in memory, the offset remains the same so the instruction works regardless of the "position" of the program in memory.

Incidentally, though you can use either Y or X as an index register, the instructions that use Y overflowed the original single byte instruction op code set so that the Y register uses a special "post byte" to identify it. Programming with the Y register is therefore less memory efficient than using the X register. Still, there are some nice uses of both X and Y registers, for example, to move the contents of a block of memory from one location to another:

```
   LEAX SOURCE,PCR
   LEAY DEST,PCR
   LDB COUNT
LP LDA ,X+
   STA ,Y+
   DECB
   BNE LP
```

If you are certain that the move is an even number of bytes you can move two at a time:

```
   LEAX SOURCE,PCR
   LEAY DEST,PCR
   LDB COUNT
   ASRB  divide count by 2
LP PSHS B
   LDD ,X++
   STD ,Y++
   PULS B
   DECB
   BNE LP
```

This of course runs considerably faster since the loop must be executed only half as many times as in the first case. Of course there is a little more overhead since the D register consists of the A and B registers joined, we have to hide away the loop count on the stack and get it back in the B register to manipulate it. When you push something on the stack and then later pull it back off you must be careful. This is particularly true if you happen to have the code in a subroutine. Too few PULs will leave a data item on the stack in the way of the return address, and too many PULs will remove part of the return address. You could alternately set up the user stack at some address and use it to save the contents of B temporarily. You can also use a memory location but generally pushes and pulls are faster than writing and reading a memory location.

It is possible to write a general subroutine that can see if the number of bytes is even or odd, and to handle the last (or first) byte differently if the numnber is odd. In a case where you wanted to move a lot of data around in memory as fast as possible, such a subroutine would be desirable and would enhance the execution time of the program considerably.

I recently received a letter from a reader asking for recommendations on books about 6809 assembler programming, "prefferably still in print". That is a big order. I offhand know of none still in print. In fact my library is wiped out. I've either loaned them to friends or left them at work to be borrowed by others. Osbourne published one that was adequate but not very well done in my opinion. I remember the very complex looking photo on the front cover (of the layout of a microprocessor chip) that gave the impression of a very complex subject. The content was there, however. There was a book by John Wakerly published about ten years ago, covering a number of different processors. This one may still be in print. There was and is the Motorola programming guide (a book supplied by Motorola to their customers), and the minimal information that came with various assembler programs for the 6809. There is the "Advanced Programmer's Guide" supplied by SWTPc first at extra cost and later, I think, as a part of FLEX when it was purchased either from SWTPc or from Technical Systems Consultants.

**Miscellaneous**

Well, I guess this time the Assembler part of the column has been reduced to bits and pieces rather than an example program. I am more or less stalled for an example that is more complex than what has preceded and not so complex that I would have to spend a month of evenings to work it out. I have neither the time nor the inclination to get into a long project right now. Our daughter is getting married about 8 weeks from now, probably long before this will be published, but anyway, we are in the midst of finding organists, vocalists, photographers, caterers, supplies such as tablecloths, napkins, cups etc. for the reception. Fortunately our daughter is an organizer and she has a good bit of things already worked out. We have invitations and our half of the guest list pretty well worked out.

**Troubleshooting Hardware**

I enjoyed Bill's Article $10 PC in issue 73. Let me see if I can add some advice for troubleshooting an old computer you might have bought like Bill's PC. First, remove fastening screws so you can remove the cover. Most covers slide off the front of the case. Screws that hold the cover in place are generally run through the back of the case into a flange that is part of the top, so remove the screws along the top edge of the back and generally in the two bottom corners. Don't overdo it and remove the screws that hold the power supply to the back of the case.

Once the case is off you are ready to start troubleshooting. Don't worry about shock hazards contrary to printed instruction and warning labels on some of these units. The XT has all the powerline connections confined to inside the power supply box. Even the switch is mounted on the box. The wires that come out to supply the power to the disk drives and motherboard have only low voltages on them. If the computer is a bit newer and the power switch is mounted on the front, you will find a cable running from the power supply box to the switch. AVOID THE BACK OF THE POWER SWITCH. Most of them have plug on "AMP" lugs and the wires are usually well insulated with plastic sleeving. If in doubt unplug the computer and wrap the back contacts of the switch liberally with plastic electrical tape so there is no bare metal exposed. Now you are safe and you can reconnect the power.

Assuming you have a monitor connected to the computer and that it doesn't have 'words' on it after turning the computer on, the first thing to check is the power supply. Is the fan running? If it is, the 12 volt supply is probably working. If you have a voltmeter you can check the voltage at one of the disk drive power connectors. It is easiest to check one that is not connected to a drive. The two black wires in the middle of the connector are "ground". Connect the negative or common lead of your voltmeter to either of the two black wires. You can simply insert the probe or clip on the end of the voltmeter lead into the connector. You might have to hold it there. Now you should measure +5 volts DC on the RED wire and +12 volts DC at the yellow wire. If either is not there don't panic just yet.

GENERAL RULE: DON'T PLUG OR UNPLUG POWER CONNECTORS OR CIRCUIT BOARDS WITHOUT FIRST TURNING THE POWER OFF.

First disconnect the disk drives including the hard drive, by unpluging the power connector on each. Then power up again and measure voltages. If the voltages are now correct (wait a few minutes for the power supply to recover from its overload condition) you have a bad drive or a weak power supply. Power down and reconnect the floppy drive(s). Power up and if power is still OK the hard drive is the culprit. If power is bad again, unplug the floppy drives one at a time until the fault clears. The last drive disconnected was obviously the culprit.

If power is still bad with drives disconnected, power down again and connect one floppy drive, but now remove the two power connectors from the motherboard. Power up again and look for the two voltages. If they are now good, the problem is in the computer or I/O boards.

Power down again and reconnect the power to the motherboard. The two connectors are supposed to be uniquely coded so they can't be plugged in wrong, but most manufacturers don't bother with this. The connectors go so that the black wires are together and in the middle of the power connector on the motherboard.

Now before powering up again remove all the I/O boards from the motherboard, i.e. the disk controller or controllers (the boards that are connected by ribbon cable to the hard and floppy drives), any serial or parallel port boards or game port boards. Leave only the monitor adaptor board, the one to which the monitor is connected. Now power up. If the supply voltages are now good, one of the boards you have removed is bad. If not, chances are the video board is bad. Power down and remove it and retest the voltages. If power is still bad you have a bad motherboard.

If the system comes up with only the video board and monitor connected, plug in the I/O and disk controller boards one at a time (power down please) and see if the video is still there. If you find a board that causes the video to go away, it is bad and needs to be replaced or repaired.

If you can get the floppy controller back into operation and the video remains, you ought to be able to boot from a bootable floppy. You can test the hard drive if there is one by changing to drive C. After booting from a floppy type C: and see if you get the C> prompt or an error message saying the drive or controller is bad.

You get the idea. The name of the game is divide and conquer.

If you can't get video you have a problem that is harder to track down. Turn power on and see if the hard drive becomes active. The usual sequence is that the hard and floppy drives are each accessed briefly. A hard drive goes through a little self test routine on power up. If it is going to try to boot, there will be a single BEEP from the speaker and then the hard drive will access and shift from track to track. That indicates that the system is booting but you have no video. Either the monitor or video adaptor is bad. It is time to visit a friend and try another of each. CGA boards can be had for $50. So called Multi-IO boards that will still run on the XT bus can be had for $12 or so. They have one parallel port and two serial plus a game port. Don't get the one with an IDE interface on it since that is excess baggage, and an IDE drive won't run on an XT or an early 286 system (without a BIOS upgrade that costs more than the computer is worth).

If you hear multiple beeps from the speaker, it is possible the system is trying to tell you something. The Power On Self Test (POST), if it can't access a monitor or if it can't get far enough in its testing to be able to write to the monitor, will send out a "beep code" that is meaningful. Some codes mean the memory is bad etc. I seem to be running out of space here, so I will try to include the beep codes and the error codes that are shown if the system gets as far as writing to the monitor. The list is long and very helpful.

One more bit of advice. It is OK to mix and match systems, but you need to do something irrational sometimes if you do. You can run an old XT type hard disk controller in an AT (286 machine) as would be the case when upgrading the motherboard. If you do this, you must understand that the old XT controller boards contain a BIOS chip that basically runs them. If you plug the controller and drive into an AT you have to go into the CMOS setup mode (or use the supplied CMOS setup program) to tell the AT that the hard drive is not installed. The XT controller bios runs the drive and it doesn't work if you tell the CMOS that it is there! If you have an AT type controller you have to tell the CMOS setup all about the drive.

If you want to upgrade a 286 machine to run an IDE drive it is not difficult. There are numerous ads in "Computer Shopper" for BIOS upgrades. First, if you can, boot the system and look at and write down the numbers that appear at the bottom of the screen on boot. These numbers identify the BIOS that is installed. The supplier will ask you for them. You may find out that the upgrade will cost $70 or so, and you may want to think about it for a while, but in my experience, I have never seen a really reliable MFM drive. IDE drives simply are newer technology and are much more reliable long term than the old MFM or MFM/RLL drives. Strangely, though old RLL drives are less reliable than their MFM counterparts (sometimes you can use the same drive as either by using the appropriate controller), all IDE drives use RLL encoding of the data since that gives a 50% increase in data capacity. The problem is apparently with the hardware, not the technique.

Should you decide to go with the BIOS upgrade, the salesperson at the other end of the line will try to sell you an upgrade to the keyboard portion of the BIOS as well, for an additional $25 or 30. Considering that even after the expense you will still have an old 286 system, this extra cost is in my mind unjustified. I'd skip it (as I did when I bought an upgrade for my first 286 system). When my ROM set arrived, I plugged in the two (carefully so as not to bend any pins). The computer came up with the new BIOS and it recognized my IDE drive immediately. I did the high level format and the system ran nicely.

Well, that is about it for this time. Next time I promise example programs.

# The TCJ Store

## Regular Items

**Back Issues ........................................................ See page 44**
All Back Issues of TCJ are available.

**TCJ Reference Cards .................................. $3.00 + $1 S+H**
So far, all we have is the Z80 Instruction Set card from Issue #77. These are on heavier stock than the one sent with the issue.

The next two items are Group Purchase Items. TCJ doesn't have the resources to stock these for you, so we have to collect a minimum number of orders before we can provide these.

***GIDE kits ............................................................................ $73**
Tilmann Reh's GIDE board was featured in several issues of TCJ. It is a 'Generic' IDE board for the Z80 that plugs into the Z80 socket (you plug the Z80 back into the GIDE board). This is still an experimenters kit. Sample code and docs including the articles from TCJ are provided, but you have to write your own BIOS routines.

**CP/M CD-ROM ................................................ $25 + $4 S+H**
This is the Walnut Creek CP/M CD-ROM (normally $39.95+S&H) with 19,000 files from Jay Sage, David McGlone, FOG (First Osborne Group), the Beehive BBS, the Enterprise BBS, ftp.demon.uk, and the SimTel20 CP/M collection from the Internet.

## Special Items

We currently have two each of Tilmann Reh's **CPU280** boards and the **IDE** boards that go with them. The CPU280 was featured as the Centerfold in Issue #77 and the IDE interface was in Issue #56. These are bare boards and are not for the faint of heart. They are expensive and the parts are hard to get. But they're fast.

***CPU280 bare board ........................................................ $150**
Comes with docs and utility disk.
***IDE bare board .............................................................. $ 65**
Comes with docs.
***CPU280 & IDE together ................................................ $200**

## CP/M Kaypro Catalog

| | |
|---|---|
| Upgrades | |
| Advent TurboRom | |
| K4-83 | $35.00 |
| K10-83 | $35.00 |
| Kx-84 | $35.00 |
| MicroCornucopia Roms | |
| Pro 8 | $35.00 |
| 884 Max | $35.00 |
| 884 Max (Lo) | $35.00 |
| Character ROM | $35.00 |
| Add-ons | |
| HandyMan | $75.00 |
| Disk Drives | |
| Dual Density TEAC FD-55BV | $15.00 |
| Quad Density TEAC FD-55FR | $15.00 |
| Pair | $25.00 |
| ST-225 20 MByte MFM HD | ?? |
| Disk Controllers | |
| WD-1002-05 HDO | $75.00 |
| Tech Data | |
| Kaypro Technical Manual | $25.00 |
| *Microcornucopia* Schematics with Theory of Operation | |
| K-II/4 83 | $15.00 |
| K-10/83 | $15.00 |
| All-84 | $15.00 |
| Any two | $25.00 |
| All three | $30.00 |
| Software | |
| Advent Harddisk Formatter | $25.00 |
| TurboRom Applications Patches | $10.00 |
| TurboRom Developers Diskette | $10.00 |
| Kaypro 10/83 Tinker Kit | $10.00 |
| Kaypro 2,4/84 Tinker Kit | $10.00 |
| Kaypro CP/M 2.2H Autoload set 8 diskettes for K-10/84 | $40.00 |
| Other Stuff | |
| Keyboards | $30.00 |
| Video - CRT and board | $40.00 |
| Kaypro Carrying bags | $75.00 |
| Kaypro machines | |
| K-II, K-2, K-4, K-10 available in various condition. | |

TCJ can accept credit card orders by phone, fax, or mail or you can place an order by sending a check to:

**The Computer Journal**
PO Box 3900, Citrus Heights
CA 95611-3900
Phone: 800-424-8825 or 916-722-4970
Fax 916-722-7480 / BBS 916-722-5799

Include your shipping address with your check, and your Internet address if you have one. For more info, contact TCJ via E-mail at tcj@psyber.com

* In Europe and particularly Germany, contact Tilmann Reh for a current price and shipping. His email address is: "TILMANN.REH@HRZ.uni-siegen.d400.de"

His postal address is:
Tilmann Reh
Autometer GmbH
Kaenerbergstrasse 4
57076 Siegen (optional "-Weidenau")
GERMANY

Regular Feature
Contact Listing

# SUPPORT GROUPS FOR THE CLASSICS

## *TCJ* STAFF CONTACTS

**TCJ Editor:** Dave Baldwin, (916)722-4970, FAX (916)722-7480 or TCJ BBS (916) 722-5799 (use "computer", "journal", pswd "subscriber" as log on), Internet tcj@psyber.com, dibald@netcom.com . Also CompuServe 70403,2444.

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

**32Bit Support:** Rick Rodman, 1150 Kettle Pond Lane, Great Falls, VA 22066-1614. Real Computing BBS or Fax: +1-703-759-1169. E-mail: ricker@erols.com

**Kaypro Support:** Charles Stafford, on the road somewhere. Email: CIS 73664,2470 (73664.2470@compuserve.com). TCJ has taken over Chuck's Kaypro parts and upgrade business.

**S-100 Support:** Herb Johnson, 59 Main Blvd. Ewing, NJ 08618 (609)771-1503. Also sells used S-100 boards and systems. E-mail: hjohnson@pluto.njcc.com.

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

**Z-System Support:** Jay Sage,1435 Centre St. Newton Centre, MA 02159-2469, (617)965-3552, BBS: (617)965-7046; E-mail: Sage@ll.mit.edu. Also sells Z-System software.

## REGULAR CONTRIBUTORS

**Brad Rodriguez**, Box 77, McMaster Univ., 1280 Main St. West, Hamilton, ONT, L8S 1C0, Canada, E-mail: bj@headwaters.com..

**Frank Sergeant**, 809 W. San Antonio St., San Marcos, TX 78666, E-mail: pygmy@pobox.com.

**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.

## USER GROUPS

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

**SMUG**, Sacramento Microcomputer Users Group, has disbanded after all these years.

**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).

**NATGUG**, the National TRS-80 Users Group, Roger Storrs, Oakfield Lodge, Ram Hill, Coalpit Heath, Bristol, BS17 2TY, UK. Tel: +44 (0)1454 772920.

**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/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, 17-885 Berwick Rd. Qualicum Beach, B.C., Canada V9K 1N7, (604)752-1984. dcobley@qb.island.net

**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.

**The San Diego Computer Society** (SDCS) is a broad spectrum organization that covers interests in diverse areas of software and hardware. It is an umbrella organization to various Special Interest Groups (SIGs). Voice information recordings are available at 619-549-3787.

The **Dina-SIG** part of SDCS is primarily for Z-80 based computers from Altair to Zorba. The SIG sponsored BBS - the Elephant's Graveyard (619-571-0402) - is open to all callers who are interested in Z-80 and CP/M related machines and software. Contact Don Maslin, head of the Dina-SIG and the sysop of the BBS at 619-454-7392. Email: donm@cts.com.

**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, local chapters.

**The Pacific Northwest Heath Users Group**, contact Jim Moore, 1554 - 16th Avenue East, Seattle, WA 98112-2807. Email: be483@scn.org.

**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*** - has ceased publication.

***The Analytical Engine***, by the Computer History Association of California, 3375 Alma, Suite 263, Palo Alto, CA 94306-3518. An ASCII text file distributed by Internet, issue #1 was July 1993. Home page: http://www.chac.org/chac/ E-mail: engine@chac.org

***Z-100 LifeLine***, Steven W. Vagts, 2409 Riddick Rd. Elizabeth City, NC 27909, (919)338-8302. Publication for Z-100 (an 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, 6889 Crest Avenue, Riverside, CA 92503-1162. $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. Email:IQLR@nccnet.com.

***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. Emil: fdavis@holli.com.

## 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, (541)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. Email:??

**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. SK*DOS 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.

**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. Web page: http://www.micro-solutions.com.

**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.

# 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
- 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
- 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: 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: Root Locus Analysis & Feedback Loops
- The C Column: 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
- 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
- 32000 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:**
- copies still 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: OS 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.
- A Short Course in Source Code Generation: Disassembling 8088 software to produce modifiable asm 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 experimenter hardware, 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.
- ZSDOS: Anatomy of an Operating System: Part 1.

**Issue Number 38:**
- C Math: Dollars and Cents With C.
- Advanced CP/M: Batch Processing and a New ZEX.
- C Pointers, Arrays & Structures Made Easier: Part 2, Arrays.
- Z-System Corner: Shells and ZEX, 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 HP 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 5.0xThe 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.
- 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 Character Attributes.
- Forth Column: Lists and object oriented Forth.
- The Z-System Corner: Genie, BDS Z and Z-System Fundamentals.
- 68705 Embedded Controller Application: A single-chip microcontroller application.
- Advanced CP/M: PluPerfect Writer and using BDS C with REL files.

**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: Routines for the IBM PC, and the Turbo C 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.

**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 emulator for CP/M.
- Advanced CP/M: ZMATE and using lookup and dispatch for passing parameters.
- Forth Column: Handling Strings.
- Z-System Corner: MEX and telecommunications.

**Issue Number 45:**
- Embedded Systems for the Tenderfoot: Getting started with the 8031.
- 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.
- 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 and using the Z80 CTC.

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

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

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

**Issue Number 50:**
- Offload a System CPU with the Z181
- Floppy Disk Alignment w/RTXEB, Pt. 2
- Motor Control with the F68HC11
- Modula-2 and the Command Line
- 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
- Home Heating & Lighting, Pt. 3
- The CPU280, A High Performance SBC

**Issue Number 53:**
- The CPU280
- Local Area Networks
- An Arbitrary Waveform Generator
- Zed Fest '91
- Getting Started in Assembly Language
- The NZCOM IOP

**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 Virtual BIOS
- The Bumbling Mathmatician
- YASMEM

**Issue Number 56:**
- TCJ - The Next Ten Years
- Input Expansion for 8031
- Z-Sys Corner - Zed-Fest
- + Connecting IDE Drives to 8-Bit Systems
- 8 Queens in Forth
- Real Computing - Linux, BSD 386, Minix
- Kaypro-84 Direct File Transfers
- Analog Signal Generation

**Issue Number 57:**
- Z-Sys Corner - Language Independence
- DR. S-100 - the start
- Home Automation with X10
- File Transfer Protocols - Info
- MDISK at 8 MHZ. - Ampro Update
- Shell Sort in Forth
- Introduction to Forth
- Z AT Last! - ZCPR on a PC? MYZ80!

**Issue Number 58:**
- Z-Sys Corner - Language Independence II
- Real Computing - Minix, UZI, and GNU
- Affordable Development Tools
- DR. S-100 - Tips and info
- Mr. Kaypro - Move the Reset
- Computing Timer Values - Monostables, C
- Multitasking Forth

**Issue Number 59:**
- Z-Sys Corner - ZMATE MACRO usage
- Moving Forth - Part 1
- Center Fold - IMSAI MPU-A
- Developing Turnkey Forth Applications
- Mr. Kaypro - Versions of Kaypros
- DR. S-100 - Vendors

**Issue Number 60:**
- Next Ten Years - Part II
- Moving Forth Part II
- Center Fold - IMSAI CPA
- Four for Forth - Forth CPU's
- Debugging Forth
- Z-Sys Corner - 8 years of Z-System
- Mr. Kaypro - Turning a Kaypro II into a IV
- DR. S-100 - Letters

**Issue Number 61:**
- Z-Sys Corner - Automating GEnie Mail
- Multiprocessing / 6809 part I
- Center Fold - XEROX 820
- QC Using the Commodore 64
- Real Computing - JPEG, WORM, archivers
- Support Groups for Classics
- Operating Systems - CP/M
- Mr. Kaypro - 5 MHz Upgrade

**Issue Number 62:**
- SCSI EPROM Programmer
- Center Fold - XEROX 820
- DR S-100 - Exploring the S-100 Bus
- Moving Forth part III
- Programming the 6526 CIA
- Reminiscing and Musings - 10th Year
- Modem Scripts

**Issue Number 63:**
- Z-Sys Corner - Failsafe Scripts in 4DOS
- SCSI EPROM Programmer - part II
- Center Fold - XEROX 820
- DR S-100 - Disk Drives and BIOS code
- Multiprocessing Part II
- 6809 Operating Systems
- + IDE Drives Part II

**Issue Number 64:**
- Z-Sys Corner - Failsafe Scripts in 4DOS II
- Small-C - Review and comment
- Center Fold - last XEROX 820
- DR S-100 - Disk Drives and BIOS - part II
- Moving Forth Part IV
- Small Systems - 6800/6809 History
- Mr. Kaypro - Sign on and Clock Upgrade
- + IDE Drives - Part III

**Issue Number 65:**
- Small System Support - 68xx Serial Comm
- Sinclair ZX81 - Letters and Books
- Center Fold - ZX80/81
- DR S-100 - Christmas letters
- Real Computing - Linux and Linking
- European Beat - AMSTRAD in Europe
- PC/XT Corner - Day-Old Computing
- Little Circuits - Reset Circuits
- Levels of Forth - Selecting a Language

**Issue Number 66:**
- Z-System Corner - Failsafe Scripts in 4DOS
- Real Computing - TCP/IP and OSI
- Small System Support - 'C' and 68xx
- Center Fold - Advent Decoder Board
- DR S-100 - Spring Letters
- + Connecting IDE Drives (IDE part IV)
- PC/XT Corner - Day-Old Computing
- Little Circuits - Battery Backup Circuits
- Multiprocessing Part III

**Issue Number 67:**
- European Beat - more AMSTRAD history
- Small System Support - 6800/09 programs
- Center Fold - SS-50/SS-30
- DR S-100 - Trenton/Z-Fest & letters
- Serial Kaypro Interrupts in Forth
- Real Computing - Tiny-TCP and WIN
- Little Circuits - Wire and Cable
- Moving Forth Part 5

**Issue Number 68:**
- Small System Support - Languages
- Center Fold - Pertec/Mits 4PIO
- Z-System Corner II - Intro CP/M and Z-Sys
- PC/XT Corner - A bit of everything - Part I
- Little Circuits - CMOS and RC's
- Multiprocessing Forth Part 4
- Mr. Kaypro - Notes, Repairs, and Macros

**Issue Number 69:**
- Small System Support - 6809 ASM, Flex
- Center Fold - S-100 IDE
- Z-System Corner II - Intro, part 2
- Real Computing - Tiny-TCP
- PC/XT Corner - Stepper Motors and Forth
- DR. S-100 - Mail Bag
- Moving Forth Part 6
- Mr. Kaypro - Advent Decoder Construction

**Issue Number 70:**
- Small System Support - 6809 ASM
- Center Fold - Jupiter ACE
- Z-System Corner II - Intro part 3
- PC/XT Corner - Stepper Motors & Forth
- DR. S-100 - Mail Bag
- Multiprocessing Part 5
- European Beat - 8-bit idiot and AMSTRAD

**Issue Number 71:**
- Computing Hero of 1994 - David Jaffe
- Small System Support - 6809 ASM
- Center Fold - Hayes 80-103A S-100 modem
- Power Supply Basics
- PC/XT Corner - Stepper Motors
- + Connecting IDE Drives (5) - GIDE Preview
- DR. S-100 - Generic IDE and CompuPro
- Moving Forth Part 7
- Mr. Kaypro - ROM options
- 8048 Emulator Part 1

**Issue Number 72:**
- Beginning PLD - good and bad
- Small System Support - 'C' and ASM
- Center Fold - Rockwell R65F11
- Playing With Micros - 5 to learn with
- Real Computing - Languages
- Small Tools Part 1 - Forth, 68HC11
- DR. S-100 - CompuPro 8080/8086
- Moving Forth Part 7.5
- 8048 Emulator Part 2

**Issue Number 73:**
- $10 XT - what you can get at a swap meet
- Small System Support - 'C' and ASM
- Center Fold - 640K XT
- + IDE Part 6
- Real Computing - Linux
- Small Tools Part II - New Micros F68HC11
- DR. S-100 - Trenton and Letters
- PC/XT Corner - software quandaries
- 8048 Emulator Part 3

**Issue Number 74:**
- Antique or Junk - How to judge your system
- Small System Support - 'C' and ASM
- Center Fold - S-100 Power Supply
- Real Computing - Linux and Minix
- AMSTRAD PCW Now
- DR. S-100 - Mailbag
- Mr. Kaypro - Adding Composite Monitors
- Palmtech CPUZ180 - Review
- Disk I/O in Forth
- Moving Forth part 8

**Issue Number 75:**
- The European Beat - East German Z80
- Small System Support - 'C' and ASM
- Center Fold - Standard Bus and I/O
- Moving Forth part 8
- Real Computing - Rick moved
- Embedded Control Using the STD Bus
- DR. S-100 - Mailbag
- EPROM Simulator
- High-Speed Serial I/O for the Applicard
- Disk I/O in Forth, Pt. 2
- T9600 Source Code (Small Tools)

**Issue Number 76:**
- Real Computing - Minix and more
- PC/XT Corner - Bank Switching/Supercharge
- The European Beat - 10 years for user group
- Alternatives to the XT
- DR. S-100 - GIDE and the Jade Bus Probe
- Center Fold - JADE Bus Probe
- PC Time Clock - Improving Accuracy
- PC Security System - Home Security
- Small System Support - 'C' and ASM
- Floppy Disk Problems - design problems

**Issue Number 77:**
- Mr. Kaypro - External Video
- Hands-on with PLD's
- Center Fold - CPU280
- The First TRS-80
- Program This! - the Z80 SIO
- Small System Support - Prime Numbers in C

**Issue Number 78:**
- 6502 DIY Board
- Program This! - 8051 Startup Code
- Center Fold - AMR 80552
- Simplex III - Homebuilt TTL CPU
- Real Computing - Small C, C-64, Win95
- Small System Support - C and Assembler

| | 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)** | **Shipping + Handling for each issue ordered** | | | | |
| Bound Volumes $20.00 ea | x$3.00 | x$3.50 | x$6.50 | x$4.00 | x$17.00 |
| #32 thru #43 are $3.00 ea. | x$1.00 | x$1.00 | x$1.25 | x$1.50 | x$2.50 |
| #44 and up are $4.00ea. | x$1.25 | x$1.25 | x$1.75 | x$2.00 | x$3.50 |

Items: ______________________________

______________________________

______________________ Back Issues Total ________

-10% for 10 or more or with subscription,

-15% for 10+ with subscription - discount ________

California Residents add 7.25% Sales TAX ________

Shipping Total ________

Subscription Total ________

Total Enclosed ________

Name: ______________________________

Address: ______________________________

______________________________

______________________________

______________________________

Email: ______________________________

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 3900, Citrus Heights, CA 95611-3900**
**Phone (916) 722-4970 / Fax (916) 722-7480**

Regular Feature

Editorial Comment

Busy Summer

# The Computer Corner

By Bill Kibler

Well summer is gone and with it all chances of catching up with projects. I must admit my time off for a trip to Alaska was enjoyable and I can recommend that people should see Alaska if possible. Trying to make up for the time lost and all the problems that happened after I got back may take some time to recover from however.

**System Overload**

I had promised to do some programming on the side this summer, and with that came the need to use Win95. I have moved from using Win 3.1 to WinNT at work and find that a rather nice change. I have not exhaustively tested DOS programs, but then the main ones I use all do work. That has been a very big surprise. I thought most would fail or do something strange, but it seems I was wrong.

The Win95 actually installed very easily and I was surprised to find all my PCDOS 7.0 drivers were moved and used just as they had been before the change. NT however was installed by someone else at work and so I was unable to see what happens when you upgrade. I think WinNT blows all the old items away which causes many problems if there is not an NT driver for one of your devices.

Win95 solved the driver problem, by basically using any previous drivers or better put, by just running a DOS under the win program so it still can use the DOS drivers without problem. I'm still trying to decide if there really is a good reason to do it if your happy with Win 3.1 or something else. A friend who I told not to upgrade feels he would not do it after what has happened or better put, what he had to relearn and reload. Seems he had lots of programs with 95 that didn't work right after the change.

What I find interesting is how Microsoft is still hiding from the public how many bugs and problems have been quietly fixed with upgrades. With NT you see the "build" number when it boots. This is sort of their way of telling you which minor bug fix version you have. My guess is that there is some similar way to determine where in the bug fix loop your version of 95 is, but at present I really don't care since I know there isn't much you can do to try and keep up with the changes. I have heard there are plenty of revisions or upgrade packs for 95.

**Older Sales**

I was told by a friend that he saw someone selling a IMSAI with front panel for $4000, I think on the Internet. Everyone was giving the person a bad time about the price, only to have him tell them it was sold already. I think there are plenty of our readers who would agree with both sides of the issue on the price. By that I mean, some will think it is too much money, while others and myself say it is only the beginning. I expect to see similar stories over the next few years until the collecting side gets well established.

What will it take to get established? I am sure a big sale at Sotherby's would do it. An IMSAI or such going for say a $100,000 would certainly make some headlines and send people scrambling. Do I really think it will happen, yes - but how long is the only question. It seems to me that collectibles sort of grow slowly in price till they are rather scarce. Once the availability of the items drops from thousands to hundreds or reach rare status, then and only then do people start bidding up the price.

Are IMSAI's rare, yes and no. The yes side is from people hording their older systems and also from giving up on trying to sell them. That comes from so many people being interested in the latest hardware gimmick at swap meets, that selling something outside the normal is pretty much a waste of time. I haven't seen a S100 card for sale at a swap meet for years now. It is getting to be that most buyers have no idea of what the older systems were or if they have any value at all. For those of us who do know an IMSAI from a PC, a great find ing if they shouldn't have sold.

The NO side of me says that IMSAI are not rare, people are just waiting. I know plenty of people who have numerous collectible systems stashed away. There is not a shortage, just people waiting for higher prices. If all goes well, I see a later time when there will be once or twice a year collectible computer swap meets. At first probably based on invitation to attend, just to make sure only vendors of older systems are there. If PC sellers are allowed, forget it, it will not work.

I feel it would probably be possible to do one in Silicon Valley now and easily two or three hundred sellers could be found to come to it. You would get both the honest buyers and sellers as well as the media. Once the big TV shows stopped by and it hit national news, boom were in business and the 100K IMSAI would be but a few years down the road.

The key is finding real collectible vendors willing to talk and display their systems. No junk piles, no boxes of bad boards, no systems without books and software. Of course you would need to let in the few people left who also support the old systems with magazines, software, and hardware fixes. It can't be some parking lot sale or some business back lot. You got to rent a real convention hall with guards, drinks, and vendor booths all which means some money up front. But be prepared to talk more than sell the first few years!

## Hardware

Prices of new hardware keep dropping and for those using the PC platform, power and more has become very cheap. When I visited my cousin in Alaska, we talked Internet, and I offered to build them a system for $300. I could have bought several used 386 systems already setup for the Internet from local used dealers. I had however many pieces to make a system and at first was only going to buy a new case for them.

A local dealer however had a few sales going and 300 became $600 for a 586 with a 16Meg SIMM and a 1 gig drive. I ran some tests and found the system very fast and was very surprised at the VLB video card. It was faster than my PCI bus video card. It seems the VLB is better tuned for video, while the PCI is a faster overall bus. The cost was $40 for the mother board, $60 for the 586, and $110 for the ram. I had planned on buying wholesale, but the 1 gig drives where going for less than my wholesale quotes. The drive got me in the store and then I found the other items and bought quick before they were gone.

When looking at PC hardware items, it can seem that the PC market has gone crazy, but so have others. The embedded hardware sales literature keeps coming faster than I can read it. The people behind the PIC chips can't make new versions fast enough. Motorola is spinning off variations of their 6805 and 6811's like mad. I believe there is now a 80151 that fits in between the old standby 8051 and the new 80251 high end. Numerous Japanese vendors are making big inroads with their own chips as well.

I think the only other area that has been getting more news than all the hardware vendors is the battle brewing over cheap Internet boxes. Several vendors are planning on having systems ready to sell by THIS Christmas. Cable companies are trying to get hardware in place and software on line to meet the hoped for demand. I think the demand will be there, but plenty of behind the scenes action has to happen first.

There is more than enough people who feel that a melt down will happen if even half the people start getting on line that could now. What would happen if all the Christmas buyers really did buy and tried to get on is any bodies guess. My guess for this year is forget being on the Internet for all of January. Ask any BBS owner and they will tell you how the next few weeks after Christmas is a nightmare. All these first time users with their new modems learning how to use them on your BBS. Now take that concept and give them the Internet. Just plan on giving it to them, since you will not be able to use the Internet till they burn-out or give up.

## Projects

The project queue is very deep and got thrown off track over the summer. I had a few great plans to sort of catch up, but we lost three Llamas this summer to heat, two of my wife's relatives were buried, all in 45 days. Busy is an understatement for me. The hot weather is still lingering longer than normal and since my hardware shop has no cooling, I have been forced out of it for a little longer. What will I be doing?

Number one on my list to do is hooking up my pile of S100 systems. I stacked them up in my shop next to the work table. I start with a hard drive, 3 tiers of 8 inch drives, two S-100 cabinets, and a 5 inch hard drive for a total of four feet of systems. I am thinking of using a Z-100 system as a terminal and disk interface. I have other ideas in mind for hooking them altogether, but it is still too early to know how that will actually be done.

The idea is to be able to test some older hardware and get BIOS listings off of the many disks I have. The CP/M and *TCJ* CDROM is still in the works, just dragging behind a little. I think Dave is getting all the items working so the hard disk with the current CDROM software can be on his BBS. My plans are to add to that drive until we get a CDROM full. We will review the contents periodically and keep you posted. If you can add BIOS code or embedded software please help out. Remember however that DRI products are copyrighted and still not public domain. BIOS code in many cases was not protected, while some was and may still be even if the company has gone out of business. When a companies assets were bought by another, all copyrights normally went as well. My hopes are to put software on the BBS and if nobody complains after some reasonable amount of time, we can assume that sets a precedence, and we can then include it on the CDROM. If a rightful owner does complain, off it comes with our apologies and the CDROM will then indicate who does have the rights and where to mail letters if you need support.

I suppose one way out would be to make sure that all text referring to copyright and ownership has been removed before we get it, but then that would not be legal. We do try to be legal at *TCJ* even if the companies make it very hard by refusing to allow releasing information that is needed to keep a system running 10 years after the company went of business.

I had numerous plans in the wings and will try getting to them for later issues, but for now, keep hacking and let me know how your fun is going!

Bill.

# TCJ CLASSIFIED - Items Wanted and For Sale

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 | $150 | $90 |
| 1/2 Page | $80 | $60 |
| 1/3 Page | $60 | $45 |
| 1/4 Page | $50 | $40 |
| Market Place | $30 | $120/yr |

Send your items to:
**The Computer Journal**
P.O. Box 3900
Citrus Heights, CA 95611-3900

**CHICAGO TI99/4a** Classic Computer Faire (14th annual) will be held November 9th, 1996 at the Evanston Public Library in Evanston, Illinois. For more information please contact Hal Shanafield, (847) 864-8644 or Chicago TIUG, P.O. Box 7009, Evanston, IL 60204-7009. Specifically for owners of TI99/4a and Geneve 9640 computers!

**DIBs**
Electronic Design

Dave Baldwin

Microprocessor, Digital, and Analog circuit design.
PC layout and more.

Voice (916) 722-3877
Fax (916) 722-7480
BBS (916) 722-5799

**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 $14, or try an issue for $3. HCS, 3649 Herschel St., Jacksonville, FL 32205.

Start your own technical venture! Don Lancaster's newly updated **INCREDIBLE SECRET MONEY MACHINE II** tells how. We now have autographed copies of the Guru's underground classic for $21.50. Synergetics Press, Box 809-J, Thatcher AZ, 85552.

**THE CASE AGAINST PATENTS** Throughly tested and proven alternatives that work in the real world. $33.50. Synergetics Press, Box 809-J, Thatcher AZ, 85552.

**Wanted: Form filling software** for the KayPro CP/M computer. Trying to find "Formation" by PBT software once of Grand Rapids, MI, or "StanForm" by MAP, Micro-Art Programmers. Other software capable of filling out preprinted forms considered. Help give a KayPro meaningful work! Please reply to Stephen Stone -Tel. (805)569-8329 or stephen@silcom.com

**Wanted: Intel SDK-85** documentation. This is a single board design kit with the 8085 CPU, includes a hex keypad and 7 segment LED readout. I have several of these units and would consider trading for interesting older computers. Ron Wintriss, 100 Highland Ave., Lisbon, NH 03585.

**TRS-80 - MODELS I, 2, III, IV, 12, 16, POCKET COMPUTER, AND COCO:** Software, hardware, internal and external disk drives (360K/720K), hard drive's (both complete and bubles), replacement motherboards, floppy drive controllers, video boards, RS-232 boards, keyboards, and more. Send 4x10 SASE for list.
Pete Bumgardner, Rt. 4 Box 36-H, Fort Payne, AL 35967-9408, (205)845-0581.

**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!

**FOR SALE: THE FORTH ARCHIVE** from taygeta.com on CDROM is available from Mountain View Press, Rt 2 Box 429, La Honda, CA. 94020 Ph: 415-747-0760
ghaydon@forsythe.stanford.edu.

**FOR SALE: Kaypro 2,** appears to be in good condition, located in Indiana. Call Bob Finch, 317-564-4226.

***Kibler Electronics***

**Hardware Design &**
**Software Programming**

8051, 6805, Z80, 68000, x86
PLC Support and
Documentation

**Bill Kibler**
**P.O. Box 535**
**Lincoln, CA 95648-0535**
**(916) 645-1670**
*e-mail: kibler@psyber.com*
*http://www.psyber.com/~kibler*

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

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

The Z-Letter
Lambda Software Publishing
149 West Hilliard Lane
Eugene, OR 97404-3057
(541) 688-3563

---

**Advent Kaypro Upgrades**

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

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

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

---

***TCJ* MARKET PLACE**
Advertising for small business

| | |
|---|---|
| First Insertion: | $30 |
| Reinsertion: | $25 |
| Full Six issues | $120 |

Rates include typesetting.
Payment must accompany order. VISA, MasterCard, Diner's Club, Carte Blanche accepted. Checks, money orders must be US funds. Resetting of ad consitutes a new advertisement at first time insertion rates. Mail ad or contact
***The Computer Journal***
**P.O. Box 3900**
**Citrus Heights, CA 95611-3900**
**(916) 722-4970**
**Fax (916) 722-7480**

---

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

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

---

**VINTAGE COMPUTERS**
IBM Compatibles
Tested - Used Parts for
PC/XT AT PS/2
Working systems from $50
All parts including
cases monitors floppies
hard drives MFM RLL IDE
Technical Specs
Send 5x7 SASE to:
**Vintage Computers**
Paul Lawson
1673 Litchfield Turnpike
Woodbridge, CT 06525
or call for a faxed list
203-389-0104

---

**MORE POWER!**
**68HC11, 80C51 & 80C166**

☑ More Microcontrollers.
☑ Faster Hardware.
☑ Faster Software.
☑ More Productive.
☑ More Tools and Utilities.

Low cost SBC's from $84. Get it done today! Not next month.
**For brochure or applications:**
**AM Research**
**P.O. Box 43**
**Loomis, CA 95650-9701**
**1(800) 949-8051**
**http://www.AMResearch.com**

---

**VERSATILE 80C32 AND 68HC11 SINGLE BOARD COMPUTERS**

The DC8032-1 includes the following:
- 11.059 MHz 80C32 processor.
- 32K of EPROM.
- 4 different memory maps.
- Extended BASIC-52 with 28 additional commands.

The DC6811-1 includes the following:
- 8MHz MC68HC11 processor.
- 32K of EPROM jumper selectable as 2 16K Eproms.
- MBASIC11 with custom analog and digital I/O commands.

All units include the following standard features:
- 32K of battery-backed RAM.
- Real time clock.
- 8-channel/8-bit A/D.
- Centronics parallel printer port.
- 24-bits of digital I/O.
- Watch dog timer.
- 4 x 6 inch board size.
- Operates on a single 9 to 12 volt DC power supply.
- 40-pin expansion connector.
- RS-232 port.
- 30 day money back guarantee.
- One year parts and labor warranty.

All unit come with a 9 volt DC wall cube, serial cable, users manual, and DC_TERM terminal software. A utility disk of shareware and freeware is also included at no charge.

| | |
|---|---|
| D. C. MICROS<br>1843 Sumner Ct.<br>Las Cruces, NM 88001<br>Ph. (505) 524-4029 | $140.00 kit or assembled and tested. Add $5.00 shipping and handling plus $5.00 for COD |

---

**THE FORTH SOURCE**

**Hardware & Software**

**MOUNTAIN VIEW PRESS**

**Glen B. Haydon, M.D.**
**Route 2 Box 429**
**La Honda, CA 94020**

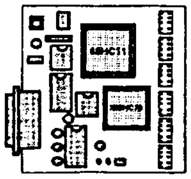
**(415) 747-0760**

http://www.taygeta.com/jfar/mvp.html

---

***$79.95*** **68HC11 Single Board Computer SBC-8K**

**8K EEPROM for More Program Space!**
**SER-8C, Optional 8K Serial EEPROM $10**



- Small Size, 3.3"x3.6"
- Low Power, <60 ma
- 8192 Bytes EEPROM
- 256 Bytes RAM
- DB-9 RS-232
- 24-TTL I/O Bits
- 8-A/D Inputs
- Power Reset Circuit
- 8 Mhz Clock
- Log Data with SER-8C

**A Complete 68HC11 Development System.**
**New "CodeLoad+ 2.0" and Sample Programs.**
**No EPROMs or EPROM Programmers!**
**500 Pages of Manuals, 3.5" Utility Disk.**

**LDG Electronics**
1445 Parran Road
St. Leonard, MD 20685
Voice / Fax
410-586-2177

TCJ The Computer Journal
Post Office Box 3900
Citrus Heights, CA 95611-3900
United States

Phone: 1-800-424-8825 or (916) 722-4970
Fax (916) 722-7480 / BBS (916) 722-5799

ADDRESS CORRECTION REQUESTED
FORWARDING AND RETURN POSTAGE
GUARANTEED

**Check Out the NEW return
Address and phone numbers!
BBS, FAX, and WEB SITE!**

**Advanced Systems
and Modems This Issue!**

**Check label for expiration issue
and renew early.**