Reader to Reader .................................. Letters and messages
Real Computing ............................... Small-C, C-64, Win95
The European Beat ............................ GIDE and KC-85
6502 ............................................... DIY board
XT Corner ...................................... Forth, Internet, and GUI
Simplex III ..................................... Homebuilt TTL CPU
Center Fold .................................... AMR 80552
Dr. S-100 ........................................ Messages and Mail
Small System Support ........................ C and Assembler
Program This! ................................ 8051 Startup Code
Computer Corner ...................... PLD's, linux and embedded
**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-FOHTH  Fax: 510-535-1295

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

---

**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

---

**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

---

**AWESOME ZBO COMPUTER**

Perfect for Project or Product

INCLUDES

- KEYBOARD (17"x15.5x2.5")
- Durable Plastic, 66 Keys
- 2 Connectors for Joysticks
- ENCLOSURE (19"x11x4")
- 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!

UNIT PRICE
1 UNIT... $125.00

KART COMPUTERS
P.O. Box 144
Mullbury, WA 01527
(308) 735-9778
Editor's Column .............................................. 2
And in this issue...

Reader to Reader ........................................... 3

Real Computing ............................................. 5
Small-C, C-64 emulators, and Win95 Review
By Rick Rodman.

The European Beat ....................................... 8
Putting GIDE to work on the KC-85.
By Helmut Jungkunz.

CP/M 86 Enters the 90's ................................ 9
A Patch for CP/M 86
By Kirk Lawrence

6502 .......................................................... 10
DIY board
By Doug Beattie

XT Corner .................................................... 15
Large Data in Forth, the Internet, and GUI stuff.
Frank Sergeant

Simplex III .................................................. 21
Homebuilt microcoded TTL processor, Part 1.
By Dave Brooks.

Center Fold ............................................... 23
amr552LC and 8031

Dr. S-100 ...................................................... 27
Messages and Mail
By Herb Johnson.

Small System Support .................................. 31
C and Assembler
By Ronald W. Anderson.

Program This! .............................................. 37
8051 start-up code with interrupts.
By Dave Baldwin.

TCJ Store .................................................. 41
Things for sale from TCJ.

Support Groups for the Classics ..................... 42
Back Issues ............................................... 44

The Computer Corner .................................... 46
PLD's, Linux, and embedded.
By Bill Kibler.
About TCJ:

The Computer Journal really is a subscriber supported magazine. In a way, it's kind of a large, international computer club. In this issue, there are less than $100 worth of paid advertisements. For normal newstand magazines, ads pay more 90% of the cost. For TCJ, subscriptions and orders for Back Issues pay more than 95% of the cost of publishing each issue.

The rest of the ads are either exchange ads like the Forth Dimensions ad (they give us an ad in exchange for us running theirs), 'payment' for articles (the only payment the writers receive), or 'public service' ads to tell you where to find things.

The Up-side of this is that you don't have to wade through tons of ads to find the articles. The Down-side is that the next issue can't be printed until we get enough renewals and new subscriptions each time. This means that when renewals are slow, TCJ is slow getting out. On the other hand, TCJ doesn't owe anybody but the subscribers, so it can't be forced out of business. Everybody will get all of their issues, just maybe not as quickly as we had planned. If renewals and subscriptions pick up, we'll get back on schedule.

In order to increase TCJ's income and to make a few items available to our readers, the TCJ Store has been 'opened'. See the TCJ Store page in the back of this issue. In addition to the Back Issues we've always sold, TCJ is now the US 'distributor' for Tilmann Reh's GIDE kits and we can provide the Walnut Creek CP/M CD-ROM at a discount. The one catch is that we have to collect a minimum number of orders for these items before we can supply them to you. It's like the group purchases your computer clubs have done at times. TCJ doesn't have the resources to 'stock' these items for you. Call, write, or email to see what the current status is for these products.

For that matter, CALL and WRITE! I get very few calls and letters. This makes it very hard to tell what you want to see in TCJ.

Bill Kibler has generously provided the TCJ/DIBs BBS with a 630 MB SCSI hard disk where we can collect a huge number of files. The BBS also has a CD-ROM drive now where you will be able to find the CP/M CD-ROM files. These will be on-line as soon as I find a power supply that will boot up the hard drive reliably.

Dave Baldwin,
Editor

We have Doug Beattie's 6502 DIY board construction project this time. Doug has also provided a 6502 cross-assembler in both MS-DOS and CP/M versions. You'll find those on his Web page and on the TCJ/DIBs BBS.

Dave Brooks has the first of his series on Simplex III, his homebuilt TTL processor and Frank Sergeant is back with Large Data In Forth and comments about the Internet.

The Centerfold this time has schematics for two 8051 related boards. The fancy one is AMR's amr552LC board with the 80C552 from Phillips and the simple one is my 8031 core schematic. Program This! is my article on 8051 startup code for interrupt operation.

Our regulars are here too, with Rick Rodman and Real Computing, Helmut with The European Beat, Herb Johnson as Dr. S-100, and Ron Anderson with Small System Support. And last is Bill Kibler with The Computer Corner.

Next Time...

It looks like #78 will be the Modem Issue. I already have 3 articles on modems including David Goodenough's article on using the AT command strings.

Other Stuff...

Here are a couple of items that I thought you'd like to know about.

There is no TCJ reference card this time, but I did find out that MICRO LOGIC still has all of their MICRO CHARTs available. These are the plastic charts that many of us have used for years to look up the Z80 instructions when we're programming or debugging. Other CPU's they have charts for are the 8086/88, the 6502, the 8080A/85, and the 68000. They also have charts for BASIC, 'C' and Unix Shell along with Wordstar, Postscript, 7400 Pinouts, Basic components, and Algorithms. The charts are $6.95 each and you can get in touch with MICRO LOGIC at (201) 342-6518.

Also available (still) are the CP/M versions of the cross-assemblers from 2500AD Software. They're not cheap at $250 each and they're available only on 8 inch disk (!), but they Are still available. Contact 2500AD Software at (719) 395-8683. They also have products for MS-DOS, of course.

The MOR (Morrow Owners Review) newsletter is no MOR, but the MOR-BBS is alive and taking calls. Jay Huddleston has been working on it to get PBBS and NZBYE running to his liking on his Morrow system. Call the MOR-BBS at 360-293-8165, 2400-8-n-1.
From: Jon Titus
<JONTITUS@cahnens.com>
Subject: Gary Ratliff’s #77 Article

Dear Gary:

Thanks for your interesting article about the TRS-80 computer. I thought I’d add a couple of references to your list and make a correction:


2. Your article states, “The December 1975 issue of Popular Electronics ushered in the computer age for the masses...” I think you’re referring to the JANUARY 1975 issue of the magazine which featured the MITS Altair. However, the Mark-8 computer was featured on the cover of Radio Electronics magazine in July 1974, and as far as I can tell, THAT computer was what ushered in the personal computer age. There were several users groups and at least one newsletter prior to the Altair. Granted, the Altair was popular, but the Mark-8 was what got it started.

Jon Titus
Milford, MA

From: Aristarco Palacios Lopez
<apl@speedy.coacade.uv.mx>
Subject: Wanted and free.

Hi everybody in TCJ!

I received lots of help with my C64. To say thanks I’m giving copies of the Speedscript code from Compute! magazine, Speedscript 3.0 for the Atari and the Apple II. And SpeedCalc (a good Lotus 1-2-3 clone) for the Atari. Just send a message to my e-mail address or send a postcard. The issues are dated 1985.

If someone can send me the C64 version of Speedscript 3.0 (appeared in March 1985) I’d be very grateful.

I’d like to know if someone has a Disk Drive for my Atari 65XE and the Operating System. I’m very interested on buying one.

Finally, thanks to Bill for takin’ time to read this and sorry to Emmanuelle Roche. I haven’t get the Beetle brochures yet, but I promise I’ll send them as soon as possible, OK?

Aristarco P.

From: Glenn Haydon
<ghaydon@taygeta.com>

Hello Dave,

Thanks for the nudge. The computer I use at Stanford lost its motherboard. A new one is on the way - today or tomorrow. I am forwarding all of my email to that and it all bounces!

Yes, I have available THE FORTH ARCHIVE from taygeta.com on a CDROM. I am making them one off. $50 + $4 + Calif sale tax, from Mountain View Press, Route 2 Box 429, La Honda, CA 94020.

I also have a home page and catalog available: http://www. taygeta.com/jfar/mvp.html

I should have written long before but that is the way it is. I have a number of old CP/M system including several H89s which I put together years ago. I also have a large library (100s of disks) of CP/M software but have not gotten to it lately. I also have 10 feet of CP/M program documentations - Many things I have not looked at in years.

I originally brought up MVP Forth on one of the old H89s and have migrated it to other platforms. The kernel dates back to 1984 and is unchanged! I would have done some thing differently but I leave it stable as is and still sell it. Almost all of the sales are for the PC. I have added applications to the distribution so that one can compile from text files as well as blocks, and many other things. Contact me.

Of late I have been learning to record CDROMs. It sounds simple, but I have had my problems. I now seem to be able to make them without the freebies.

I am also interested in Linux. I have it up on one of my PCs and like it fine. However, it is certainly not a simple out of the box process. On the other hand it is an introduction to Unix. It works fine and fast on the PC and with the GNU software, one has considerable power!

I still have the WISC CPU/16 and CPU/32 processors. There is ongoing interest in the CPU/16 as a teaching tool at several Universities and Colleges. I have been thinking of putting a description and schematics in the TCJ for non-commercial use. If there is any interest, let me know.

There are other things besides computers in my life and there are just too few hours. Retiring makes it harder, not easier!
Again, thank you for the nudge.

Glen B. Haydon, M.D.
RT 2 Box 429
La Honda, CA 94020
415 747 0760
ghaydon@forsythe.stanford.edu

--

Dear Mr. Kibler

Enclosed is a M.O. for $24.00 to renew my subscription. I have been enjoying this magazine very much. In my issue #70 there are 5 sheets between pages 43/39 and the center fold that are missing. I would appreciate it if you could get them to me so I can finish reading that issue.

I originally subscribed for the hardware and CP/M articles. However lately I find myself going to Ronald Anderson's 6809 articles first. Mind you I don't own a 6809 computer (at least not yet) but since I do most of my real programming in 6502 Assembly I was amazed at how similar they are. I think that I might feel right at home with a good reference guide and macro assembler. Most of my other programming is done in 'C', Pascal, BASIC, COBOL and now Z80 assembler. While I am on the subject of 6502, I was curious as to why I don't see any 6502 articles (not really a complaint).

Now on to other things in issue #71 Aristarco Palacios wrote that he was in search of Commodore stuff (if you haven't already guessed, that's the 6502's I am talking about) his list might be a bit beyond what he wants to pay. Starting with the harddrive, current prices for Commodore compatible harddrives are to say the least overpriced even used. Unless you find someone that's just wanting to dump what they have, you will pay a premium even for used ones. The current NEW price for a CMD 20 meg SCSI drive will set you back about $299.00 U.S. (maybe a little less) and they are in short supply a 85 meg is about $490.00 and add shipping to that! I would recommend getting the smallest that was available and adding a second APPLE compatible SCSI external drive as large as you need and use the CMD drive as a interface. GEOS for the C-64 and C-128 is about $45.00. The rest of the list I don't know what the used prices will be you'll just have to make your best deal. I do know where you might try. For new/used you can try:

Commodore Country
RT 1 BOX 333
Burleson, Texas 76028
Phone 1-817-295-7658
1-800-676-6447 <--- this might not work from Mexico!

In the used only category a fellow in the users group I belong to is selling all of his Commodore stuff you can contact:

Midessa Commodore Users Group
2411 W. Francis
Midland, Texas 79701

Ask that the letter be given to John Michael. I don't think they will do you wrong. However if you want a warranty Call Commodore Country. I have dealt with them and have been satisfied with their service.

Well enough Commodore, I noticed that you were collecting BIOS source files. I think somewhere I have source for my ALTOS 8000 but it will be on an ALTOS 8" disk and although I can copy it, I can't get it to another format. If you would like I'll set up my ALTOS and make you a copy and sent it to you. While I am thinking about the ALTOS, do you know of a place to get a harddrive that would be a compatible replacement for QUANTUM 2020. The one in my ALTOS has expired and with a single disk drive it's a real pain to use.

A while back I received my copy of the CP/M CD-ROM and have had quite a time going through all the files. I recall in a back issue of TCJ that something was said about a NON CP/M NON MS-DOS CD-ROM was being considered, has anything come of that?

Well I am going to stop here, it's midnight and I have a full day ahead

Thanks for the great mag...
Ronald Austin
706 N. Carroll
Spur, Texas 79370

------

Dear Dave,

I wonder if any of your subscribers have any information on the "Legacy" Real Time Clock which was used in the Kaypro? I would welcome any documentation and, if possible, a copy of the program which runs with it.

I look forward to your next issue.

Alwyn Stockley, KJ7DA
P.O. Box 1764
Sisters, OR 97759

------

Dear Dave,

Thanks for putting the Z80 Reference Card in the latest issue. I checked over the instruction cycle times, and my modification to William Colley's compiler had the wrong time for the "ld (nn),hl" instruction. Should have been 16, but I had 20. However, I think their "pop ix" and "pop iy" instruction times were incorrect. Shouldn't they have been 14 cycles instead of 15?

Frank Wilson
P.O. Box 55
Tomales, CA 94971

------

From: "Ronald W. Anderson"
<rwilanders@provide.net>
Subject: Prime Numbers

Dave,

There has been a rather lively dialogue (or should I say quintalogue or hexalogue) among the TCI authors regarding the Prime Number program presented in issue #77 by me. I thought the readers might like to in on it so I'll digest it here as a letter to the editor.

First, a couple of people said that 1 is not a prime number. I won't argue the point. It qualifies because it is divisible only by 1 and itself, but perhaps since 1 and itself are the same it is a special case.

I had reported times of execution on a 50 MHz 486-50 SLC/2 as follows:

Limit #primes Seconds
250,000 22,044 80
500,000 41,538 227
1,000,000 78,497 602

The number of primes differs from those in the article by 1 because of the exclusion of 1 as a prime. John Baker coded the program and ran it on his Amiga 500 running a 14.3 Megahertz 68000. His times:

continued on Page 20
Real Computing

By Rick Rodman

Small-C in PROM

For small embedded controllers, we’ve typically had to use assembly language. However, for some CPUs, Small-C offers reasonable code efficiency, so if you can live within its syntactical limitations (no longs, structs, or multidimension arrays, among other things), you can use Small-C for your embedded projects.

In this example, the hardware is Brad Rodriguez’ Scroungemaster board, the C is from C User’s Group volume 309, and the processor is a Motorola 6809. Small-C generates assembly language, so you need an assembler too. The assembler I used in this case is called AS9, and comes with source code in C. It’s a fairly simple assembler which does not support linking.

The first thing you have to do is to modify the startup logic for your particular hardware. In this case, the startup logic is contained in a header file called “startup.h”. Since no linking is possible with AS9, all header files include actual functions within them. This means you’d better watch out what you’re including if you’re concerned about code size! The Scroungemaster has 32K of PROM space, however, so there’s no need for concern there.

My modified startup.h file is in listing 1, space permitting, and the whole package from this article is on the KPCF BBS. Note the initialization routines for the Scroungemaster’s memory mapping, and the ORG and stack addresses.

A tricky issue with C going into ROM is static data. If you have initialized static data, it must be initialized before the program starts. It gets worse - you might have pointers which need to be initialized with addresses. This is where relocation becomes a real hassle. The simplest thing to do about these things, in the firmware world, is to disallow them. You can initialize your data structures yourself in C code.

C programmers often assume that static data is initialized to zero by someone else before the program loads. Well, in the firmware world, there isn’t anyone else. Do it yourself if it’s that important to you.

Where does static data come from? It usually comes from what Real Programmers disparagingly call “global variables.” Unfortunately, in most real projects it’s almost impossible to avoid using some global variables. The problem in this case is that AS9 doesn’t support separate “code” and “data” origins. Using this assembler, if you declare a global variable, it’ll be in PROM. That takes care of the initialization problem, anyway... While a “no global variables” restriction appears harsh, I’d rather work around it or fix it than switch assemblers, because to me, having source code to everything is worth it. Your feelings may vary.

A nice thing that Small-C lets you get away with is very free interchange of ints and pointers. For example:

```c
#define BUFFER 0x2000
*(BUFFER + i) = ‘0’;
```

Of course, Small-C is just a language. You may still need a tick interrupt for real-time work, hardware-interface drivers, and other things.

On a bigger project, Small-C’s syntax will be too limiting. NS32 developers can use Phil Prendeville’s C, available from KPCF. For other processors, Dunfield Development Systems offers the Micro-C compiler and development package which implements most of ANSI C.

And you’ll notice that all of these tools are running on a PC or Sun or some other “development platform”, with PROMs being burned for a “target platform”. This is a cross-development facility. At present, there isn’t any simple platform available for native embedded system programming in C analogous to Forth or TDS.

My intention is to use this setup to develop a PROM-based version of TinyTCP to use on small embedded computers. By means of a serial router and/or Little-Net interface, I’ll be able to “telnet” into each controller and interact with it remotely. Secondly, controllers will be able to send data streams to one another through TCP connections. More on this project will be dribbling out in future columns.

 Commodore 64 emulators

There is a whole world of activity going on on the Internet related to Commodore 64 software emulators. If you have a
lot of C64 software (as I do), you may want to look into this. When Bill last mentioned this topic, you needed a Hercules board. Since that time, there are several new emulators available, and an Internet newsgroup dedicated to them, comp.emulators.cbm. Here's some basic information from the group's FAQ (frequently asked questions file).

The emulator Bill described is called C64, by Johannes Kiehl. A newer emulator, C64S by Miha Peteren, is considered much better, and is available in a "shareware" and a commercial version. It requires a 386 or better and a VGA. A similar emulator from Germany is PC64 by Wolfgang Lorenz. Under development is C64 Alive by Frank Littman, as well as Windows versions of PC64. Several C64 emulators are available for the Amiga as well.

But how do you get your C64 programs to your PC? By means of a cable connected from your parallel port to a 1541 or 1571 disk drive. Several utilities are available to do the transfer, which basically amounts to making a sector-by-sector image of each diskette (with filetype .d64) on your hard disk. Because of the group-coded format used by the 1541, you can't read C-64 disks on your PC's drive.

Windows 95

Here's your official TCJ review of Windows 95. I installed it on a 486DX-33 with 250MB hard disk and 4MB of RAM. Microsoft would consider this machine, a monster in 1992, only a minimal configuration for Windows 95. Windows 95 takes a long time to install, to boot, and to do much of anything else. The "Install Wizards" are very slow, and irritatingly stupid besides. Most Windows programs I tried worked, but many DOS programs would not work, especially communications programs - they just freeze up. My recommendation is, if you've got 8MB of RAM or more, and you've got to run DOS or Windows programs, use either OS/2 Warp or Windows NT. I don't see any reason for anyone to run Windows 95.

Computer telephony

Everyone's so breathlessly enthused about the Internet and how much money it's going to make for them. It reminds me of the seventies, when everyone told me that there was so much money out there in the computer field. Where, exactly, I never could find out.

Anyway, some exciting stuff has been happening in computer telephony too. There are new voice boards available which make it possible to set up simple voice-response systems fairly inexpensively. I've been fiddling with some new stuff in this area, but don't have enough to report (or room left to report it, for that matter).

Next time

The data-stream concept for building distributed computing systems.

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

Microsoft Corporation
One Microsoft Way
Redmond WA 98052-6399

Dunfield Development Systems
(Micro-C for 6809, 68HC11, 8051, 8080, 8086, 8096)
P.O. Box 31044, Nepean, Ont. K2B 8S8
Phone: +1 613 256 5820
Fax: +1 613 256 5821
BBS +1 613 256 6289

Seattle Lab (C64S commercial version)
214 First St., Kirkland, WA 98033
+1 206 828 9001
fax +1 206 828 9011
email lab@seattle.wa.as.com

Ted Drude
(US contact for PC64, $30. Cables, manuals available.)
103 Belle Circle, Madison, AL 35758
email teddrude@delphi.com

PS: I got a surprise the other day when a developer sent me an email message about a New add-on product he was working on for the Commodore 128. When I asked him about it, he said a number of new products had come out recently. One was an accelerator board that bumped the clock speed up to 20 MHz! DB.

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 I: STARTUP.H FOR SCRUNGEMASTER II

/* startup.h used to generate startup code for C6809 */
/* */
/* 960212 rr mods for AS9, Scrounge */

#asm
ORG $ffe
FDB $8000  * Address of EPROM
ORG $8000
lds #$3ffe  * Set stack at top of RAM

* set up memory map of scroungemaster

clra $F000  * map Fxxx into 7xxx - I/O and RAM
inca sta $E000  * map Exxx into 6xxx - RAM
inca inca sta $D000  * map Dxxx into 5xxx
inca inca sta $C000  * map Cxxx into 4xxx
inca inca sta $8000  * map Bxxx into 3xxx
inca inca sta $A000  * map Axxx into 2xxx - RAM (8K chip
  * alias address)
inca inca sta $9000  * map F9xxx into 1xxx
inca inca sta $8000  * map F8xxx into 0xxx

* Init SCC ports A and B

ldx #scatbl  * port A setup table
bsr scinit
ldx #scbtbl  * port B setup table
bsr scinit

* Go to user program

jmp main

* Zilog SCC initialization routine.

scinit:
ldy .x++  * get SCC port address
ldb .x+   * get # bytes to output

sccloop:
lda .x+  * get a byte from the table
sta .y   * store it to the SCC control

port
decb
bne sccloop

rts

scatbl:
  * SCC A command address
  * 37 bytes follow
  * just in case, reset reg ptr
  * hardware reset, interrupts off
  * 16x clock,async,1 stop, no par.
  * no dma, all irpts disabled
  * irpt vector (for future use)
  * rx 8 bits, rx disabled
  * tx 8 bits, tx disabled, RTSB hi
  * status low, irpts off
  * nrz encoding
  * no xtal, BRG-xxc txc, TRxC in
  * BRG lo byte - 4800 baud at 16x
  * hi byte - w/ 4 MHz BRG clk
  * DTR pgm'd, BRG from PCLK
  * as above, plus BRG enabled
  * as above, plus tx enabled
  * no ext/sts interrupts
  * reset ext/sts interrupts twice
  * no dma, all irpts disabled

scbtbl:
  * port B command address
  * 31 bytes follow
  * just in case, reset reg ptr
  * 16x clock,async,1 stop, no par.
  * no dma, all irpts disabled
  * rx 8 bits, rx disabled
  * tx 8 bits, tx disabled, RTSB hi
  * nrz encoding
  * no xtal, BRG-xxc txc, TRxC in
  * BRG lo byte - 4800 baud at 16x
  * hi byte - w/ 4 MHz BRG clk
  * DTR pgm'd, BRG from PCLK
  * as above, plus BRG enabled
  * as above, plus tx enabled
  * as above, plus tx enabled
  * no ext/sts interrupts
  * reset ext/sts interrupts twice
  * no dma, all irpts disabled

* Following was C.AS9, with minor formatting changes
(This portion of listing deleted for clarity, since
C.AS9 is present in the distribution and is not
machine-specific.)

@endasm

/* end of startup.h */
Putting the GIDE to work on the KC

Ever since the advent of Tilmann's GIDE, many CP/M systems have gone through wondrous changes. Slowly, the circle of happy hard disk users has been growing. But even though things have become much easier with the GIDE, implementing the software side still means lots of work - and a skilled programmer.

One of the great dilemmas is the fact, that many Z80 CPUs are soldered in! This is the point, where it comes down to determine between filigrane operation and mere butchering. Beware! Your CP/M computer is not a cat and therefore does not have nine lives!

In the case of the East German KC 85/4 (remember - I wrote about it before) the CPU is a UA 880, a true clone of the famous Z80. In such a case, best get a spare CPU first, for the UA 880, a Z80 will perfectly do. The soldered-in processor must be cut out, pin by pin. This makes it easier to remove the leftovers. After cleaning the holes with a solder pump or the like, it is best to fit a socket in its place. This will also be the home for the GIDE interface. If the small contact "eyes" get too much heat, they will get destroyed, leaving the computer useless. Users with little experience might try soldering the socket directly to the leftovers of the pins, to avoid overheating the circuit board itself.

The physical position of the GIDE is rather uncritical, in fact, even mounting it upside-down (like in the KC), doesn't make any difference. When considering the size of the future hard disk, the often desired "small" sizes of 40 to 80 MBs are harder to get every month, so move quickly! It is advisable to always use a separate power supply for the hard disk, since most CP/M power supplies are probably too weak to take up more load. This is especially the case with the "D004", the main unit of the KC.

Typically, configuring the BIOS for the IDE drive is done like patching any CP/M for various reasons. Only that a table has to be designed for the Disk Parameter Block (DPB) to hold the IDE drive characteristics. With the KCs, though, we are facing a truly exotic operating system. It has its origin in a Moscow institute and returns a version number of 2.6! This is due to the fact, that some features of CP/M Plus were supposed to be integrated already. This MicroDOS, however, is not like any regular CP/M at all. Needless to say, no source comes with the machine. Experts from the KC user group tried reverse engineering, but were quite puzzled with the bland mixture of code, they found. Mario Leubner, chief wizard of the assembler crew, commented each and every routine found, resulting in 400K of text!

Gradually, MicroDOS revealed it's secrets. Basically, there are 4 parts: a KC-specific BIOS, a MicroDOS BIOS, a MicroDOS-BDOS and a MicroDOS CCP. The three latter obviously are of Russian origin. They contain many unused and undocumented functions from CP/M Plus. To make things worse, the parts are not clearly separated, but all mixed-up with each other.

Mario's first grand deed was then, to eliminate this cross-assembly and to reduce all this to CP/M format. The IDE and the Realtime Clock where integrated into the BIOS, like with any other system, as well. A lengthy debate by email between Joerg Linder and myself resulted in choosing ZSDOS as BDOS replacement. Unfortunately, the included INSTALOS could not be used, since there is no MOVCPM. (I recommended using the file \CPM\BDOS\DOSPLSOR.ARK from the CP/M CD-ROM, where sources for MOVCPM and SYSGEN are found.) For CCP, he is using a minimal version from scratch, derived from the resourced code. The new system offers 4B larger (!) TPA. Mario has been using his 42MB drive for a while already. Since the RAM-disk of the KC is only addressable via a coupling RAM of only 128 Byte size, the hard disk access is even faster.

The next step is to integrate the clock driver into the system, so DATESTAMPER can be used. (When I first passed the idea and praised the advantages of true datestamping, Mario and lots of others were against it. It works out, that Mario now is one of the hard-core fans of datestamping!) For CCP, ZCPR34 or maybe the patched ZCPR40 will be integrated into the OS. It will be one of the best Z80 operating systems, maybe the best one - after the CP/M Plus of the CPU8280 (says Joerg Linder).

I'd like to thank Joerg for his precious input for this article, I am happy, to have been of assistance, and we all hope, the KC user's have good luck with their new CP/M hard disk machines.

Regards
Helmut

This may be completed by a report from the KC user meeting after the 10th of March.
Run CP/M-86 on a Pentium-based IBM clone? Yes, it's now possible. A simple patch has been found that enables CP/M-86 to run on any AT-class computer. It's an interesting new discovery which just might breathe some extended life into this presumed-dead operating system.

To experienced CP/M-86 users, it'll seem like a miracle. Because if you've ever tried to run CP/M-86 on an AT-class IBM-compatible computer, you've already discovered the sad truth: it doesn't work. Attempting to boot this teenage O.S. on any AT-class machine results in an "Unexpected Interrupt" error message on the status line. Then everything shuts down.

Now, in all fairness, it must be noted that "CP/M-86 for the IBM PC and IBM PC XT, Version 1.1" was never intended to run on AT-class computers. But as processor technology advanced throughout the 1980's, CP/M-86's incompatibility with the AT became increasingly problematic. The operating system wasn't all that popular even in its heyday, and its inability to run on an AT was just one more nail in its rapidly-closing coffin. Mainstream computer users were moving to AT-class machines, and CP/M-86 couldn't follow.

Nonetheless, a few of us die-hard CP/M-86 fans stuck around. In 1993, while cruising various CP/M BBSes around America in search of CP/M-86 software, I happened to electronically "meet" Richard Kanarek (72371.111@compuserve.com). He and I began to correspond by e-mail, and he soon became a fellow CP/M-86 aficionado. We periodically exchanged messages about CP/M-86's AT phobia.

Then one fateful day in November of 1995, Richard Kanarek decided to take on the daunting task of trying to determine why CP/M-86 won't boot on an AT-class computer. Ultimately, his sleuthing turned up the cause of the problem — and, more importantly, a "fix" for it. This is an amazing and brilliant piece of detective work, and should certainly earn Richard his own page in the annals of CP/M esoterica. Thanks to the diligent efforts of Mr. Kanarek, we now know why CP/M-86 won't boot on an AT-class machine. At boot-up, CP/M-86 installs a default interrupt service routine (ISR) for interrupts 20h through 32h, and 41h through A0h. One can only speculate as to why Digital Research built the system that way; possibly to insure that control would be returned to CP/M-86 if one of those off-limits interrupts should be called by some renegade program gone awry.

The presence of the ISR is perfectly acceptable to PC and XT class computers, because they normally don't use the interrupts which the ISR traps. However, in an AT-class machine, it appears that the computer itself generates one or more of these "forbidden" interrupts for its own internal purposes during boot-up. When this happens, the ISR is triggered, the infamous "Unexpected Interrupt" message is generated, and CP/M-86 attempts to return control from itself back to itself — resulting in a system lock-up.

Of course, knowing the cause of the problem is one thing. But finding an after-the-fact "fix" is quite another. Richard Kanarek concluded that the most direct solution to the problem would be to prevent CP/M-86 from installing its default interrupt service routine in the first place. As luck would have it, he was ultimately able to do this by altering only TWO BYTES within the CPM.SYS file.

Here, for your edification and enjoyment, is the magic AT compatibility "patch" for CP/M-86: simply change the bytes at offset 3DAFh and offset 3DB9h in the CPM.SYS file from "AB" (a STOSW instruction) to "47" (an INC DI instruction). This disables the installation of the default ISR, and allows the "patched" version of CPM.SYS to boot and run perfectly on any AT-class computer. Incidentally, the patched system file will work just fine on PC and XT class machines, as well.

An easy way to make the patch is to transfer the CPM.SYS file over to DOS (using Sydex's shareware program 22DISK, or any other software designed for converting files between disk formats). Then, using Norton Utilities or any byte-level editor, change the two bytes as described above. Save the changes to disk. Finally, transfer the patched CPM.SYS file back to a CP/M-86 formatted double-density floppy disk, and you're all set. Shove that floppy into the A: drive of your favorite AT-class machine, and boot from it. Bingo! You're off and running.

Please understand that this patch is specific to "CP/M-86 For The IBM PC and IBM PC XT, Version 1.1." The file offsets won't correspond with any other version or permutation, so don't try it with Concurrent CP/M-86 or with any of the non-IBM O.E.M. versions of CP/M-86.

Continued on Page 14
There are many eight-bit processors, which have been around for a long time. One of these is the 6502, which has been used in a number of home computers. The 6502 is an ideal processor for a prototype. It is also quite affordable.

The 6502 prototype

A prototype is a basic model from which to build and expand upon. It allows us to evaluate and analyze a minimum system. With a prototype, we can study the operation of components functioning as a system. This allows us to gain experience with the components, and better understand how they work. It would be difficult to study a component in detail without using it as part of a system.

As an example, perhaps there is a particular RS232 communication chip you wish to learn more about. It is not practical to use the chip all by itself. We connect it to other chips (compon-ents) and utilize its ability to communicate. If it is a chip we haven’t used before, then we must learn to “speak its language,” learn how it operates. One practical way to do this is with other computer chips, on a little circuit board, working together as a system. This type of system is often referred to as a prototype.

The 6502 prototype board is surprisingly easy to build, and if you build it on a bigger board than is needed, it can be expanded and enhanced later. After you understand the minimum system (your prototype) you can add other things like a printer port, floppy drives, and even a hard disk, CD-ROM, plotter, or whatever.

At first, we want a real feeling for the capabilities of the 6502 prototype board, so at least let’s have a keyboard, and a video screen to communicate with it. A computer terminal does both. If you have a TRS-80, you could use that as well. I also remember somebody mentioning once that a dumb terminal was a perfect application for a PC. All we do is run QMODEM or TELIX or PROCOMM or whatever... and instead of initializing a modem and dialing it, we fire up the little computer board and compute on it.

Overview

This is the computer board; you should build it. It’s just a prototype, but the potential is here to expand and modify into many useful things. It consists of the following circuit chips:

6502 CPU - the heart or “engine” of the system.

2K or 8K EEPROM (jumper selectable) - a special type of Read-Only Memory, also known as “E-2-prom,” this is like EPROM but no dangerous ultraviolet rays are required to erase the chip. It is erased electronically.

32K Static RAM - usually called “S-ram,” static RAM is simple to interface.

6522 VIA (Versatile Interface Adapter) - combination of two parallel I/O ports (each with eight synchro-nized lines, like for a printer device), and timers. Timers are useful for one-shots or counting duration, or alarms for overtime, etc.

One-shots: something happens for a precise timed duration. It might be a length of a tone, squirt of fuel or a dose of water, or a shot of air.

Duration: how long something happens... perhaps how long after dawn your street light finally decides to go off. How long it takes your little heater to heat the backyard pool up to comfy.

Alarms (time-out): been in the freezer room for too long? an alarm sounds.

6551 ACIA - the Asynchronous Communications Interface Adapter. Don’t let the long technical name fool you. It’s fairly simple. Basically, it provides the interface to a modem or to another computer.

The ACIA connects your terminal to the computer board, so you can interact with your computer board, update information, set parameters of operation, duration of timing, error correction factors, etc.

Basically, that’s it... CPU, RAM, ROM, VIA and ACIA

Other Chips

The 7404 (chip U2) is an inverter. It produces a high level from a low input, and a low level from a high input. The 74LS139 (chip U3) is similar to the popular 74LS138, but instead of producing eight signals from three lines, it pro-
duces four signals from two lines. The 74LS139 also includes two identical decoders within one package.

The MAX233 (chip U7) is a single-chip solution for RS232 line drivers/receivers, producing the proper voltage transitions between 5-volt TTL level and 12-volt RS232 level. It is an ideal chip for serial communication when only 2 input and 2 output lines at most are needed. No external capacitors are required, and it draws very little power.

Theory of operation

System ROM is jumper-selectable. The jumper option allows either 2K ROM or 8K ROM, addressing the 2K option at $F800, or the 8K option at $E000. The schematic references an 8K (28-pin) socket, but no distinction is made between the two — either ROM will work effectively if the jumper is set correctly.

Address decoding for upper memory is accomplished using 1/2 of a 74LS139 (chip U3A).

The 6522 VIA is decoded at location $C000..C0FF. Chip selects come from chip U3A and address line A12 (inverted).

The 6551 ACIA is decoded at location $D000..D0FF. Chip selects come from chip U3A and address line A12 (true).

The CPU clock comes from a 1 MHz crystal oscillator. VIA and ACIA clocks are driven from the buffered CPU clock PH2 (for greek letter “phi”, and 2). The ACIA also utilizes a separate 1.8432 MHz crystal oscillator for baud rate generation.

A full 32K of RAM is located at $0000..7FFF. Chip select for this RAM is provided by CPU address line A15; when A15 is low, the 32K RAM is selected. Read and write signals are provided by U3B, another 1/2 of a 74LS139.

Construction

It seems that wire-wrap is the way to go. Now, if you have tried to wire wrap before, and were not successful, this time you should be:

1. because there is a wirewrap list.

2. I drew the schematic to represent the component layout.

You don't HAFTA build it that way, but I tried to make it easy for you. With a few sockets and a perfboard, you can build the prototype.

There is a drawing of where the sockets should go - try to position the sockets on the perfboard according to the diagram. There is a schematic which closely resembles the actual layout.

There is a systematic wire-wrap list - not much more difficult than a dot-to-dot.

There is also a parts list. I picked all parts that are available from Jameco Electronics. Jameco has been around for many years and has competitive prices.

Be sure to use Wrap ID tags. These are worth the extra cost. The task of wire wrapping becomes a smoother process.
Parts

The toughest part of this whole project will probably be filling out an order.

Filling in an order form, and punching in all those part numbers is not exactly my idea of fun. Any spreadsheet program makes light of the job. I used a really old version of Lotus 1-2-3 (the one that runs on a PC-XT) — it worked just fine. And evidently Jameco didn’t mind. Their service was fast.

Call Jameco for their latest catalog at (800) 831-4242, or e-mail them at info@jameco.com, or request a catalog through their computer bulletin board at (415) 637-9025.

Parts List:

1 2816A-25 EEPROM, 2K x 8, 250nS (CMOS)
1 43256-12L Static RAM, 32K x 8, 120nS (CMOS)
1 65C02 MPU with internal clock, 1 MHz
1 6522 Versatile Interface Adapter
1 6551 Async. Comm. Interface Adapter
1 74LS139, 2-to-4 Decoder/Demultiplexer
1 74LS04, Hex inverter
1 MAX232CPP, +5V powered, Dual RS-232 Txm/Rx
1 Crystal Oscillator, 1.8432MHz
1 Crystal Oscillator, 1MHz
1 TP-2381 Push button switch, 20mA @ 15VDC
1 10uF electrolytic capacitor
1 Resistor, 10k-ohm
1 2852 PCB Prototyping Board, 3.60 x 7.00, 2225 holes
4 14-pin Wire-wrap sockets, 2-level
1 16-pin Wire-wrap sockets, 2-level
1 20-pin Wire-wrap sockets, 2-level
3 28-pin Wire-wrap sockets, 2-level
2 40-pin Wire-wrap sockets, 2-level
1 SMH03 header, 3-pin
1 .100” shorting block
5 T44 Miniwrap Wiring Terminals
3 Wrap ID, 14-pin
1 Wrap ID, 16-pin
1 Wrap ID, 20-pin
3 Wrap ID, 28-pin
2 Wrap ID, 40-pin

Software

If you do decide to build this prototype, and then expect it to do something, it will need two things:

An application - what the computer will be applied to do. Applications can be simple or complex, depending...

A program in ROM. That would be software, but we call it firm-ware when it is programmed into an EPROM or EEPROM. Program development usually requires a cross- assembler. You can also “hand-assemble” without an assembler program.

I wrote a short test program to verify that the board is working. BBOARD.ASM transmits a sequence of “01234567” repeatedly to the serial channel. while at the same time receiving keys through interrupt. Whenever a key is received, it is also re-transmitted to the serial port.

While all this is happening, parallel port B is turning LEDs on and off. This assumes that you have put eight LEDs on the parallel port.

You might wire a 20-pin socket to port B of the VIA (as I have done). By using a ribbon cable with IDC socket plugs on each end, you can do all further experimenting on a breadboard. Use port B, because it can sink more current (handle a bigger load). The test program BBOARD.ASM is only to prove that the board is working. I might provide a follow-up article which will give some more software examples. The source for BBOARD.ASM is on the TCJ Web page and on the TCJ/DIBs BBS.

If you already have a 6502 assembler, then cool. Otherwise, scour the internet; I have found 6502 cross assemblers in any one of six different locations. I also wrote one for IBM-compatibles. This is MAS65.EXE and can be found at my web site, URL:

http://www.whidbey.net/~beattidp/comput/x65tools/

If you don’t have access to the internet, my advice is GET IT. You have no excuse not to. It is affordable, easy to learn, and readily available. It can be accessed through an x86 PC or a CP/M machine or even a dumb terminal. You probably would use the x86 PC since more cross assemblers are available for the PC. I also saw a CP/M cross- assembler for the 6502 somewhere.

Applications

You tell me! What would you use a small computer board for? (Note one example above for the deep freezer time-out alarm.) Feel free to send some suggestions via E-mail or Mail. The 6502 prototype is rather powerful; plenty can be done with it.

More To Come

In the future, we will look at ways to enhance the 6502 CPU, expanding its ability. I will provide functionally equivalent TTL and PLD examples, along with a diagram of the state machine.

Further Reading


Ronald A. Reis, Electronic Project Design and Fabrication, Merrill Publishing Company, pp80..83, 1989

Rodney Zaks, 6502 Applications, Sybex Inc., 1979

Rodney Zaks, Programming the 6502, Sybex Inc., 1980

References and Data

Rockwell International, R650X and R651X Microprocessors (CPU), Document No. 29000D39; Rev 8; June 1987
Does throwing away the default ISR adversely affect CP/M-86’s performance? Not that I’ve been able to discover. I’ve personally tested this patched version on 286, 386, 486 and Pentium-based IBM clones, and haven’t encountered any difficulties. Naturally, that’s not a guarantee, and your experience may differ. So if you make use of the patch information described in this article, you do so entirely at your own risk and peril.

The only potential problem that’s been discovered (and this has nothing to do with the patch itself) involves creating a CP/M-86 partition on some AT-class machines’ hard disks. A few of the newer hard disks and/or host adaptors do some ultra-fancy sector-and-track translations. These electronic shenanigans appear to conflict with the way CP/M-86 addresses the hard disk — and in some cases, can prevent a partition from being created properly. But if you’re using a fairly “normal” hard disk and controller, chances are you won’t have any trouble creating a CP/M-86 partition on most standard MFM, RLL, IDE or SCSI hard drives.

The AT-compatibility patch comes a decade too late to save CP/M-86 from its decline into relative obscurity. But the patch does offer some renewed possibilities for this neatly-crafted and under-rated operating system. Those of us who enjoy keeping it alive can take a bit of comfort in the fact that CP/M-86 has, at last, entered the ‘90’s. Thank you, Richard!

Contact Information

Rockwell International
(R714) 833-4600
http://www.rockwell.com/

NEC Electronics Inc.
(415) 960-6000
http://www.nec.com/

Xicor Inc.
(408) 432-8888
http://www.xicor.com/

National Semiconductor
(800) 272-9959
http://www.nsc.com/

Maxim Integrated Products
(408) 737-7600
http://www.maxim.com/

About the Author

Douglas Beattie Jr. is an independent consultant with no degree of any kind — he was much too busy studying the minutiae of state machines and combinatorial logic to be sitting in a classroom, learning at a standardized rate. He enjoys reading data-books and scrutinizing waveforms; designing prototypes; preparing technical reports, and documenting his work for future reference. His personal library consists of over 450 books and technical manuals, with not one work of fiction.

Douglas Beattie Jr.
P.O. Box 47
Oak Harbor, WA  98277-0047
email: beattidp@whidbey.net
Handling Large Data in a Small Forth

My main topic this time deals with the problem of handling a collection of data that is too large to fit in the Forth dictionary. To start off, though, I'll touch on a number of other subjects.

The Internet

I did sign up with Eskimo North in Seattle for 5 years. My email address there is pygmy@eskimo.com. I also paid Pobox for 12 years of email forwarding service. This should make pygmy@pobox.com a fairly stable address. Currently, any mail sent to pygmy@pobox.com is merely forwarded to my "real" account: sergeant@axiom.net. I never know how long I'll keep Axiom.

I had set things up with a program running in my directory on Axiom's computer to collect a few newsgroups and collect my incoming email, zip it all up, and download it to my pc. Then, off-line, I would read and reply to newsgroups and email using the program Yarn, running on the pc. Yarn produced a zip file which another program running on Axiom would upload, unzip, and mail or post. This was working fairly well until Axiom stopped keeping the newsgroup files on its own machine. Instead, Axiom contracted with another provider to supply remote news access. This was a disappointment to me, as the programs I was running on Axiom cannot handle the remote news server. There may be a way around it, but I haven't found it. I still use those programs to download and upload email messages, but now I use similar programs running on Eskimo to collect and post my newsgroup messages. I still need Axiom to give me the local phone call that allows me to access Eskimo.

I can read news on-line via Axiom if I bring up the Windows software to do so. I don't like this as much as reading news off-line. Off-line, in Yarn, as soon as I press a key the requested action happens, such as moving to the next article. Reading news on-line requires putting up with an annoying delay. From a human factors standpoint, I think waiting on a computer rots your brain and eats away your soul. If I can ever figure out whether I'll be in school for the next year, I might drop Axiom altogether and use my SWT account to access Eskimo for both newsgroups and email. In that case, I'll have Pobox forward mail to my Eskimo account. We really do have three ISPs in my little town of San Marcos, not counting SWT. I am still amazed.

I'm using a Web browser more and more, and using ftp less. For file transfers, it is very convenient to click on the file to be downloaded. I usually use a text-only Web browser named lynx. You probably have it on your system, too, if you have a shell account with your ISP. You can see my Web page by typing

```
lynx http://www.eskimo.com/~pygmy
```

or by plugging in the address at the right place in Netscape or whatever other browser you are using. I'm usually not interested in seeing pictures when I use a Web browser, so lynx has been fairly convenient for me, but I'm beginning to use a graphical browser more often. Even then, I usually turn off the pictures so the browser will run faster.

Among other things on my web site, I have various Forth links, a few jokes, and a picture of myself as an innocent child.

Various means of searching have evolved in response to the enormous amount of data on the internet. Lycos, Yahoo, Dejanews, and other web sites allow you to search for documents and people. Have you lost someone's email address? Perhaps he has posted something containing his new address to a newsgroup. Dejanews might help with that. I won't go into the details of how you use them. Just jump in with your browser and follow various links and read various documents until you find what you are looking for. My Web page has links to a few such sites.

I do question whether I can justify Axiom's monthly cost of $30 + tax (for up 150 hours per month -- I use about 30). Are any of you getting email via a BBS or FIDOnet, etc.? How is that working out?

It's a GUI World

I always thought "graphics" meant "writing." so I stay somewhat concerned about the term Graphical User Interface. Nevertheless, today it means a display on a computer monitor that is more involved than a mere fixed-font text-based display. I certainly see the need for pictures and drawings on a computer. What I fail to understand is why anyone would consider proportional fonts and tiny fonts to be an improvement. I guess it came about because of the idea of
WYSIWYG (what you see is what you get) plus accepting poorly drawn characters in return for allowing pictures on the screen at the same time as text. Am I the only one who hasn't invested in a super duper monitor? (As it is, mine does have a 28 mm dot pitch, so it isn't the very worst monitor I've ever seen.) I can understand the desire for WYSIWYG when you are laying out a newspaper or a printed circuit board. For ordinary text, I think it is a hindrance. Of course, you don't really get what you see and you don't really see what you get. Several times I've missed spotting an unwanted character that crept into a document between the last printable character of the sentence and the period because the proportional font 'space' is so narrow. What might be better than WYSIWYG is WYWIWYG (what you want is what you get).

J and Forth Conferences

I notice that J (the APL-like language I mentioned in my previous article) is having a conference in Toronto on June 24-25, 1996. This is just after the Rochester Forth Conference in Toronto June 19-22. I think it would be great fun to attend both and contrast the different cultures and approaches to software development. I don't think I'll get to either conference this year, but if you do, please write to tell all about them.

Linux

I mentioned in my previous article that a Linux CD ROM could be had from Daniel Jimenez for $10 plus $3 shipping. That still doesn't sound bad. Splurging, though, I bought the InfoMagic 5-CD Developer's Resource set. This includes several Linux distributions, including a version of Slackware newer than the one I got from Jimenez, and lots and lots of other material related to Linux. I haven't had time to go through all of it yet. I ordered it via email late at night on January 17 and received it on January 20. The cost, including shipping, was $30. So, my experience with InfoMagic has been positive. But, not everyone has been happy with them! Ed Ngai had a very unpleasant experience with their technical support line and summarizes it as "I spent $44.00 for 22 min. of nonsense tech. support."

Ed eventually figured out what he needed without InfoMagic's help. There is a quick summary of how he partitioned a 16GB SCSI drive to work with Linux in the table.

<table>
<thead>
<tr>
<th>Device</th>
<th>Boot</th>
<th>Begin</th>
<th>Start</th>
<th>End</th>
<th>Blocks</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/sda1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td>111248</td>
<td>1</td>
<td>DDS 12-bit FAT</td>
</tr>
<tr>
<td>/dev/sda2</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>1665</td>
<td>1693696</td>
<td>5</td>
<td>Extended</td>
</tr>
<tr>
<td>/dev/sda3</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>62</td>
<td>52208</td>
<td>6</td>
<td>DDS 16-bit &gt;=32M</td>
</tr>
<tr>
<td>/dev/sda4</td>
<td>63</td>
<td>63</td>
<td>63</td>
<td>163</td>
<td>103408</td>
<td>6</td>
<td>DDS 16-bit &gt;=32M</td>
</tr>
<tr>
<td>/dev/sda5</td>
<td>164</td>
<td>164</td>
<td>194</td>
<td>194</td>
<td>31728</td>
<td>82</td>
<td>Linux swap</td>
</tr>
<tr>
<td>/dev/sda6</td>
<td>195</td>
<td>195</td>
<td>1023</td>
<td>1023</td>
<td>848880</td>
<td>83</td>
<td>Linux native</td>
</tr>
<tr>
<td>/dev/sda7</td>
<td>1024</td>
<td>1024</td>
<td>1665</td>
<td>1665</td>
<td>657392</td>
<td>83</td>
<td>Linux native</td>
</tr>
</tbody>
</table>

"basically ... yes! you can run linux from a logical partition."

Speaking of documentation and technical support, did anyone get a copy of Russ Walter's _Secret Guide to Computers_ that I recommended? Did you enjoy it? Also, did I mention that I think the Linux CDs (and the Linux and other documents on the internet) are a useful source of hardware information about the IBM PC? All of these alternative operating systems need to interact with the hardware. If they don't use DOS or BIOS calls to do the work, then they need to have their own routines. So, keep Minix, Linux, FreeBSD, NetBSD, MMURTL, and maybe Oberon in mind when you need to figure out how to interface to the PC hardware.

TradeOffs

There are tradeoffs everywhere we look. I recently studied some famous string searching algorithms (in a course at SWT). The point that impressed me the most was most of the speed up comes from simple improvements to the naive algorithm. After that, a great deal of extra complexity does speed the search up some, but relatively little. That final extra bit of speed could cost a lot of programmer time. Depending on the application, that extra programmer time might never be recovered. (I shuddered to think of the man-years spent developing those algorithms. Who paid for that? If it was a labor of love, then no problem. However, I suspect a good bit of our tax money went into it.)

One tradeoff often overlooked is that of "cognitive burden." As the software world gets further from Keep It Simple, the cognitive burden seems to increase. A typewriter was straightforward. You could see how it operated. Now we have Word Perfect and Microsoft Word and they are harder to learn and harder to deal with (that's what I mean by cognitive burden). Even if there is a net gain to the end user, increased complexity surely places a burden upon the programmer. I am thinking now of these great software libraries. You can buy a communications library, a database library, and on and on. I am becoming suspicious that the cognitive burden of dealing with the tremendous complexity of the libraries is too much. Perhaps it would be cheaper to write the routines yourself, that you really need, than to look up and understand and learn to use the canned library routines. This is part of why I am in favor of small, simple Forths -- small enough that you can understand them completely. More is not always better.
Tcl/Tk

I've been playing some with Tcl/Tk, a string-based scripting language and its companion set of GUI widgets for building windowed applications. The internet is filled with information about Tcl/Tk so I won't try to write another tutorial. I'll just tell you my first reactions.

One of its strong points, perhaps its main point, is the promise of platform independence. Its applications are supposed to run on Windows, Unix, and the Mac with essentially the same source code. I don't know about the Mac, but I think it really only runs well under Unix. It runs under Windows, but there are little annoying things, such as the screen blacking out when you try to execute a system command. There may be ways around this or it may be fixed in the future. John Ousterhout, previously at Berkeley, is the force behind it. Now that John has moved to Sun, Tcl/Tk now has corporate support as well, so I expect it to continue to improve. As I see it today, you get platform independence as long as the only platform you use is Unix.

Another touted feature is how easily it can be extended with C routines, or how easily it can be embedded in a C program. This lets you write your workhorse routines in the faster C language, but co-ordinate them with the more comfortable, convenient, interactive Tcl. This makes sense, yet I believe we have the same ability even more conveniently with Forth. I have just illustrated one way to do this with Pygmy in my article "Coordinating Pygmy and C" that appeared in the March/April 1996 issue of _Forth Dimensions_ (Vol XVII, No 6). Sample code showing how to call C library graphics routines from Pygmy appears in the article and also on my Web page. It just seems to me that Tcl works too damn hard at doing something that should be much easier. It is supposed to have simple syntax. I find its syntax regular, but awkward. It takes a good bit of work (and perhaps reading several books) to get up to speed on it. Perhaps it earns its keep when developing X windows applications on Unix (where Tcl runs a little more smoothly), but I suspect you'd be better off with Borland C/C++ or Delphi or Forth or J if you are developing just for Windows.

Tcl does have a thorough set of list and string handling functions, the usual C-like control functions, and associative arrays, so it may not be as bleak as I make it sound above. Once you get the hang of it, it is fairly easy to put up windows, pick lists, check boxes, buttons, sliders, scroll-bars, and so forth. It is a bit tricky learning how to position these items just where you want them. Tcl/Tk is free and has lots of add-on packages, but many of these require you to recompile Tcl.

Forth Questions and Answers

I occasionally get questions about Pygmy. I've collected a few recent ones here along with my answers.

Q. In Pygmy, is there a place that explains what the stack comment abbreviations are (like what n is, etc)?

A. Here are some common abbreviations I use

- n: 16-bit signed number
- u: 16-bit unsigned number
- #: usually means a count
- a: address
- f: usually means a flag

I like simple stack comments that don't have too much clutter. I write the stack comment for CMOVE, which takes a from address, a to address, and a count as ( from to # -). The inputs are to the left of the hyphen and the outputs to the right. In the example for CMOVE, there are no outputs, but I still put the hyphen, so I don't have to remember whether the items are inputs or outputs.

Q. What are DEFER'd words?

A. A DEFER'd word gives an extra level of indirection to make it easy to redefine the action of a word later. For example, EMIT makes a good candidate for a DEFER'd word. Most of the time we want EMIT to print to the screen, but sometimes we want it to print to the printer or a disk file or whatever. So, the word (EMIT) does the real work when we print to the screen and (PEMIT (or whatever) does the real work when we print to the printer. Outside a colon definition we alter which word EMIT will execute by typing something like

` : ( EMIT IS EMIT

Inside a definition we alter it with something like

` : >PRN ( - ) [ - ] ( PEMIT IS EMIT :
` : >SCR ( - ) [ - ] ( PEMIT IS EMIT :

Q. I was looking for the definition of EMIT and got no further than scr #45 which told me that this was a DEFER'd word?

A. The easy way to track down a word that is currently in the dictionary, such as EMIT, is to type

V EMIT

to pop you into the editor at the word's definition. This shows that EMIT is a DEFER'd word. Then, to see what word EMIT is pointing to, type

SEE EMIT

which probably says (EMIT). Then you could type

V (EMIT

to see the definition of (EMIT).

Q. How can I disable ok from appearing in cases where, for cosmetic reasons, it is undesirable?

A. What I do in these cases is to add KEY DROP to my code. That makes the display wait until I press a key before returning to the OK prompt.
The Big Question

Q. What can I do if my data won’t fit into the 64KB segment available to Pygmy? I have a large amount of static data to contend with, mostly 64-bit floats, and not all of it is amenable to being put in some structure on the heap. Basically what I’m doing can involve several megabytes of data.

A. Let’s look into this a piece at a time. The 64KB segment into which Pygmy’s dictionary must fit can contain a lot of source code, so the desire to have a larger address space usually comes about from problems of data storage rather than of code storage. So, if we could just put the data somewhere else, a small Forth wouldn’t seem so limiting after all.

We will start with a simple example of how to handle an array of 16-bit integers. If the array is small enough, the most convenient solution is simply to put it into the dictionary. We will do this first and then consider how to modify the code to put the array on disk and still have the very same convenient access to it as if it were in the dictionary.

Since we might want to define more than one array of integers, let’s write a defining word to create such an array:

```forth
: ARRAY ( u - ) ( u a )
CREATE ( u ) CELLS ALLOT
DOES> ( u a ) SWAP CELLS + ;
```

The first thing you might notice is that ARRAY has two stack comments. The first shows what happens when ARRAY executes. The second shows what happens when a word defined by ARRAY executes. In our example, we will use ARRAY to define an array named MARCH-SALES. The first stack comment tells us that when we define MARCH-SALES we must pass to ARRAY the number of elements the array should contain. The second stack comment tells us that when we execute MARCH-SALES we must pass to it the number of the element we want to access and that the result will be the address of that element. Note that the additional stack comment right after DOES> indicates a count and an address are on the stack. Where does the address come from? When the child of ARRAY executes, the caller put the count on the stack and, because it is a DOES> word, the runtime portion of DOES> puts the address of the child’s parameter field on the stack automatically. This address is the address of the beginning of the array. To point to the requested slot in the array, we must add the correct offset (i.e. the number of the desired item times the length of each item). That’s what the SWAP CELLS + does.

This first version does no error checking. It depends upon the word CELLS to convert a count of integers into the corresponding count of bytes. In a 16-bit Forth such as Pygmy, it could be defined as

```forth
: CELLS ( cells · bytes ) 2 * ;
```

Once ARRAY has been defined, we can use it to define an array named MARCH-SALES which will have 31 slots, each slot holding the count of how many lemonades were sold on that day.

```forth
32 ARRAY MARCH-SALES
```

When the word MARCH-SALES is executed, it returns the address of the slot for a particular day. The array is zero based, although we could change the definition of ARRAY to make it one based. For now, we will leave it zero based and just waste the zero cell. That is why we declare MARCH-SALES to have 32 slots instead of the 31 slots a calendar might suggest. Suppose we want to change the count stored for March 3rd to 75:

```forth
75 MARCH-SALES !
```

Suppose we want to get the count stored for March 17th:

```forth
17 MARCH-SALES @
```

This method of handling the array is very simple and easy to follow. It also makes processing the array in a loop easy to do. The following would print a report of the counts for each day in March:

```forth
: REPORT ( - )
CR " Number of Sales per Day during March, 1996" CR
CR " Day Count" CR
31 1+ ( day )
CR DUP 5 .R 4 SPACES DUP 5 .R
NEXT ( day ) DROP ( ) ;
```

The 5 .R prints the number right justified in a 5-character field --to make the report line up nicely.

If we only had this one short array, we would just keep it in the dictionary. But, nevertheless, we will use it as a very simple example of how to move the array to disk so it doesn’t take up space in the dictionary. After we change MARCH-SALES to reside on disk, it should be interesting to see how we must change our _use_ of MARCH-SALES so it can work from disk.

To store the array on disk, we take advantage of Forth’s built-in virtual memory, via the word BLOCK. BLOCK is given the requested block number, loads the block from disk into a disk buffer (if it is not already in a disk buffer), and returns the address in memory of the disk buffer.

``` forth
Suppose we find that we have block 2017 available in YOURFILE.SCR and that the file is automatically opened when Pygmy starts up. Merely by saying 2017 BLOCK we get the address in memory of the data on that block. How convenient! Since we only have 32 16-bit integers in the array, we only need to store 64 bytes on the disk. We could rewrite the defining word as

```forth
: ARRAY ( u - ) ( u a )
CREATE ( u ) DROP
DOES> ( u a ) DROP CELLS 2017 BLOCK + ;
```

Note in this particular case we do not need to tell ARRAY how many items will be in the array. We could have changed the first stack comment to (-). Instead, I want to keep the stack comments for ARRAY exactly as they were in the first
version of ARRAY (the one that kept the array in the dictionary). So, when ARRAY executes, it simply DROPSt the number of elements. By keeping the stack comments exactly the same, the definition of ARRAY is the _only_ change we must make! We change that definition then reload our code and the definitions of MARCH-SALES and of REPORT do not need to change at all. So, the use of the array is exactly the same, whether it is on disk or in the dictionary! (The only exception is UPDATE discussed below.)

I’d better point out that this most recent definition of ARRAY has two serious limitations! First, we have taken advantage of the fact that the entire array (64 bytes) will fit within a single disk block (1024 bytes). The second limitation is that the starting block number (2017 in the example) is hard coded. Thus, we’d better not define more than one array, or they would conflict with each other. Let’s fix the second problem first by passing a starting block number to ARRAY instead of a count (since we aren’t really using the count anyway):

```
: ARRAY ( starting-block - ) ( u a )
   CREATE ( starting-block )
   DOES> ( u a ) # BLOCK SWAP CELLS + :
```

We save the starting block number when ARRAY executes by ‘commaing’ it into the child’s parameter field. Then, when the child executes, the first thing it does is fetch the starting block number. This requires us to change how we define MARCH-SALES to

```
2017 ARRAY MARCH-SALES
```

but allows us to define additional arrays that won’t conflict with each other, such as

```
2018 ARRAY APRIL-SALES
2019 ARRAY MAY-SALES
```
supposing, of course, that blocks 2018 and 2019 exist and are available for our use.

Next, we tackle the problem of how to allow the array to span more than one block. This is a simple extension of what we have so far. Instead of just adding an offset to the beginning of a block buffer, we must calculate a two-part offset. The first part is how many blocks deep the element we want is located. The second part is the usual byte offset we used previously.

```
: ARRAY ( starting-block - ) ( u a )
   CREATE ( starting-block )
   DOES> ( u a )
   SWAP CELLS 1024 U/MOD ( a rem quot )
   ROT @ + BLOCK ( rem a ) + :
```

First we find the total offset in bytes to reach the element we want with SWAP CELLS then we divide this number of bytes by 1024 (the number of bytes per block) with U/MOD. This produces a remainder and a quotient. The quotient is the number of full blocks we must add to the starting block to get to the block containing the element. Then we add the remainder to get the actual address inside the block buffer. A minor point is that instead of dividing the number of bytes by 1024 we could have divided the number of elements by the number of elements that fit on a single block. In other words, it doesn’t make any difference whether we divide the number of bytes by 1024 or the number of cells by 512, as long as all the numbers involved are small enough to fit into a 16-bit integer. If we have more than about 32000 elements in the array, we’d prefer to divide cells by 512. In that case, we’d have to consider whether the remainder represented cells or bytes. So, the equivalent definition would be

```
: ARRAY ( starting-block - ) ( u a )
   CREATE ( starting-block )
   DOES> ( u a )
   SWAP 512 U/MOD ( a rem quot )
   ROT @ + BLOCK ( rem a ) SWAP CELLS + :
```

In the several disk versions we’ve assumed we had a block file open with space available on it for our arrays. You might need to create a file of the correct size and open it in the right place. One of the easiest ways is to copy an existing block file at the DOS level and then open it in Pygmy and extend it to the desired size in the editor with the F9 key. Suppose you want two very large arrays, each holding nearly half a million integers. You could create two files, perhaps ARRAY1.BLK and ARRAY2.BLK as just described, containing 1000 blocks each. Then, you could open the files and declare two arrays like this:

```
" ARRAY1.BLK" 6 OPEN
" ARRAY2.BLK" 7 OPEN
```

6000 ARRAY COUNTS
7000 ARRAY PRICES

to devote all 1000 blocks in ARRAY1.BLK to the COUNTS array and all 1000 blocks in ARRAY2.BLK to the PRICES array.

We haven’t done anything about range checking. I would ordinarily just try to guarantee that the children of ARRAY were never passed an out of range element, rather than trying to catch it with a runtime check. But, your approach to error handling might differ, so we could store the number of elements allowed and do a runtime check to be sure the requested element is in the proper range.

One point is that you have a lot of flexibility to handle the data storage exactly the way you wish. Another point, is that the rest of the code can be used essentially unchanged regardless of whether you store the data in the dictionary, in a fixed place in a single block, or in a range of blocks. (The only change needed is discussed later -- see UPDATE.) These days many disk controllers have built-in cache RAM and often the OS will cache the disk data also. All of this is in addition to the control you have of how many Forte disk buffers you set up. Because of this, you might neither see the disk activity nor think the performance hit you might expect by storing the data on disk.

The above examples assumed each array element was 16-bits wide. It is just as easy to setup arrays of elements of any size up to 1024 bytes wide. Suppose we had the words F@, F!, F+ etc to fetch and store and add 8-byte wide floating
point numbers. We could define an array of floats to reside on disk as

```
: FARRAY ( starting-block - ) ( u · a)
CREATE ( starting-block ) .
DOES> ( u a)
SWAP 128 U/MOD ( a rem quot)
ROT @ + BLOCK ( rem a) SWAP 8 * + :
```

We could go one step further and define a two dimensional array of floats. In this case we need to specify how many elements are in each row.

```
: 2FARRAY ( starting-block width - )
 ( row column · a)
CREATE ( starting-block width) . ,
DOES> ( r c a)
DUP 2 + @ PUSH ( save starting block)
( r c a) @ ( r c width) ROT * +
( element) 128 U/MOD ( rem quot)
POP + BLOCK ( rem a) SWAP 8 * + :
```

Now we could define MILLIBARS as a 2-dimensional array of air-pressure readings on a 50x50 grid.

```
7000 50 2FARRAY MILLIBARS
```

and then get the value of element row=17, column=30 with

```
17 30 MILLIBARS F@
```

Note, when the data reside on disk, you must use the word UPDATE after changing any element so the disk buffer will be marked as needing to be saved back to disk. Thus, the example of

```
75 3 MARCH-SALES !
```

would become

```
75 3 MARCH-SALES ! UPDATE
```

Disclaimer: the code is untested. Check it out if you use it, and please send me any corrections. It should give you a starting point for handling large amounts of data with a small Forth.

Contact info:

Eskimo North  info@eskimo.com

Forth Interest Group
(510) 893-6784
http://www.forth.org/fig.html

J and its conference:
Anne Faust (612) 470-7345 amfaust@aol.com
http://www.jsoftware.com

Linux
Daniel Jimenez
adras@crl.com
InfoMagic

(800) 800-6613, (520) 526-9565
info@infomagic.com
http://www.infomagic.com

Pobox  info@pobox.com

Rochester Forth Conference:
Larry Forsley (716) 235-0168
http://macs.dcss.mcmaster.ca/~ns/96roch.html

Russ Walter  (617) 666-2666

Reader to Reader - continued from Page 4

Limit  Time
250,000 168
500,000 441

Dave Baldwin entered my code and compiled it with Borland Turbo C 3.1 and had the compiler generate 386 object code. He ran it on a 486-DX/33 and reported times:

Limit  Time
250,000  4.67
500,000 11.04
1,000,000 27.36

Dave correctly deduced that my compiler had generated 286 code. I replied that my Turbo C 3.0 could not generate 386 code. I re-ran my program (in 286 code) on my 486-100 DX/4 system and times were:

Limit  Time
250,000  17.7
500,000 48.5
1,000,000 125.2

Today at work I compiled the program for these three limits with Borland C++ 4.5 set to generate 486 code, and to optimize for speed, not code size. Times on my 486-100 DX/4 were:

Limit  time
250,000  1.49
500,000 3.62
1,000,000 8.90

These are obviously about eleven times faster than the same computer running 286 code. I had stuck with Turbo C 3.0 since that was the last version that could be run under DOS. 4.0 and 4.5 run only under Windows (UGH). They both still can generate object code that runs under DOS. Such is the price of progress. Since the article was written, my "current computer" ran the program about 53 times faster (thanks of course to the suggestions of the others).

Ron Anderson  rwilanders@provide.net
Simplex III
by Dave Brooks

Simplex-III: A home designed computer

This series of articles describes the design and construction of "Simplex-III", a home-built CPU, using TTL parts. After two false starts in the early 1970's, the final machine was built over the period 1975 - 1977. Testing resources were minimal: I was occasionally able to borrow a 20MHz oscilloscope, and I had a multimeter and a homemade logic probe. That was all. This was the reason for the very thorough monitoring of internal states: it is possible to step the machine one clock at a time, and see the effect on (almost) every internal flipflop at each step. Rather like modern boundary-scan testing.

The principal features are summarised in Table 1.

| TABLE 1. |
|-----------------|-----------------|
| Principal features: | 8 bits wide   |
| Data paths: | 1 to 8 bytes    |
| Operad size: | 64k             |
| Address space: | 1.68uS          |
| Machine cycle: | 8 cycles (13.5uS) |

The entire CPU fitted into 109 TTL/MSI parts, including monitoring functions.

The machine still exists, and was recently run-up for the first time in a dozen years. Amazingly, it still worked.

The necessary design trade-offs are described, and also how the machine was built and operated. It is not intended to provide full constructional details. In any event, the machine was hand-wired; it did not use tracked circuit boards. In those days, personal CAD software (and indeed the personal computer) was unknown, and it would not have been worthwhile to lay out a set of boards by hand, for a one-off project.

Background

It all began in 1968, when I was at University in Bangor, N. Wales. The entire campus possessed just two machines (Elliott 803 and Elliott 4130), each of which filled a large room. Students punched their own programs (usually in Algol) on cards or 5-track paper tape, and submitted them to the operators. This whetted my appetite for computing.

When I left University, I lost access to a computer, and gradually developed the idea of building my own. There was a free-floating community of computer experimenters in the UK at that time. We made do with what could be found, and no two people's machines bore the least likeness to each other. Designs invariably were conditioned by what hardware we could lay our hands on. In those days, even basic 7400 logic parts were almost unobtainable by amateurs in the UK. DRAMs and EPROMs did not exist.

Simplex-I

I was fortunate enough to get hold of bits of some wrecked hardware from the early 1960's: logic boards from an old IBM mainframe, and most of a non-functioning VERDAN computer (that's a tale in itself: a military hybrid digital/anologue unit - used in SINS - Ships Inertial Navigation System). Both of these used discrete transistors, with one board roughly equivalent to an SSI package of today. There was also a disk memory unit for VERDAN: 2048 words of 26 bits each. Yes, this was a disk, with a separate head-pair for each track. The read-head was one sector's length before the write, so that a bit-serial CPU with a 1-word latency could do read-modify-writes to a single word on disk. That disk also implemented most of the VERDAN CPU registers, using a purely 1950's technique called "revolvers": a write head precedes a read head, so that the segment of track between implements a shift register. Several such registers may exist on a single disk track. The system clock was generated from a pre-recorded disk track, so that the entire system synchronised to the disk.

Design principles for my project derived largely from Booth's "Automatic Digital Calculators" - 1951 or thereabouts. Booth and his co-workers were building with vacuum tubes: the book is a mine of ideas on how to use as few gates as possible.

Having acquired a motor-generator to produce the power for the disk motor (115V, 400Hz 3-phase), I could start to run this thing up. A typical VERDAN PCB was a 8-bit shift register, so these were used unaltered. The architecture was pure 1950's: 7 serial "accumulators" (in the terminology of the time), and 2048 24-bit words of memory.

This project was later dubbed Simplex-I. It had reached the stage of having a working bit-serial arithmetic unit, when new vistas opened up.
Simplex-II

Having moved to London, I was able to obtain scrap hardware containing 7400 TTL chips. A London surplus dealer had a bin full of logic modules which he knew nothing about (and hence would sell for a song), but which I realised had been built at the place where I worked. So I had access to the manuals. Another surplus store provided a “no details” 4096 x 12-bit core memory array, and Simplex-II was under way. This was to be a virtual clone of a Digital Equipment Corporation PDP-8/S (for which I even had some software).

IO was a World War II surplus teletype machine (all of 7.5 chars/second).

Simplex-III

In late 1974, when Simplex-II was well along, I moved to Australia and the rules changed again. 7400 chips were now available at acceptable prices (via surplus dealers in the USA), and the first (1k bit) DRAM and EPROM chips were reaching the amateur market. Now I had spent my last 3 years in the UK working with the GEC 2050, at an engineering and assembly-code level, so I knew it thoroughly (including a collection of useful software techniques). Consequently Simplex-III was modelled on the 2050.

It was now 1976, and the 8080 CPU was selling at $180. I thought long and hard, whether to use an 8080 anyway, but realised the cost would be little different, and a much better learning experience to do it myself. In those days, the only minicomputer (not micro) to use a stack was the PDP-11, of which I had no experience. So I stuck with the architecture I knew. The use of stacks as a subroutine linkage device was quite rare in those days also, most stacks were used as a data-handling device in compilers, and those were routinely emulated in software.

Naturally, there was considerable input from various friends during the design phase, not least in the form of a “wish list”. Many of them saw themselves as potential users of the machine, once it was built. One of those friends was into astronomy, and indicated he would like to do a lot of maths on very large numbers. Largely for him, Simplex-III included arithmetic on integers up to 64 bits. There was never any intention to provide hardware floating-point maths.

The object was to provide much of the GEC 2050’s functionality, with far less cost. I dumped the 2050’s elaborate IO system (it had an in-built 64-channel DMA) and left everything memory mapped. The interrupt system again was vastly cut down, owing more to the Elliott 903 (another vintage machine, using discrete transistors). In those days, no-one believed you could ever run out of 64kB main memory.

Simplex-III was based around TTL/MSI logic, and 1kb DRAMs. A design goal was to fit the microcode into two 32-byte bipolar PROMs. In the event, a PROM programmer was not available, so the PROMs were simulated by a discrete diode matrix, which connected to the PROM sockets.

The machine was built in a home-made case, approximately 380 x 200 x 200mm, including power supply and space for expansion cards.

The following articles will describe the machine architecture, instruction set, debugging facilities, and finally a look at the detailed logic operations.

Ed. - Here’s a blurb about Dave’s company and the Z182 board he has available.

D-X DESIGNS PTY LTD
7 Buchan Close
SPEARWOOD
Western Australia 6163
Tel/fax: +61 9 434 4280
Email: daveb@linet.net.au
Web page: http://www.linet.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 specialty is design for Xilinx FPGAs, for which we have a full tool-set.

Z80182 8-BIT CPU BOARD (‘P112’)

What is it?

It provides a Z80182 (Z-80 upgrade) CPU with up to 1MB of memory, serial, parallel and diskette IO, and real-time clock, in a 3.5-inch drive form factor. Powered solely from 5V, it draws 150mA (nominal: not including disk drives).

1. Dimensions: 130 x 100mm (5.1 x 3.9 inch)
2. Support for 5.25 and 3.5 inch diskette drives (up to 4 drives, mixed types)
3. Z80182 CPU at 16MHz (12.288, 18.432 or 24.576MHz optional)
4. 32kB flash ROM, in-board reprogrammable
5. 64kB SRAM, upgradeable to 1MB
6. Real-time clock/RAM, with on-board battery
7. 5 (yes, five) serial IO ports, 2 as PC-AT compatible connectors, 3 as TTL outputs
8. Parallel port, IBM compatible, with bidirectional ability
9. Bus expansion/logic analyser socket
10. Software included:
    Shareware DOS+ and CCP+ (replace CP/M)
    Shareware PIP (replaces PIP)
    Shareware UUENCODE & UUDECODE
    BIOS support for diskettes, parallel & RS232 serial ports
    ROM monitor, including debugger
    Utilities:
    ASCII file-transfer (eg UUENCODEed)
    Disk format
    In-system flash-ROM reprogrammer
    Real-time clock sample code
    All source files
    All code (except the shareware items) is offered under the GNU General Public License.

The Computer Journal / #78
The TCJ Centerfold is usually for older computer stuff, right? Well, the 8051 is only a couple of years younger than the Z80. It was designed as a successor to the 8048 series. It had an 'improved' instruction set, a built-in serial port, and could address much more memory.

By the end of the eighties, many variations of the basic 8051 had been introduced. Now there's several hundred and those who can afford it can get their own versions made.

From the 8051 FAQ by Russ Hersch, some of the major manufacturers are:

- AMD: Enhanced 8051 parts (no longer producing 80x51 parts)
- Atmel: FLASH and semi-custom parts
- Dallas: Battery backed, program download, and fastest variants
- Intel: 8051 through 80c51gb / 80c51sl
- Matra: 80c154, low voltage static variants
- OKI: 80c154, mask parts
- Philips: 87C748 thru 89c588 - more variants than anyone else
- Siemens: 80c501 through 80c517a, and SIECO cores
- SMC: COM20051 with ARCNEN token bus network engine
- SSI: 80x52, 2 x HDLC variant for MODEM use

The 8051 FAQ is available on the TCJ Web pages and on the TCJ/DIBs BBS. The TCJ/DIBs BBS also has many files from the Phillips and Intel BBS's.

Intel has introduced the 8x151 and 8x251 variants also. These are supposedly code compatible with the original 8051, but there's a catch. Even with the ROM-less parts, you have to have access to an Eprom programmer that will allow you to set configuration bits before you can drop them in place of an 8051/31.

8051's and it's descendents are used by the truckload at times. I was in a telephone switching center that had an 8032 on virtually every one of the thousands of plugin cards. They were used for the low level control of each one of the line cards in the system.

The center of the Centerfold is the schematic of the amr552LC from AM Research. It's based on the 80C552, a variation of the 8051 that's produced by Phillips. Al Mitchell, the 'AM' before the 'Research', has included a special offer for TCJ readers.

On the last page of the Centerfold, is my 8031 Core schematic. I use this as the starting point for my 8031 designs. It's 4½ chips. If the internal ram is enough, you can eliminate the 6264 and the 74HC00 and use only 3 chips which I've done a couple of times. For an RS-232 interface, the MAX232 shown on Al's schematic is an excellent solution. The 8031 doesn't have any handshaking lines as such for it's serial port. If you want or need them, you have to use some of the other port pins.

<table>
<thead>
<tr>
<th>8051/31</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.0</td>
<td>1</td>
<td>40</td>
<td>VCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.1</td>
<td>2</td>
<td>39</td>
<td>P0.0 (AD0)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.2</td>
<td>3</td>
<td>38</td>
<td>P0.1 (AD1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.3</td>
<td>4</td>
<td>37</td>
<td>P0.2 (AD2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.4</td>
<td>5</td>
<td>36</td>
<td>P0.3 (AD3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.5</td>
<td>6</td>
<td>35</td>
<td>P0.4 (AD4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.6</td>
<td>7</td>
<td>34</td>
<td>P0.5 (AD5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1.7</td>
<td>8</td>
<td>33</td>
<td>P0.6 (AD6)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESET</td>
<td>9</td>
<td>32</td>
<td>P0.7 (AD7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RXD)</td>
<td>10</td>
<td>31</td>
<td>EA/VPP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(TXD)</td>
<td>11</td>
<td>30</td>
<td>ALE/PROG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(INT0)</td>
<td>12</td>
<td>29</td>
<td>PSEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(INT1)</td>
<td>13</td>
<td>28</td>
<td>P2.7 (A15)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(T0)</td>
<td>14</td>
<td>27</td>
<td>P2.6 (A14)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(T1)</td>
<td>15</td>
<td>26</td>
<td>P2.5 (A13)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(WR)</td>
<td>16</td>
<td>25</td>
<td>P2.4 (A12)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(RD)</td>
<td>17</td>
<td>24</td>
<td>P2.3 (A11)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XTAL2</td>
<td>18</td>
<td>23</td>
<td>P2.2 (A10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XTAL1</td>
<td>19</td>
<td>22</td>
<td>P2.1 (A9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(GND)</td>
<td>20</td>
<td>21</td>
<td>P2.0 (A8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The amr552LC Single Board Computer

The amr552LC SBC is one of a series based upon the venerable 8051. We elected to offer this hardware superset of the 8051 to TCJ readers to demonstrate just how much the embedded control industry has progressed since the 8051 was released in 1977. Over the years this microcontroller has consistently been our most popular SBC due to the many features and low cost.

Should anyone wish to prototype or wirewrap their own variant of the 8051 family then the following schematic will work just fine. All versions use Port 0 and 2 for external address and data buses so you can just pencil in your favorite version. We have other SBC's for most should you want to avoid the problem-prone wire-wrapping stage.

Note that every member of the 8051 family has an external pin labeled *EA or “Not External Address.” When tied low this pin tells the 8051 to look at external memory for program and data memory. When tied high program data is taken from the internal ROM if the variant has any.

The 8051 family has what is called a “Harvard” architecture. Harvard meaning that separate Program and Data spaces are used. These two distinct 64K blocks of memory are addressed by the *PSEN signal in conjunction with *RD and *WR. In most every interpretative 8051 implementation these two memory spaces are overlaid. Rather than using a myriad of jumpers we have optimized the glue logic to implement this always. This means that not only our FORTH development environment but BASIC-51/2 will also run on our boards.

Further, two memory sockets have been implemented, one for RAM and one for ROM. Either an 8K or a 32K device may be installed in either location with only one jumper required to select the RAM size. Either ROM size fits the same socket without a jumper change.

Most of the SBC failures we’ve encountered were invariably due to incorrect power connections so we decided to install the regulator to isolate the CPU from wiring errors. Further we use a 2.1 mm coaxial connector to further minimize the confusion of first getting the system working. Also included is a MAX232 communications transceiver. Some SBC’s do not use legal RS232 levels and therefore are limited in baud rate or distance they may communicate. The MAX232 uses charge pumps to generate a real ±10V from the +5V power supply. With this design our standard software can communicate at 19.2 Kbaud over reasonable distances.

The 80C552 controller chip used is made by Philips and the second-most powerful 8051 variant in existence. In addition to all the features of the 8051 it includes: two Pulse Width Modulator outputs, twice the onboard RAM, two extra 8-bit I/O ports, one of which can be a fast 8-input 10-bit A/D converter(!!!) three extra timers, one of which is a Watchdog timer, many timer/counters with invokable interrupts should you decide to use them, a timer prescaler and is very inexpensive.

The final hardware feature is not visible on the schematic, it is a 10 sq in prototyping area on our pcb ringed with both ground and +5v. +Unreg is nearby should it me required. The printed circuit board is 4” by 6”, black with white silk screen.

The editor of TCJ has severely twisted our collective arms, and since Dave is bigger and uglier than we are, we have agreed to make the bare board, a complete kit and system available for TCJ readers at reduced prices of $25/ $80/$199.

A quick word about the system above: it includes not only an assembled pcb but all memory, cables, power supply and development software. The software includes a communications package so you can speak to the sbc from your PC, an integrated assembler, high level Forth compiler, full screen text editor with multiple file cut&paste capability, a large library of useful functions such as LCD’s, keypads, editors, etc. and Frequently Asked Questions.
The burdens of time

For once, I've been able to share my burden of "lawn care" with the rest of the country. Namely, the three feet of snow we've had in February with two feet coming down in a few days! But now, at the end of February, the wind is howling and the temperature is in the mid-50's, both events most unusual. With snow still on the ground, I'm trimming bushes and thinking of sod.

For this issue, I reach again into the mail file - very little postal or phone correspondence these days - and also acknowledge some questions and some received stuff.

My regrets, but I find my new second-shift job requires more time than I have without cutting back. In addition, my S-100 correspondence has declined in the last year or two, now mostly on the Internet. To do more S100 stuff than just answer my e-mail, to assimilate some of the systems I've received in the last year, and to do more astronomy, I've decided to put my "Dr. S-100" column on vacation for a few issues. And, with the first batch of GIDE controllers now sold and initial IDE software in development, I'm turning over US sales and support to Dave Baldwin of TCJ. But maybe I'll contribute an astronomical electronics article in the future!

I will of course take email and postal mail inquiries, and offer docs and hardware for modest costs. This issue, in fact, I show you the kind of correspondence I'm doing. But I must note that the trend is lower prices on S-100 stuff, not higher. This is not necessarily good, as it discourages efforts to save and distribute. That too is a burden of time.

Recent Acquisitions, help needed:

Keith M Andress kindly donated a Big Board system and a selection of S-100 cards, in exchange for copying some 8-inch disks with software onto MS-DOS diskettes. As the 8-inch disks were single sided, single density, it was relatively simple.

The Bernoulli boxes we have here are several dual 10 Meg drive models. We however have no cartridges for them. I do not know if they are in functional order or not, as I have not had the time or inclination to mess with them much. If you care to, you are welcome to visit us on a weekday or weekend (I am at the office 24/7 it seems), and test them yourself.

I did visit and accepted the drives for a modest price. I now have several of these Bernoulli drives. However, I do not have any controllers for these, and only a few cartridges. Any one have docs, software, IBM XT controllers?

To: jjustin@m1.cambrrex.com
Date: Mon, 18 Dec 1995 21:13:00 -0500

I need documentation on the Monitor Dynamics HDC1016 hard disk controller which was an ST-506 technology card. I also need the software that went with it that interfaced the hard disk controller to CPM/80 v. 2.2. I think it was a BASIC program that invoked several .COM programs.

While I have some Monitor Dynamics cards, and perhaps a disk, I don't seem to have much documentation. I have been asked for this before, so I think I'll ask on the Internet and other sources for some docs. I'll review the disk and get back to you: figure several dollars for a 8-inch copy of the disk and postage.

To: "THE ROCHE APPROACH 1609"
<JR0CHE@FAB10.intel.com>
Subject: info on SORD CP/M 68K
Date: Tue, 16 Jan 1996 22:04:27 -0500

On Sun, 14 Jan 96 <JR0CHE@FAB10.intel.com> wrote:

*> Hi there,
*> A friend of mine was asking about cp/m68k awhile back
*> and you replied with some suggestions and info which was great.
*> In addition i was wondering if you know if there is such a thing
*> as a cp/m68k emulator for msdos and if so where could it be got.

You might ask on comp.os.cpm about this. But, I doubt there is such an emulator. Emulating CP/M is easy enough, DOS 1.0 calls will do most of that. Emulating the 68K might be problematic. You might scan the Simtel CD-ROM's or related online archives for a 68K emulator, and see if someone included hooks for CP/M.

I now have a few 68K Compumpro cards, and could someday in principle bring up CP/M 68K. Not highly likely...

Let me know what you come up with.
Maybe I’ll ask in my Mar/Apr column.

To: syoung@nucleus.com (syoung)
Subject: Re: Possible CP/M machine?
Date: Thu, 18 Jan 1996 21:33:05 -0500

You wrote:

I have run across, and subsequently acquired, a rather old machine that doesn’t work at the moment, although I think I can get it working. What I’m wondering is, does this look like a CP/M machine to you. Here is the description (the reason I wonder is because the date on the side is of the CP/M era).

It could be, maybe not. Check the boards themselves for information that is ETCHED into the board as copper: a manufacturer, model number, etc. Check the chips for date codes: 8512 would be the 12th week of 1985, that sort of thing. Describe the bus connectors: how many pins, how far apart are they (.1 inch, .125 inch, or what?). See if the INSIDE of the box or cover has any info. Let me know.

From: Eric Magnus
<eisley@sfedu.edu>
Date: Wed, 21 Feb 96 15:50:27 0000

The keyboard on my Sol 20 is dead. Well, a few of the vowels work, and some of the really good consonants, but more or less dead. My question is: was the keyboard on the Sol a Processor Technology design, or was it just a standard terminal keyboard? Could I replace it with another? Is it repairable? How could I determine what the problem is?

Sorry to be babbling like this to a complete stranger, but I would just love to get this friendly blue computer working again.

That’s why the Doctor is here.

Thank god someone understands my well, let’s call it a “problem”. :)

I have a few of these myself. The keyboard is a crappy design. It uses foam pads with aluminum foil to make the “springs” that make contact. These pads turn to powder after a de-

cade or so. If you are at all mechanically inclined you can open it up and see. I bought some foam pads of about the right size some time ago and have thought about replacing them, but I have not done so. I think I got the pads at a hobby store or plastics supplier. I would NOT advise hand-cutting out lotsa little circles!

Well I think I’ll open up the keyboard itself and take a look. I’ll let you know if I electrocute myself, and/or figure something out. Thanks!

Typical correspondence

To: hjohnson@pluto.njcc.com
From: “Roy J. Tellason”
tanstaaf@postoffice.ptd.net
Subject: Re: S-100

One of these systems is a Cromemco System/3, which used some of the bus lines as individual select lines, rather a weird arrangement when you can deal with a whole lot more ram in the system by decoding these lines on board, but hey. I don’t know what they were thinking when they designed the board... <g>

Well, 64K * 8 was a fair amount of memory at one time...

Yeah. They used eight lines to select one of eight possible 64k boards, which I guess is all they thought that people would ever really need...

I remember at around the time I got that system Cromemco was pretty much into making only 68000-based systems to be used in industrial control and similar applications. Do you know anything about what ever happened to them? One of these days I’d like to begin building a chronology of some of these companies...

As of the early 1990’s they were still around. Do me a favor, and I’ll put your response in the next TCJ. Check out Cromemco for me under their new name:

Dynatech Computer Systems
280 Bernardo Ave
PO Box 7400
Mountain View CA 94039-7400
415-964-7400

The other box is my Imsai, which doesn’t have the original boards in it. I got two rather nifty Teletek manuals with the system, one of which describes a rather nifty cpu board that’s pretty complete unto itself, but it looks like I have the other one... Not sure which ram board I have in that system, and I don’t think I have docs in it.

Tell me what board and manual you have: maybe I have the others!

I don’t know what the ram board is that’s in it, and I can’t get into it at this point in time. The machine is sitting at the back end of a table with about a foot of space in front of it, and the terminal on top of it. (BTW, that terminal is an ADDS Viewpoint — do you have anything on that? I’d like to get WS looking right!) The two manuals I have are for a Teletek “Systemmaster” (there are some spec sheets in there for their other products) and the FDC-I, which I believe is the board that’s in it at the moment. Plus some sort of a 64k ram board but I can’t remember any specifics on that one offhand. The one I don’t have appears to need other boards for io but has its own ram on it, so there’s the tradeoff...

I don’t have much in the way of original docs for the insaion setup, and a bunch of the front panel doesn’t do anything any more since the board that ran that was apparently removed at some point in time. I recall a couple of ribbon cables with IC-type plugs on the end of them, and would like to hook these up to something so I can use those switches later on. The lights work, though... <g>

The ribbon cable connects the front panel to the data lines of the processor. It’s necessary for any front panel switch activity.

Oh, on the subject of $100 stuff, I have a Vector motherboard that I got somewhere that would make a dandy setup for prototyping stuff. It has room for something like 6-8 slots. Problem is, I started with a blank board. This is really pretty nifty, as it has active termination on it. What I’d need to get that going is a few more parts (not a
problem) and s100 connectors (problem). Do you have, or know of a source for those connectors?

They are available from JDR Microdevices, sales phone # 1-800-538-5000 100 pin S-100 edge connector, .125 centers between pins.

Solder tab version: 100P-ST, $4.95
Wire Wrap version: 100P-WW, $5.95

From: "Ken Montgomery"
<KENM@comptr.ccs.csus.edu>
Date: Mon, 22 Jan 1996 15:50:32 PST
Subject: Re: Exity Sorcerer's etc..

As I write articles on the S-100 for The Computer Journal, I'd appreciate a bit more info on the Sorcerer for publication in my column. I've seen a few of these in my time, but I don't have one. I'd be interested in a manual set, if that is a reasonable thing to put together, and some kind of bootable disk (or tape) if I have to offer one to somebody in the future. Your recent post in comp.os.cpm was pretty good: if you would care to expand it a bit I'd be glad to publish it!

I have 7 of them. There were several flavors of the Sorcerer. These included at least 3 revisions of the board, plus a 'Spellbinder' version and others. Yes, they used hollowed out 8-track carts for their ROM boards!

My 'archive' of Sorcerer stuff is VERY buried at this time, but I of my current projects is to pull out each and every one of the systems in my collection and photograph them. I'm working on building a virtual museum which I'll put up on the WWW. With the amount of information I have and the quantity of systems, it should be quite an overwhelming site! At this stage, I'm working on a general description/history of each of the types of systems (a paragraph size amount) and a description/history of that specific system(s), like who I got it from if I didn't buy it new, and what is was used for. I haven't seen anything on the net yet that has more than a half dozen systems listed, and I have over 40 unique types. I expect to put it up after I've got somewhere around a third of it done.

I'm an old SMUG member so Bill Kibler can tell you all about me. He knows I've been planning this for some time now. I'll keep you posted on my progress.

>From hjohnson Tue Jan 23 14:13:55 1996
To: "N. White" <normill@maine.net>
Subject: Repair of CP/M System 8" Drive and/or Floppy Controller Card
Date: Tue, 23 Jan 1996 13:31:33 -0500

On Mon, 22 Jan 1996 01:16:43, "N. White" <normill@maine.net> wrote:

I am trying to resuscitate a S-100 bus, CP/M system, which ceased functioning suddenly about eight years ago. Running monitor diagnostics shows the CPU card and 64K RAM card to be functioning perfectly. The problem seems to be with either the 8 inch drive, or the floppy controller card. Diagnostics on the latter show that I can move the head around o.k., but can not READ or WRITE to the disk. I have tried replacing all the TTL chips on the controller card, replacing the ribbon cable, and even swapping in an equally dubious spare 8 in. drive, but nothing works. All three cards were made by a company called 'SDK', based in Dallas, Texas. Any ideas?

SDK sounds kinda familiar. Do you have names for these boards? Docs?

IF you have an oscilloscope, try looking at the disk drive READ signal. You should have no signal until the head is loaded (touching) onto the diskette AND there is enough pressure between the head, diskette, and the head or pad on the other side. THEN, you'll see a stream of highs and lows. Even a logic probe will show the difference. Borrow one of these if necessary.

Replacing the floppy controller chip seems like the next course of action. Some older controllers used logic and even analog chips for disk reading: replacing these without retuning the circuits may cause loss of data.

I have 8-inch drives which I can verify before shipment. Cost: around $25 plus shipping, depending on size (full heights are cheaper) or single or double sided (SS are cheaper). What drives do you have?

To: sporkster@cris.com (Sporkster)
From: hjjohnson@pluto.njcc.com
( Herbert R Johnson)
Subject: Re: What do you do with it??
Date: Sat, 27 Jan 1996 19:28:43 -0500

In article , you wrote:
> In article <>, egriebel@ptid12 says...
> I love playing around with old systems and getting them working.
> but I never know what to do with them once I get them working
> (as if I had any time, anyway :) 
> >
> I'm curious—what do you all do with your old systems?
> >
> Use It as a door stop, paper weight, target practice, see what 600 volts will do to a motherboard, ash tray, anchor,
>
> People consider small systems a challenge, or use them as part of development for various hardware (read non-PC) applications, or just as a reminder of previous work. And some people respect such folks: and some don't.

To: David Saad
<dsaad@ntest.teradyne.com>
Date: Thu, 01 Feb 1996 12:46:47 -0500

I got your name from an article in the news group comp.os.cpm. I have a Northstar Horizon 64k with two single sided 160k hard sector floppy. I came across a compumore 8", 5 1/4" floppy controller but have no software to go with it. The controller rev is 171F, and I have the manual.

Is this a Disk 1 or a Disk 1A or a Disk 1B?

I have made the modifications specified in the manual for use with a 5 1/4" drive, and written some routines in turbo pascal to format the disk, but cannot get the format to complete suc-
cessfully. Do you have any experience with this hardware combination? Or better yet do you know where I could find the Compupro software that came with the board?

I have CP/M 80 and CP/M 8/16 and CP/M 86 from Compupro for this controller. Cost for CP/M 80 is $30 plus shipping, which includes a CP/M 80 serialized diskette (i.e. license). The disk is on 8-inch media. I don’t have a 5.25 inch boot disk, check the manual (and I’ll review the disk) to see how you might support 5.25 inch.

My impression is that you want to use this controller for the Northstar. In that case you have to rewrite any software from the Compupro diskette. If so, you may only want sources on diskette. I can put them on a MS-DOS disk that you could serially transfer to the Northstar via a PC compatible. That would cost $25 plus $5 shipping. 3.5 inch 1.44 Meg would be the format, but I could do others.

I have 8-inch floppy drives for various prices around $25, depending on size (half or full height) or heads (single or double). Also cases and power supplies, etc. for similar prices. Shipping extra.

I also have some SD Systems floppy disk controllers with some software intended for use in a foreign system. The software includes format and some BIOS samples, but the project is incomplete. I’d sell you a *new* card plus the software as it is for $25 plus shipping. The card supports 5.25 AND 8-inch at the same time. What would you like?

>From JRBRADY@delphi.com
Mon Jan 15 21:36:54 1996

I wanted to touch base with you regarding purchasing some S-100 equipment. Here’s what I have so far:

2 - Northstar Horizons with SSDD 5-1/4" hard-sectored floppies (the ones I brought back to life) along with extra N* Z-80 CPU cards, memory boards, and disk controllers
1 - PSS RAM 65 board (no documentation)
2 - PSS RAM 16 boards
2 - SCP 16K Plus boards
1 - SCP 16K Std. board
1 - Morrow Disk Jockey 2D/B board, no docs, one bad regulator!

OK, these are reasonably nice systems, notwithstanding hard sectoring...

I have docs and disks for some versions of the DJ 2D. Regulators are cheap.

I hope you did not fry the card otherwise!

In addition to plain-vanilla N* DOS, various versions of CP/M 2.2 and probably 100+ disks came with these systems (however, most of the application software is for the N* Advantage machine which uses the DSDD “quad” hard-sectored floppies).

What I’d like to do is build a more flexible system that will support soft-sector 5-1/4" and/or 8" disks, a hard disk, additional I/O ports, and multiple 64K banks. What do you recommend? Do you feel that the N* backplane will suffice for this project? Can you use any of the boards listed above?

I don’t need any of the above, other than a mild interest in a Northstar system again. I can sell you a reasonably flexible softsectored controller from SD systems with a generic BIOS. Or/and docs for your Morrow DJ2D-B, but that’s only 8-inch. The SD Versaflippy II with a preliminary 5-inch BIOS and docs is $30 plus shipping. It is I/O mapped, unlike the memory mapped N* stuff. I’d be kinda interested in the Morrow card, maybe you could photocopy the board and send a picture so I can figure out which version you have.

Your hard sectored disks have some incremental value: they are hard to come by these days.

On Mon, 12 Feb 1996 <JRBRADY@delphi.com> wrote:

* > Hi Herbert,
*>
*> Just wanted to drop a note and inquire if you received the letter with the *lousy photocopy of a Morrow controller board. Any comments?

Yes, sorry for the delay - snow, new job, illness.

I have a number of these cards and find them very convenient, so I’d be interested in one. I recall it was not working, at least due to a bad regulator.

I have some new SD Systems card that are good for both 8-inch and 5-inch. I have their BIOS source, which requires an SD CPU card; and some BIOS work done by Raymond Gandia to support a 3.5 inch drive, also in source form. I can send you a card, docs, and this code. Normally this would be $30 plus shipping: how about a $10 plus shipping credit for the DJ card, so your cost is $20? The only issues about the SD card is 1) you need to mount an inverter on the card to change its phase1 clock input to phase 2; and 2) it has no ROM sockets, so your BIOS or boot code must be elsewhere.

I’d reimburse your $20 IF you came up with a good 5.25 inch BIOS system for the card and FORMAT, and gave me the sources. The good news is that the SD card is I/O mapped, so you don’t have to worry about memory conflicts. The DJ card is MEMORY mapped up at the top of memory where the Northstar I/O is!!!

---

S-100/IEEE-696

IMSAI Altair
Compupro Morrow
Cromemco
and more

Cards * Docs * Systems

Dr. S-100

Herb Johnson,
59 Main Blvd.
Ewing, NJ 08618
(609) 771-1503
hjohnson@pluto.njcc.com

---

The Computer Journal / #78
Storage Classes And Scope of Variables

So far we haven’t discussed the scope of variables in C. Here is some real code to look at:

```c
/* scope of variables */
#include <stdio.h>

int x,y,z; // these are GLOBAL variables
void main()
{
    int a,b,c; // these are local to main and
               // hidden from other functions
    ....
}

void do_something()
{
    int p,q,r; // these are local to this function.
               // They don’t even exist when this
               // function is not running
    ....
}

void do_something_else()
{
    int x; // this function now can not access the
            // global variable x. If x is used in an
            // expression it will use the local
            // variable
    ....
}

void count_events()
{
    static int count=0;
    count++;
}
```

That is enough to talk about for a while. Variables declared outside of any function including `main()` are global variables. They may be used by any function in the program, with limitations described a few paragraphs down.

Variables declared inside a function including `main()` may be used only by that function. A variable declared inside a function, that has the same name as a global variable, blocks access to that global variable within the function.

All of the variables defined in the above examples are called "automatic" variables. They come into existence when the program block in which they are declared is running. That is, the variables `p`, `q`, and `r in do_something()` are allocated space when `do_something()` is called. No assumptions can be made about the contents of `p`, `q`, or `r`. They may be the same as they were left on the last call, but if `do_something` was arrived at by a different path, they may be allocated space in a different place in memory and will contain no meaningful values.

The function `count()` has had it’s variable declaration modified by the addition of the keyword "static". Static means that the variable will be allocated a permanent location in memory. A static variable is initialized only on the first call to the function, and it remains valid the next time the function is called. Count() can therefore keep track of a count of something. It is legal to use the modifier "auto" in front of variable declarations, but it is the default and leaving it out saves a lot of typing.

Static variables are used for another purpose. A C program may consist of multiple modules or files. When a global variable is declared as static it is not accessible by another program module. When a module needs to access a variable in another module (file) it declares that variable as "extern" (for external). Again it is too bad the authors of C couldn’t have used a different word (local or private, for example) rather than giving the word static two different meanings depending on the context or location where they are used.

```c
// program .......
static int x; // this one is not accessible by
              // another module
int y; // this one is accessible by another
       // module
extern int z; // this one is in another module (or
              // there will be an error when the
              // compiler tries to find it.
```

We will talk about multiple module programs in another session.

Files

C has provision for handling files for read and write. There are functions to provide this access:

```c
FILE* fopen(char* filename, char* mode);
fclose(FILE*);
```
Most of the C documentation is done as these are shown here. That is, the types of the return value and of the parameters are shown without any variable names associated. This information is available in the HELP file of Turbo C and it is generally how any description of the standard library functions is written in textbooks on the same subject. C has a predeclared (in a header file) data type called a file pointer. You first define one or more of these and then use them as follows:

```c
FILE *infile, *outfile;
main()
{
    int ch;
    infile = fopen("data.dat", "r");
    outfile = fopen("new.dat", "w");
    while((ch = getc(infile)) != EOF) {
        putc(ch, outfile);
    }
    fclose(infile);
    fclose(outfile);
}
```

The function fopen() returns the file pointer type that needs to be assigned to the variable infile and outfile. If things go well the value returned is a pointer to an area of memory (defined as a structure in C) where the file is handled. If things do not go well fopen() returns a value of zero or NULL. You can use this fact to trap errors for the user and tell him what he did wrong or what didn't happen according to plan.

There are other modes than read and write. A file that exists can be opened for "append" or "update" and it can be opened in "binary mode". A text file that is going to be read into a buffer ought to be opened in just plain read mode ("r"). If it is to be copied to another file as in the above example it ought to be opened in binary mode as in "rb" and "wb". The difference is small, dealing only with how C treats the end of line character(s). Dos uses a CR and an LF at the end of each text line. C converts this to however it represents \
(generally just an LF) when it reads a file in non-binary mode. It converts the \n back to CR and LF when it writes a file in non-binary mode. In binary mode it doesn't change the value of any byte of the file, just what we want when we copy a program from one file to another.

The filenames in fopen can just as well be a character array, since they are defined as char*:

```c
char filename[];
infile = fopen(filename, "r");
```

**Command Line Arguments**

C has a nice feature. It is a built-in way to handle command line arguments. Below is a program listing that uses this feature. To use it we define some parameters for main():

```c
main(int argc, char* argv[])
{
    int ch;
    infile = fopen(argv[1], "rb");
    outfile = fopen(argv[2], "wb");
    while ((ch = getc(infile)) != EOF) {
        putc(ch, outfile);
    }
    fclose(infile);
    fclose(outfile);
}
```

Now you have all the pieces of a genuine copy program. The command line is:

```c
ccopy data.dat new.dat
```

Assuming "ccopy" is the name of our program, and we want to copy an existing file "data.dat" to a non-existing one "new.dat". Now let's look at 'argc' and 'argv'. 'Argc' is just the count of arguments on the command line (items separated by spaces) In this case there are three including the name of the program. I.e. "ccopy" is the first, "data.dat" the second, and "new.dat" the third.

'Argv' is declared as an array of pointers to character. or if you like an array of character arrays. 'argv[0]' points to "ccopy", 'argv[1]'' points to "data.dat" and 'argv[2]' points to "new.dat". We didn't use argc in our program but we could test for user errors by expanding our program as follows:

```c
// a program to copy a file to another
// syntax: ccopy in_filename out_filename

#include <stdio.h>

void main(int argc, char **argv)
{
    FILE *infile, *outfile;
    int ch;
    if(argc != 3)
    {
        puts("ccopy needs input/output filenames");
        exit(1);
    }
    infile = fopen(argv[1], "rb");
    if(infile == NULL)
    {
        puts("can't open input file");
        exit(1);
    }
    outfile = fopen(argv[2], "wb");
    if(outfile == NULL)
    {
        puts("can't open output file");
        exit(1);
    }
    ch = getc(infile);
    while(!feof(infile))
    {
        putc(ch, outfile);
        ch = getc(infile);
    }
    fclose(infile);
    fclose(outfile);
    exit(0);
}
```
This is a complete working program written in the style of Pascal or other procedural languages such as PL/9. The next listing will present the program with the C shortcuts, some of which we have mentioned above. Basically the shortcuts are done by including an assignment statement inside of a while condition. Usually doing this can eliminate having to repeat a statement such as ch = getc(infile); If you compare these two programs you will see the economy of the shortcuts (and also how they make the program appear more cryptic and harder to understand).

// program to copy a file to another
//syntax:   ccopy filename outfile filename
#include <stdio.h>

void main(int argc, char* argv[]) {
  FILE *infile, *outfile;
  int ch;
  if(argc != 3) {
    printf("ccopy needs input/output files!\n");
    exit(1);
  }
  if ((infile = fopen(argv[1],"rb")) == NULL) {
    printf("can't open input file\n");
    exit(1);
  }
  if((outfile = fopen(argv[2],"wb")) == NULL) {
    printf("can't open output file\n");
    exit(1);
  }
  while((ch = getc(infile)) != EOF) {
    putc(ch,outfile);
  }
  fclose(infile);
  fclose(outfile);
  exit(0);
}

These are both complete working programs that can copy a file to another file. The args can just as well be complete directory paths. Note however that we have done nothing to implement the "wildcard" features of the DOS copy program. That can be done but the program becomes much more complex. We would have to open directory records and search names of files in the directory for valid string matches somehow handling the wildcard characters "*" and "?".

These programs can be the basis for what we call a "filter program", one that reads information from one file and changes it slightly, writing it to an output file. All we have to do is insert some sort of process between the read and the write. For example, we might for some reason want to convert a text to all upper case or all lower case. The while loop in the program could be changed to:

while ((ch = getc(infile)) != EOF) {
  ch = toupper(ch);
  putc(ch,outfile);
}

Assembler

Well, so much for scope of variables and working with files. Let's get on to Assembler Programming. This time I found a program I wrote some time ago as a utility for FLEX. It allows you to examine the data on a disk by sector number. If you are going to write utility programs this is a nice tool to see exactly what was written to the disk when you wrote a file. It also allows you to look at the structure of a FLEX disk. For example the first two sectors of track 0 (0000 and 0001) contain the boot information (if the disk is a bootable one). Sector 0003 contains the name and date of the disk. Sector 0004 is blank and the directory starts at sector 0005. The remainder of track 0 is reserved for directory entries. User files start on track 1 sector 1. In the notation I am using here, the first two digits are the track and the second two the sector. Numbers are hexadecimal.

In FLEX, track 0 has sector numbers starting at 0. All other tracks start at sector 01. Seems strange, but it has to do with the way disk controllers work and being compatible with IBM format standards. The number of sectors per track depend on the disk format (and in some cases on the version of FLEX). When you dump a sector, the first two bytes that are displayed are the link to the next sector in a chain. If you are in the middle of a disk file, the link points to the next sector in the file. If you are at the end of the file the sector link is 00 00. The next two bytes contain the sector count within the file. The remaining 252 bytes of a sector are data. The program dumps a sector in both hexadecimal and ASCII representation. At the left you will see the hexadecimal such as 41 42 43 ... and at the right, the ASCII representation of the same codes, in this case A B C .... Some codes don't represent printable characters, i.e. the first 32 codes, which are control codes. SDUMP represents these as a period. Codes larger than $7F have the high order bit removed in the ASCII representation. Therefore $A1 and $C1 are both shown as ASCII A.

We need to talk a little more about the File Control Block in FLEX. If you will refer to your programmer's guide (assuming you have one), you will see that the first 64 bytes of the FCB are used for housekeeping and the last 256 hold the image of the currently in use disk sector. Bytes 30 and 31 of the FCB contain the number of the current track and sector respectively. The numbers are in binary form of course so we will treat them as hexadecimal values for purposes of specifying them. Also remember that the first byte of the FCB is byte 0, the 64th byte therefore being 63. The sector image starts at byte 64.

Now let's look at the program, which uses just a few more FLEX routines than we have used previously. This program has been written in position independent code. The command syntax and a brief description of the utility are included as comments in the Assembler source code, a good
practice for something that you are going to use for a while or give away to someone else.

NAME SDUMP
TTL DISK SECTOR DUMP UTILITY
OPT PAG
PAG
* * DISK SECTOR DUMP PROGRAM
* * FORMAT: SDUMP N WHERE N IS THE DRIVE NUMBER
* * SDUMP WILL PROMPT COMMAND:
* * * * * VALID COMMANDS:
* * N TSS NEXT TRACK (TT) AND SECTOR (SS) TO BE
* * DUMPED
* * F FORWARD A SECTOR VIA INCREMENT OF SECTOR NUM
* * B BACK A SECTOR BY SAME MECHANISM. N COMMAND MUST
* * BE USED TO CHANGE TRACKS, AN INVALID SECTOR
* * NUMBER WILL RESULT IN DUMPING THE PREVIOUSLY
* * DUMPED SECTOR AGAIN.
* * * FLEX EQUATES FOR THIS PROGRAM
GETCH EQU $C0D5 GET CHARACTER
GETNAME EQU $C020 GET FILE SPEC INTO FCB (X)
DEFEXT EQU $C033 DEFAULT EXTENSION
FMS EQU $D046 WITH FUNCTION CODE AT 0.X
* * 9 = READ SECTOR
* * TRACK IS 30.X SECTOR IS 31.X
HEXIN EQU $C0D2 GET HEX VALUE IN X
OUT2H EQU $C0DC OUTPUT 2 HEX DIGITS BYTE POINTED
PCRLF EQU $C0D4 PRINT CARRIAGE RETURN LINES
STDOUT EQU $C0DE PRINT STRING POINTED AT BY X
PUTCH EQU $C0DB OUTPUT CHAR IN ACCA TO TERMINAL
WARMST EQU $C0D3 FLEX WARMSTART
* * DEFINITIONS FOR BYTE LOCATIONS IN THE FCB
* * FUNC EQU 0
ERRCND EQU 1
DRIVE EQU 3
TRACK EQU 30
SECTOR EQU 31
* * ORG $C100
START BRA BEGIN
VER FCB 1 VERSION NUMBER
* * PROGRAM VARIABLES
* * NUMBUF RMB 2
LINCNT RMB 1
CHRNT RMB 1
PAGEAD RMB 1
XTEMP RMB 2
* BEGIN JSR HEXIN GET DRIVE NUMBER FROM COMMAND
STX NUMBUF,PCR
STARTI LEAX FCB,PCR POINT AT FCB
LDA NUMBUF+1,PCR DRIVE NUMBER
STA DRIVE,X
LEAX PROMPT,PCR PRINT THE PROMPT
JSR PSTRNG COMMANDS ARE A SINGLE
JSR GETCH CHARACTER
LBSR TOUPPR FORCE IT TO UPPER CASE
LEAX FCB,PCR
* * THIS IS THE COMMAND INTERPRETER
* * EACH COMMAND LETTER CAUSES A BRANCH TO THE
* * APPROPRIATE ACTION
* * CMPA #N
BEQ NEWTS
CMPA #F
BEQ NEXTS
CMPA #B
BEQ LASTS
CMPA #E
BNE START1 IF COMMAND DOESN'T MATCH, START OVER
AGAIN
JMP WARMST IF E WE GET TO HERE AND EXIT
NEWTS LBSR GETTS NEW TRACK AND SECTOR
BSR OPAGE SHOW THE SECTOR
BRA START1 GO AROUND AGAIN
NEXTS LDO TRACK,X NEXT TRACK AND SECTOR
ADD $1
STD TRACK,X
BSR OPAGE
BRA START1
LASTS LDO TRACK,X PREVIOUS TRACK AND SECTOR
SUBB $1
STD TRACK,X
BSR OPAGE
BRA START1
* * OUTPUT A PAGE IN HEX AND ASCII
* * OPAGE JSR PCLRF
LEAX FCB,PCR POINT AT FCB
LEAX TRACK,X GET TRACK AND SECTOR ID
LBSR OUT4H OUTPUT IT TO THE SCREEN IN HEX
LEAX HEADER,PCR POINT AT HEADER
JSR PSTRNG PRINT IT
JSR PCLRF
LEAX FCB,PCR
LDB #9 FUNCTION CODE FOR READ SECTOR
STD FUNC,X
JSR FMS GET THE SECTOR DATA
JSR PCLRF
* * CLEAR LINE AND PAGE ADDRESS COUNTS
* * CLR LINES COUNTER
STD LINCNT,PCR
STD PAGEAD,PCR
STD PAGEAD+1,PCR
LEAX FCB,PCR
LEAX 64,X 64TH FCB LOCATION IS START OF
SECTOR DATA
STX XTEMP,PCR THIS IS A WAY TO SAVE A VALUE
FOR LATER USE
* THIS LOOP FOR EACH LINE
LLOOP LDB #16 CHAR COUNTER THE LINE LOOP
STA CHRCNT,PCR
LEAX PAGEAD,PCR PAGE ADDRESS
JSR OUT2H OUTPUT TO SCREEN
LDA #920 SPACE
JSR PUTCH
LDA #920 JSR PUTCH
LDA #920 JSR PUTCH
LDB XTEMP,PCR GET THE VALUE SAVED TEN LINES OR
SO ABOVE
* CHARACTER OR BYTE LOOP 16 BYTES PER LINE
CLOOP JSR OUT2H OUTPUT HEX VALUE
LEAX 1,X INCREMENT POINTER
LDA #920 SPACE
JSR PUTCH
LDA #920 JSR PUTCH
LDA #920 JSR CLOOP LOOP UNTIL CHRCNT = 0
LDB XTEMP,PCR SPACE
JSR PUTCH
* NOW BACK UP AND OUTPUT THE ASCII FORM
LEAX -16,X 
LDB #16 
STB CHRCNT,PCR 
* ASCII LOOP
ALOOP LDA #X 
ANDA #$7F 
CMPA #$20 
BGE A1I 
LDA #' 
ALI JSR PUTCH 
LDB CHRCNT,PCR 
SUBB #1 
STB CHRCNT,PCR 
BNE ALOOP 
* SAVE THE ADDRESS WITHIN THE SECTOR
SIX XTEMP 
* INCREMENT THE PAGE ADDRESS
LDA PAGEAD,PCR 
ADD #16 
STA PAGEAD,PCR 
JSR PCLRF 
LDB LINCNT,PCR 
ADD #1 
STB LINCNT,PCR 
CMPB #16 
BNE LLOOP 
END OF MAIN LOOP TO PRINT 16 LINES 
RTS 
* 
* SUBROUTINE TO GET A NEW TRACK AND SECTOR
* 
* GETTS LDB #4 
STB CHRCNT,PCR 
LEAX FCB,PCR 
LEAX TRACK,X 
CLR #X 
CLR #X 
LOOP JSR GETCH 
JSR TOUPPR 
* NOW CONVERT TO HEX
CMPA #"O 
BLT ERR 
CMPA #"9 
BGT GETPGI 
SUBA #$30 
BRA SHFT 
GETPGI CMPA #"A 
BLT ERR 
CMPA #"F 
BGT ERR 
SUBA #$37 
SHFT PSHS A 
CLRA 
* GET THE HEX NUMBER ASSEMBLED SO FAR AND ADD THE 
* NEW MYPBBLE TO IT. THEN SHIFT ALL LEFT 4 PLACES 
ADDX,X 
DECHCHRCNT,PCR 
BEQ DONGET 
ASLB 
ROLA 
ASLB 
ROLA 
ASLB 
ROLA 
ASLB 
STDX,X 
BACK UP 16 BYTES 
LOAD THE COUNTER 
GET A BYTE 
MASK HI ORDER BIT 
IF NOT PRINTABLE SUBSTITUTE A PERIOD 
PUT IT TO THE TERMINAL 
DECREMENT COUNT 
GO AROUND UNTIL ZERO 
FIRST COLUMN OF OUTPUT 
BUMP BY 10 HEX 
GO TO A NEW LINE 
COUNT THE LINE JUST OUTPUT 
WE'RE GOING TO READ FOUR CHAR 
POINT X AT TRACK,SECTOR BYTES 
IF IN RANGE 0 TO 9 ASCII 
SUBTRACT $30 
IF IN RANGE A - F ASCII 
SUBTRACT $37 
IF NOT A VALID HEX DIGIT ERROR 
CONVERTS ASCII A-F TO HEX A-F 
THIS IS A WAY TO TRANSFER 
CONTENTS OF A TO B PULS B 
SAVE THE SHIFTED VALUE BACK IN TTSS

BRA LOOP 
DONGET STD,X 
RTS 
* OUTPUT 4 HEX DIGITS BY CALLING FLEX OUT2H, 
* INCREMENTING THE POINTER (X) AND CALLING IT AGAIN 
OUT4H JSR PCLRF 
JSR OUT2H 
LEAX 1,X 
JSR OUT2H 
LEAX -1,X 
RTS 
ERR LEAX MSEG,PCR 
JSR PSTRNG 
RTS 
* 
* SUBROUTINE TOUPPR 
* WE DID THE INVERSE LAST TIME 
TOUPPR CMPA #"a IS WITHIN a-z ? 
BMT DONUP 
CMPA #"z 
BGT DONUP 
SUBA #$20 ELSE MAKE IT A-Z 
DONUP RTS 
MESEG FCC /INVALID NUMBER/,504 
PROMPT FCC /COMMAND:/,504 
HEADER FCC / 
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 
0C 0D 0E 0F/,504 
FCB RMB 320 
END START 

Notice that we have put some of the variables at the beginning of the program and the constant text strings and the File Control Block at the end. It is a common practice in FLEX utilities to place a version identification number just after the initial branch to the start of the program. There is a FLEX utility called VER (or version) that will report the version number of a .CMD file. Thus VER SDUMP.CMD would report 1.

Rather than a line description this time, I've tried to comment the code sufficiently so you can see what is going on. If you are successful at typing in and assembling this you should see something like this:

0101 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 00 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 0C 51 ETC. 0

The leftmost number in each line is what was called PAGEAD in the program.

This pattern will continue for 16 lines. You read the byte address within the sector as the number at the left of the line plus the number on top. Thus the 12th byte in the 5th line would be byte address 40 at the left plus 0B from above or byte 4B.

We did an upper to lower case conversion last time so the TOUPPER subroutine ought to be straightforward enough.
The hardest part of this to follow is the GETTS subroutine. GETTS gets a character from the user's keyboard. If it is in the range of ASCII 0 to 9 ($30 to $39) we simply subtract $30 from it to get the hexadecimal value 00 through 09. If it is within the range of ASCII A through F we subtract $37 to get the hexadecimal values 0A through 0F. For example, A has the ASCII code $41 hex. If we subtract $37 from $41 we have a result of $0A etc. Remember this is hexadecimal arithmetic. We are not subtracting decimal 37 from decimal 41 which of course would give us a result of 4.

If the character is outside of both ranges it is an error which we trap.

Having the first digit of our desired TTSS we add it to the contents of the Track and Sector bytes which we cleared at the start of the routine and shift it left four bits. For example, we input the character 2 and we add $02 to the cleared value and shift it left four places. TTSS then contains $0020. Now we input 4, add it to TTSS yielding $0024 and shift that left to yield $0240. Get a third character, say 1 and add it yielding $0241, shift that left yielding $2410, get the 4th character, say A, add it to get $241A, see that the count is 0 and exit. We are now set up to read track $24 (decimal 36) sector $1A (decimal 26). Assuming these are valid track and sector numbers for the disk format, we then read that sector.

This time I haven't broken the program into little pieces for explanation, but I have commented rather liberally. The only hard part is the double loop, actually two loops inside of a larger one. The structure is something like this in pseudo code:

```
while we haven't printed 16 lines
begin
  while we haven't printed 16 hex values
  begin
    print a hex value
    count it
  end
  back up 16 bytes
  while we haven't printed 16 ASCII values
  begin
    print an ASCII value
    count it
  end
  increment line count
end
```

Note that in assembler it is better to preload a counter variable with the desired count and decrement it, testing for zero with a BNE instruction. This is much more efficient than counting up and having to do a CMPA #16 followed by a BNE instruction. That is, the BNE is an automatic compare with zero. The code is smaller and it runs faster.

Let me continue this discussion with a bit on the trade off between execution speed and code size. Suppose you want to write a subroutine to output n characters (all the same) and return. If you want to save space you might write a subroutine:

```assembly
CHARS  JSR PUTCHR
       DECB
       BNE CHARS
       RTS
```

In the program you would use CHARs:

```assembly
LDA *  
LDB #12
BSR CHARs
```

CHARs would output 12 "stars" and return.

If you were interested in speed and were certain you would never need more than, say, 15 characters output, you could use:

```
CHAR15  JSR PUTCHR
CHAR14  JSR PUTCHR
CHAR13  JSR PUTCHR
CHAR12  JSR PUTCHR
CHAR11  JSR PUTCHR
CHAR10  JSR PUTCHR
CHAR9   JSR PUTCHR
CHAR8   JSR PUTCHR
CHAR7   JSR PUTCHR
CHAR6   JSR PUTCHR
CHAR5   JSR PUTCHR
CHAR4   JSR PUTCHR
CHAR3   JSR PUTCHR
CHAR2   JSR PUTCHR
CHAR1   JSR PUTCHR
       RTS
```

You've figured this out already, but you output ten characters by loading A with the character and doing a BSR CHAR10. You enter at whatever label and fall through the lower labels to the RTS. This comes in the "cute trick" category and there are more complex variations of it such as entering in the middle of a loop and going around to code before the entry point. Generally cute tricks are hard for someone else to understand and ought to be avoided. Some of these are holdovers from the days when every byte of memory was dear (when we had 4K to work with rather than 4 meg), or when processing was so slow that we resorted to tricks to speed up the execution.

This is a dumb example of course since you wouldn't be worried about speed of execution if you were simply outputting characters to a terminal. Another way to reduce both execution time and code is to avoid doing anything inside a loop that only needs doing once outside of it. In this example we have the loading of the character to be output. PUTCHR in FLEX requires the character to be in the A accumulator, and the routine doesn't change the contents of A, so we need only load it once. I've also used the trick mentioned above of reloading the B accumulator with the count and decrementing it so we can test for 0 rather than comparing with a value stored elsewhere.
Some of you may wonder about "Startup Code"? Well, you do have to start somewhere. Actually, every system requires 'startup' or 'boot' code to set up the initial operating conditions. And, it turns out that some microcontrollers require a certain startup sequence. This article and the program that goes with it is a result of the problems I had the first time I had to use the 8031, the rom-less version of the 8051.

The Problem

I put the prototype board together and burned a simple test program I got from the Phillips BBS into an eromp. I chose the program because it was supposed to exercise the serial and timer interrupts and I was planning on using both in my design. I plugged the chips in and turned on the power expecting to see the serial port spit out the test messages. Nothing happened. I checked the board, found the errors, fixed some errors, and tried again. Still nothing. I dug out an 8031 board I had that was known to work, plugged the eromp and the 8031 into it and tried again. Nothing again.

I was using the CMOS Intel 80C31BH in the design. I found an NMOS 8031 and put that in there and it appeared to mostly work. I tried the rest of the (brand new) CMOS samples I had and they didn't. Since I had designed the board for the CMOS version, this seemed really odd. A friend gave me a monitor program that didn't use interrupts and I tried that. It worked on all the versions of the 8031 that I had, both NMOS and CMOS. That means my problem was in the interrupt code AND that there was something different between the NMOS and CMOS versions.

```plaintext
; DIBs 80x31/51 skeleton for interrupt driven serial comm programs
; for the MetaLink assembler. Works on 8032 and should work on most
; other variations.
; By Dave Baldwin, May 1996

; Program does not expect or use external ram.
; The 80x31/51 begins execution at 0000h but has to jump immediately
; since location 00h is the beginning of the interrupt vectors

; $MODE1 8051 definition table for MetaLink assembler
; EQUATES
; timer 0 count 50 s (0b400h is 96000 = 11.0592MHz / 20Hz)
; since timers are up-counters, subtract: 10000h-0b400h = 5c00h
; normally 50 ms, 20 ms for test
TIM100 EQU 58
TIM101 EQU 0BBH ; 20ms

; THO values for different counts
TIM005 EQU 0EEH ; for 5ms
TIM010 EQU 0DCH ; for 10ms
TIM016 EQU 0CAH ; for 16.67ms
TIM020 EQU 0BBH ; for 20ms
TIM025 EQU 0ABH ; for 25ms
TIM030 EQU 09AH ; for 30ms
TIM050 EQU 05CH ; for 50ms

; R6 loop count for timer 0 int routine
TCNT EQU 18

; Timer 1 baud rates for 11.0592 MHz xtal, SMD = 0
BAUD6 EQU 0FDH ; 9600
BAUD4 EQU 0FAH ; 4800
BAUD4 EQU 0FAH ; 2400

; bit addressable location, not part of PSW
TX_FLAG EQU 7FH
TX_BUF EQU 40H ; locations 40h - 4Fh are the transis fifo
TX_HED EQU 12H
TX_TAL EQU 13H
TX_CNT EQU 14H

; RXBUF EQU 50H ; locations 50h - 5Fh are the recv fifo
RXVHD EQU 16H
RXVTA EQU 16H
RCVNT EQU 17H

; Beginning of rom, INTERRUPT VECTORS, 3 bytes each.
ORG 0

BEGIN:
LJMP START ; jump to start of program

; External Interrupt IED (from /INT0 pin) (org 3)
RETI ; put jump to your own routine here

; ORG 000BH ; vector for timer0/TF0 overflow
LJMP TO_ISR ; go to interrupt service routine

; ORG 0013H ; vector for IE1 (from /INT1 pin)
RETI ; put jump to your own routine here

; ORG 001BH ; vector for timer1/TF1 overflow
RETI ; use as baud rate generator here

; ORG 0023H ; vector for serial interrupts
LJMP SERIAL ; go to handler

; This is the earliest place to put an ID string in the rom.
DB 'DIBs8051',0
```
I put my interrupt test program and the 80C31 back in the board and reached for the scope and started poking around. At the serial port transmit output were some odd pulses, nothing that looked like standard serial output. Other pins seemed to be alright. I got out all my 8051 books, I mean all of them. They were stacked on the desk, the workbench, and the chairs. I went back and forth between the books and the program for several hours. I decided that the order of the instructions looked a little odd, so I started reorganizing the code to make sure the all the required registers were setup in what looked like the correct order. I also called the Intel help line to ask them. The person who answered tried to be helpful, but couldn’t tell me what might be wrong.

Finally, after several tries and cross checking the books and some sample code, the board finally came alive and started sending out the test messages. And the second board worked also. Apparently, the CMOS version (80C31BH) was much pickier about the initialization order. I tried an 80C32 and it worked fine also. I’ve now used this program on five different 8051/31 boards and it works on all of them.

I have since found out that there are several microcontrollers that are picky about the startup sequence. I’ve been told that the 68HC11 has some things that have to be done in the first 64 clock cycles after power-on or a reset and I hope someone will provide an article on that later this year. We might have some articles on appropriate start-up code for systems and applications also.

The Code

The startup program is written for the ML-ASM51 assembler from Meta-Link which I use for my 8031 work. It is a macro assembler though I haven’t used that part of it and it uses standard Intel syntax. It includes definition files for many 8051 variations. Meta-Link has made this available for free on the Phillips/Signetics BBS for a number of years and gave me permission to post it on the TCJ/DIBs BBS and the TCJ Web page so that TCJ readers could get it also.
MOV DPTR,#MSG7 ; get ptr for msg7
SJMP SNDMSG8 ;

SNDMSG8: MOV R7,#0 ; clear msg index counter
MOV R5,#21H ; reset to loop char
MOV DPTR,#XMSG ; get ptr for xmsg (msg0)

SNDMSG9: CALL ASM2G ;
MOV MSGCNT,#1FH ; reset counter to prevent bogus messages
INC R7 ; inc msg index
RET

; Function to send a char through the fifo (char in B register).
; Fifo functions write to tail, read from head, holds only 16 bytes.
; Set the ACC to 0FFh if the send was OK and clears it otherwise.
SEND: CLR EA ; disable interrupts
PUSH PSW ; save old register bank
SETB R0 ; get to register bank 1
CLR RS1
MOV A,TX_CNT ; get count
JNZ QUE SND ; add to fifo only if bytes are already in the fifo
JB TX_FIFO,QUE SND ; also put in fifo if a transmit is in progress
MOV SBUF,B ; send char immediately
SETB TXFLAG ; set user flag to show transmit in progress
MOV A,#OFFH ; set success
SJMP SNDEXIT

; Put byte in queue
QUE SND:
SUBB A,#10H ; max count
JZ SNDEXIT ; if zero, fifo is full
ADD A,TX_TAL ; add offset to tail
MOV R0,A ; get result in a register for indirect address
MOV R0,B ; put byte in fifo
; now adjust ptr and count in memory
INC TX_TAL ;
ANL TX_TAL,#0FH ;
INC TX_CNT
MOV A,#OFFH ; set success
SNDEXIT:
POP PSW
SETB EA ; re-enable interrupts
RET

; Function to check for recv'd char's If a char is ready, the ACC bit
; will be set to 0FFh and the character will be in the B register
; Otherwise, the ACC is cleared and the B register is undefined
RECV:
CLR EA ; disable interrupts
PUSH PSW ; save old register bank
MOV A,RCVCNT ; get count
JZ RECV_END ; if 0, no char available
SETB R0 ; get to register bank 1
CLR RS1
GET_IT: MOV A,#RCVBUF ; get addr of recv buffer
ADD A,RCVFNED ; add head ptr
MOV R0,A ; register for indirect address
MOV B,#R ; fetch the char
; adjust ptr and count in memory
INC RCVFED
ANL RCVFED,#0FH
DEC RCVCNT
MOV A,#OFFH ; set success flag
RECV_END:
POP PSW
SETB EA ; re-enable interrupts
RET

; define a handler for the serial interrupt
SERIAL:
CLR EA ; disable interrupts
PUSH ACC ; save the ACCumulator, B register, and flags
PUSH PSW ; we do a bank switch to bank 1
SETB R0 ; select register bank 1
CLR RS1
JNB RI,TX_OP ; jump if the Rx bit was not set
otherwise, drop into recv code
CLR RI ; clear the receive interrupt bit
MOV A,RCVCNT ; get recv count
JCNE A,#10H,RCV_STS ; if not max, then store char
SJMPC NO_ROOM ; else the fifo was full, so discard this char

; Store this char
RCV_STS: MOV A,#RCVBUF ; get the base of the recv fifo
ADD A,RCVTAL ; add the offset to the tail
MOV R0,A ; get the result into a register
MOV R0,BUF ; move the char into the fifo
; adjust mem ptr and count
INC RCVTAL

The Computer Journal / #78

As usual, the first part of the file is a bunch of equates for useful constants like the serial port baud rates and different timer constants. The executable code starts with the reset and interrupt vectors. In the 8051 family, all of the interrupt vectors are 'hardwired' specific ROM locations starting with the RESET vector at 0.

Next comes the Initialization routines which are the ones that turned out to be critical. Although the interrupts are disabled at power-up and reset, the first thing I do is clear them anyway. That way, I'm sure they're off and I can also call the routines later in the program if I want. Next I move the stack up to an unused memory area. The 8051/31 requires the stack to be in internal memory which is only 128 bytes. The 8052/32 expands that to 256 bytes.

Now I make sure the timers are Off and load the time constants for them and initialize memory locations used by the routines. Timer0 is used for a periodic interrupt and Timer1 is the baud rate generator. SCON sets up the serial port and following that is the setup for the receive and transmit fifos.

I have everything setup now and I can start turning things on. First I enable the timers so they will start running. Next I enable the interrupts leaving the master enable until last. The last thing before the Main program is to send the sign-on message.

The program that follows just exercises the timers and serial port. The Main loop checks for a serial input and echoes it if it's there and then checks the MSGCNT flag to see if the timer0 interrupt has decremented it to 0. If it has, the SNDMSG routine is called to send one of 8 different messages.

The rest of the file contains the message handling routines, the serial and timer interrupt service routines, and the serial fifo handling routine. And that's all there is to it.

This program (DIBS8051.ASM) and the ML-ASM51 cross-assembler are available on the TCJ Web pages and the TCJ/DIBs BBS. The BBS also has a lot of other 8051 related files.
; Transmit portion of interrupt
TX_OP:
; operations done when tx is set
JNB TI,SER_RET ; break out of this
CLR TI ; clear transmit interrupt for next int
MOV A,TX_CNT ; get Tx count
JNZ CH SND ; send next char
CLR TXFLAG ; no byte available, clear tx-in-use flag
SJMP SER_RET ; then return

CH SND:
MOV A,#TX_BUF ; get the base of the Tx buffer
ADD A,TX_RD ; add the offset to the head
MOV RO,A ; get the addr into a register
MOV SBUF,ERO ; send the char to the output
SETB TXFLAG ; set user flag for transmit in progress
; adjust mem ptr and count
INC TX_MEP ;
ANL TX_MEP,#OFFH ;
DEC TX_CNT ;

SER RET:
POP PSW
POP ACC
SETB EA ; re-enable interrupts
RETI ; return from interrupt

; define the interrupt handler for timer 0
; provides periodic interrupts
T0_ISR: CLR EA ; disable interrupts
PUSH ACC ; save the accumulator
PUSH PSW ; select register set 0 where 'TOCNT' is.
CLR R5
CLR R6
; reset/restart timer 0
CLR T0R ; clear the t0 run bit
MOV T0R,#TIML0 ; reset the counts
MOV T0H,#TIMHI
SETB T0R ; set the run bit for timer 0 again
; dec and test message loop count
DJNZ R6,TOEXIT ; if end of count, send char from R5
MOV B,R5 ; put char variable from R5 into B
LCALL SEND ; let send it in the fifo
MOV R6,#TOCNT ; reset t0 message loop count
INC R5 ; next char

TOEXIT:
; for azmsg test
INC MSCNT ; inc main loop message counter
POP PSW
POP ACC
SETB EA ; re-enable interrupts
RETI ; return from interrupt

; send ASCII message, enter with pointer to msg in DPLTR.
AZMSG:
MOV R2,#0 ; clear byte counter
AZMSG1: CLR A ;
MOV A,@EA+DPI ; get byte
JZ AZMSGXT ; zero is terminator
MOV B,A ;
LCALL SEND ;
INC R2 ; inc byte count
INC DPI ; inc byte pointer
SJMP AZMSG1

AZMSGXT:
; *** un-comment the next line to show the byte count
; for each message at P1
; MOV P1,R2 ; output count at P1
RET ;

; Messages
dibmsg: db 0DH,0AH,'DIBs8051','!',0DH,0AH,0,'????????',0
; main loop messages
XMSG: db 0DH,0AH,0AH,'Msg0 --> ',0
MSG1: db 0DH,0AH,'Msg1 --> ',0
MSG2: db 0DH,0AH,'Msg2 --> ',0
MSG3: db 0DH,0AH,'Msg3 --> ',0
MSG4: db 0DH,0AH,'Msg4 --> ',0
MSG5: db 0DH,0AH,'Msg5 --> ',0
MSG6: db 0DH,0AH,'Msg6 --> ',0
MSG7: db 0DH,0AH,'Msg7 --> ',0
; END ; end of assembly
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 .................................. US$73
   Tilmann Reh's GIDE board was featured in several is-
   sues 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 pro-
   vided, but you have to write your own BIOS routines.
   Z80 assembly language expertise required. Group
   purchase requirement - minimum 7 orders.

CPM CD-ROM ........................................ US$25 + $4 S+H
   This is the Walnut Creek CPM 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
   CPM collection from the Internet. Since we don't have
   CD-ROM reader software for the GIDE (yet), this re-
   quires one of those other kinds of computers to read it
   and a comm program to transfer it to your CPM system.
   Group purchase requirement - minimum 17 orders.

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. They are ECB bus
plug-in cards. These are bare boards and are not for the
faint of heart. They are expensive and the parts are hard
to get. Some of the parts are obsolete, especially the Z280
CPU chip.

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

* 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

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 tcej@psyber.com

The Computer Journal / #78 41
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 dibald@netcom.com, CompuServe 70432,444, tcj@psyber.com.

TCJ Adviser: Bill D. Kibler, PO Box 535, Lincoln, CA 95648, (916)645-1670, GEnie: B.Kibl, 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, 4000 Norris Ave., Sacramento, CA 95821, (916)483-0312 (eves). Also sells Kaypro upgrades. CIS 73664,2470 (73664.2470@compuserserve.com).

S-100 Support: Herb Johnson, 59 Main Blvd. Ewing, NJ 08618 (609)771-1503. Also solds 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.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, PO Box 161513, Sacramento, CA 95816-1513, BBS: (916)372-3646. Publishes newsletter, $15.00 membership, meetings at SMUD 6201 St., Sacramento CA.


NOVAUG: 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 Norma Norman. Disseuser 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, ADVISKA newsletter by David Cobley, 17-885 Berwick Rd. Qualicum Beach, B.C., Canada V9K 1N7, (604)752-1984. docb t@g. Island.net

Northern Illinois ADAM Users' 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 Zork. 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 Thursdays at SMUD 59Th St. (ed. bldg.).

The Computer Journal / #78
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: b6483@scn.org

The SN0-KING Kaypro User Group, contact Donald Anderson, 13227 2nd Ave South, Burien, WA 98168-2637.

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

OTHER PUBLICATIONS


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 88/89’er, Kirk L. Thompson editor, PO Box 548, West Branch IA 52358, (319)643-7136. $15/yr(US) publication for H-8/88s.


the world of 68 micros, by FARNA Systems, PO Box 321, Warner Robins, GA 31099-0321. E-mail: dsrfsox@delphi.com.

New magazine for support of old CoCo’s and other 68xx(x) systems.

Amstrad PCW SIG, newsletter by Al Wash, 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 Greetish. Computer History and more.

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

QL Hacker’s Journal (QHI), Timothy Swenson, 5615 Botkins Rd., Huber Heights, OH 45424, (513) 233-2178, sent mail & E-mail, swensotc@ss2.sews.wpaaf.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. Email: fdavis@holli.com.

SUPPORT BUSINESSES


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.


Star-K Software Systems Corp. PO Box 209, Mt. Kisco, NY 10549, (914)241-0287, BBS: (914)241-3307. SK*DOS 6809/6800 operating system and software. Some educational products, call for catalog.

Peripheral Technology, 1250 E. Piedmont Rd., Marietta, GA 30067, (404)973-2156. 6809/6800 single board system. 68K ISA bus compatible system.

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


GIMIXOS-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.

nSYSTEMS, 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.


Trico 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.

The Computer Journal Back Issues
Sales limited to supplies in stock.

Volume Number 1:
- Issues 1 to 8
- Serial interfacing and modem transfers
- Floppy disk formats, print spooler
- Add-on 8087 Math Chip, fiber optics
- 5-100-HZ RES graphics.
- Controlling DC motors, multi-user columns
- VIC-20 EPROM Programmer, CP/M 3.0
- CP/M user functions and integration

Volume Number 2:
- Issues 10 to 19
- Fortran: Write and Read Your Own
- 68000 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 Z801.

Volume Number 3:
- Issues 20 to 25
- Designing an 8085 SBC
- Using Apple Graphics from CP/M
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller: WDD2 MDD2 Controller for CP/M 68K
- Extending Turbo Pascal: series
- Unbridling: The Arcane Art
- Analog Multiplexer: Data Acquisition: Connecting Your Computer to the Real World
- Programming the 8085 SBC
- NEW-DOS: series
- Variability in the BBS C Standard Library
- The SCSI interface: series
- Using Turbo Pascal ISAM Files
- The Ampro Little Board Column: series
- C Columns: 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
- Selektor & Building a System
- Introduction to Assembly Code for CP/M
- Ampro 16 Column
- Z-Time 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 8180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside Ampro Computers
- NEW-DOS: The CCP Commands (continued)
- Z8024
- Affordable C Compilers
- Concurrent Multitasking: A Review of DOS
- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Scientific CO2 Generation: Disassamblying Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis & Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- Z-80
- The Hitachi HD64180 New Life for 8-bit Systems
- Z8024 Corner: Command Line Generators and Aliases
- A Turbo Program:orph; Writing a Turbo Tutor in Fort
- Disk Parameters: Modifying the CP/M Disk Parameter file in Foreign Disk Formats
- Starting Your Own BBS
- Build an A/D Converter for the Ampro Little Board

HD64180: Setting the Wat Times & RAM Refresh using PRT & DMA
- Using SCI for Real 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 2
- Using the Hitachi HD64180 Embedded Processor Design
- 68000: Why use a new OS and the 68000?

The Computer Journal / #78

Issue Number 43:
- Standardize Your Floppy Drive Disk.
- A New History Shell for ZSystem.
- Heath's HD32, The New HD.
- The ZSystem Corner: Software update service, and customizing NZCOCM.
- 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:
- Multitasking in Forth: New Micros F68FCC11 and Max Forth.
- Mysteries of PC Floppy Disks Revealed: FM, NMF, and the beast called FCB.
- DoSDis: 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 telecommunicators.

Issue Number 45:
- Embedded Systems for the Tenderfoot.
- Getting Started with the 8031.
- The Z-System and Turbo Pascal: Patcher TURBO.COC to access the Z-System.
- Embedded Modules in CP/M: Designing a Z80 RS-232 communications gateway, part 1.
- Advanced CP/M: String searches and tuning Jefft.
- Animation with Turbo C: Part 2, screen animations.
- Real Computing: The NS32000.

Issue Number 46:
- Build a Long Distance Printer Driver.
- Using the 68031's built-in UART.
- Embedded Modules in CP/M.
- The Z-System Corner: Patching The Word Processor check, 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 668C11F.
- Z-System Corner: ZMATE Macro Language.
- Using 8031 Interrupts.
- 1:1: What's & Why You Need to Know.
- ZCIFS & Turbo, Too.
- Tips on Using LCD's: Interfacing to the 68070.
- Real Computing: Debugging, NS32 Multi-tasking & Distributed Systems.
- Long Distance Printer Driver: correction.
- ROBO-SCG 90.

Issue Number 48:
- Fast Math Using Logarithms.
- Forth and Forth Assembler.
- Module-Z: Turbo ZAP.
- Adding a Bernoulli Drive to a CP/M Printer (Building a SCI Interface).
- Review of "B".
- Z-System Corner: Patching MET-Plus and TheWord is known.

Issue Number 49:
- Computer Network Power Protection.
- Floppy Disk Alignment WINDOWZ.
- Motor Control with the 801.
- Getting Started in Assembly Language.
- PMATE/ZMATE Macros, Part 2.
- Z-System Corner: ZBest Software.

Issue Number 50:
- Reflash a System CPU with the Z818.
- Floppy Disk Alignment WINDOWZ.
- Motor Control with the 801.
- Getting Started in Assembly Language.
- PMATE/ZMATE Macros, Part 2.
- Local Area Networks.
- Using the ZCIFS.
- PMATE/ZMATE Macros, Part 3.
- Z-System Corner, PEDZ ZBest Software.
- Real Computing, 32F16, Cache.
Issue Number 51:
• Introducing the YASBEC
• Floppy Disk Alignment wRTXEBL, Pt 3
• High Speed Modems on Eight Bit Systems
• A Z80 Talker and Host
• Local Area Networks—Ethernet
• UNIX Connectivity on the Cheap
• PC Hard Disk Partition Table
• A Short Introduction to FORTRAN
• Stepped Inference in Embodied Control
• Real Computing, the 32C3610, Swordfish
• PMATE/ZMATE Macros
• Z8 System Corner, The Trantox Festival
• Z8-Best Software, The Z8HELF 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 F86Hc11
• Z8 System Corner, Programming for Compatibility
• Z8-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
• Z8-Best Software
• Getting Started in Assembly Language
• The NZCOM IOP

Issue Number 54:
• B.Y.O. Assembler
• Local Area Networks
• Advanced CP/M
• Z8-PHY for a 16-bit Intel Platform
• Real Computing
• Intufts and the Z80
• 8 MHz on a 8080
• Hardware Heaven

Issue Number 55:
• Fuzzology 101
• The Cyclic Predicators Check in Forth
• The Interworkstation Protocol (IP)
• Hardware Heaven

Issue Number 56:
• TJc - The Next Ten Years
• Input Expansion for 8081
• Z8-System Corner - 2nd&Fed
• Connecting IDE devices to 8-bit systems
• Queens in Forth
• Real Computing - Linux, BSO 266, Minix
• Kayoke - DUTR4 Disk File Transfers
• Analog Signal Generation

U.S. Canada/Mexico Europe/Other
(Surface) (Air) (Surface) (Air)

Subscriptions (CA not taxable)
1 year (6 issues) $24.00 $32.00 $34.00 $34.00
2 years (12 issues) $44.00 $60.00 $64.00 $64.00
Back Issues (CA tax)
Shipping & Handling for each issue ordered $3.00 $3.50 $6.50 $4.00 $17.00
Bound Volumes $20.00 ea. $3.00 $3.50 $6.50 $4.00 $17.00
#32 thru #43 are $3.00 ea. $1.00 $1.25 $1.25 $1.50 $2.50
#44 and up are $4.00 ea. $1.25 $1.25 $1.75 $2.00 $3.50

Items:

Name: ________________________________
Address: ________________________________

Payment is accepted by check, money order, or Credit Card (MC, VISA, CarteBlanche, Diners Club). Checks must be in US funds, drawn on a US bank. Credit Card orders can call 1(800) 424-8625.

TCJ The Computer Journal
P.O. Box 3900, Citrus Heights, CA 95611-3900
Phone (916) 722-4970 / Fax (916) 722-7480

The Computer Journal / #78
It has been great not doing the managerial duties of TCJ, besides Dave did a wonderful job on the last issue. I think his changes will make TCJ just that much better. So thanks for continuing on with TCJ, Dave.

**Duties**

My duties for TCJ are mostly this column and the occasional article. You will find my article on picking embedded programming options elsewhere and is the response to letters and email messages I have received. So if there is some item you need some advice or help on, please drop me a message and I'll see what I can do for you. I think this makes my thirteenth year supporting TCJ readers.

**PLD's**

I want to thank Robert Brown from Alta Engineering for his great PLD article, not only is he supplying a cheap PLD programmer, but he also explained very clearly the why of using PLD's. One reason I have been down on using PLD's was simply the cost of a programmer. Most of our readers have a mantra that goes something like this: "if it cost more than $100, I'm not interested!" I think Bob's product stays well within that criteria.

Another point of the article is his explanation of why and when to use PLD's. His figure 5 is exactly what I wanted for showing the main reason to use a PLD. You can see very clearly why it is cost effective to use one in that case. All the parts used in the TTL version are not the 25 cent variety and the number of sockets and trace savings more than make up for the higher cost of the PLD. It is also very common for items like this (clock timing) to get changed as the design progresses. Simple changes in the PLD can mean you are still able to use the same board to achieve your new goals. Memory decoding is also covered in the article. The only thing he didn't mention was when not to use a PLD. I recently designed a board and used a 139 for it's decoding. The 139 is cheap ($1.90) and does the needed decoding with extra pins to spare. The way to decide is not on cost alone, but on logic reasoning. If you really do need the fine grain memory layout as outlined in the article, a PLD is your only option. Any design using a precise memory map like that is also apt to change many times before your done and burning a new PLD is the only way to go.

I have included the schematic of some decoding I used. You can see how simple my decoding is. The design has two sections, one that talks to the PC BUS and one part that is a standalone 8031. There is a dual ported RAM device which fits between the 8031 and the PC, for buffering data or looking at variables. I thought about using a PLD to replace both decoding parts, since then maybe the cost could be justified. The problem here is that you run out of input or output pins. That means two PLD's would be needed and since the 138/139's are only $5.00 for both, compared to $2.49 for a single 16V8, PLD's are just too costly. You also need to keep in mind, that this would be the only board in house using a PLD of any type. It is pretty hard to justify a single PLD when the other 30 to 40 boards you make all work fine without them.

You also need to look at the board layout. In my design, the dual ported memory sits in the middle of the board, with circuitry on either side. If I tried to use one part for the cost savings, some of the lines would have to cross the middle of the board, and it was all I could do to get 30 some odd lines from each side to land properly on the memory. Having to run 4 or 5 more lines through this mess might just be impossible. Keeping in mind how the board traces run, is another reason to use or not use PLD's.

My experience has shown that often these fine grain designs lack real reasoning for their design choice. My 139 is anything but fine grain, you waste tons of memory, are very fixed in your options, and are pretty much stuck in doing it that way forever. Since my design's layout is most likely to never change, never use all the options I provide anyway, these are trade off's I can certainly afford to play with.

That's basically (and has always been) my position. As the design gets more complex and you need more flexibility, going from basic TTL to PLD's changes from overkill to the most appropriate for the application. It's not a black and white choice! Every project can have a time when the design is better suited for one choice over another, but it then could change directions as you continue to develop and polish your project. My stand still remains, that you must know both options to be a cost effective designer in today's market of lean and mean machines.

**Code/Data**

While I was looking at the memory layout, I remembered I had to make a small change in the design to use Brad's Camel Forth. Remember that 8031's have separate space for DATA and CODE. Forth needs to run code out of data memory. That means you must combine the two signal lines that control the use of RAM or ROM devices.

The /PSEN signal is used to turn on the ROM device, while the /RD signal is used in talking to regular RAM. Keeping the memory areas separate, means it is possible to have 64K of each memory. My combining of the spaces limits total memory usage to 64K. Now please keep in mind these are suppose to be embedded systems and I have always felt if you needed more than 16 or 20K of ROM,
you must be looking at your design a bit askew.

Implementing the dual use of memory is very simple. There are other ways than the one I use, but two simple diodes work just fine. You need the pull-up resistors to make sure they are biased properly, but overall cost is well under 20 cents. I use a jumper so you can select whether or not the option is in use.

After installing the diodes and resistors, using Camel Forth required that I recompile it for the memory map shown in the schematic. I was then able to burn a ROM, power up and get the OK prompt. I then wrote test routines to check out memory and other operations. Camel Forth made hardware testing a breeze and that design problems were not associated with code I made up for the projected use. Several times I burnt new ROMs to test a completely different use only to have it lock up with no output of any kind. I simply put back Camel Forth and got the OK to tell me it was in my new code. A very handy tool!

**linux**

My full time duties now encompass doing Unix development. I had some Unix experience before taking this job and have since discovered how much more complex and involved Unix hacking is from regular programming. To help bring myself up to speed I started using and playing with Linux.

If you own a 386/486 system and have some extra hard disk space, loading and using Linux is very simple. I have tried several versions of the releases and am still not sure which installation to recommend. For the casual user or just curious experimenter, my favorite is the Mini Linux. This version is not recommended by the real Linux users since it resides on top of DOS.

Linux is so well developed, there are versions for almost any option and system around. Some users in Portugal, I think, wanted a simple version to run under DOS to be able to connect with Internet servers and get their E-mail. The version runs entirely on DOS and fits in 20 meg subdirectory.

I use it as a simple testing ground and place to un-archive files, all without having to install a full system. It will do X-windows and could do everything a full Linux does, but I agree with the others, that if you are doing a full system, partition your hard drives and do it properly. The mini version is just a simple scaled down, single task system installation. I plan on doing some testing to see if I can use it in place of our expensive Windows based X-windows interface to the HP-UX systems I work on.

**Embedded**

I keep seeing various articles and messages about using Linux for embedded development. I suppose you can do that, and there are certainly plenty of tools to help you in that idea. I find some features of Linux/Unix a bit overkill and restrictive. You can get Linux to boot fast, but I find the 5 or so minutes a bit long. You need to remember however, that normally you would boot Linux and never shut it down.

For those of us who have grown up doing embedded and hardware work for years, the idea of not being able to just hit the reset switch to get yourself out of a problem seems like a step backwards. The idea behind the bigger systems like Linux is memory and hardware protection. Your program can’t really get control of what we are so use to dealing with in simpler systems. I find that the biggest problem for me.

When dealing with embedded devices, you know where every device is, how memory is laid out, and usually which hex codes are needed to make it play. Linux/Unix systems do not allow that sort of access. I look back at several projects and see that using Linux would have made the development considerably longer and more difficult.

Since I now do most things in C, my understanding of C is becoming more developed. C really works on systems like Linux. You have large libraries that handle all the inner system calls. Those system calls are part of the operating system, and as such you give a direct hook into the system kernel. By that I mean the design of the operating system is based on the use of the libraries. The big difference for embedded use is these same calls just get converted to some assembly language code that gets included into the ROM, making it very bloated with often unused code.

A better way to explain this would be program size needed to do similar processes. Most Unix systems have sharable libraries, this means the code is not sucked into the program, but called at run time, not possible in DOS. So it is possible to make a few hundred byte program in Unix, that would have to have all the actual library calls coded inside the program to run on DOS, since DOS has no sharable option. That is a big difference and also why I don’t recommend it for embedded controllers.

For embedded work that means the library code gets sucked into the ROM and thus the ROM can get pretty large. In the case of using Forth, it is more like the Linux/Unix idea. The kernel contains or is made up of all the tools you need and you only make calls to them to get things done. It also reminds me of why a lot of CP/M users have trouble understanding the PC architecture. In CP/M and other simple systems, one can grasp and understand the whole system, and thus take advantage of system hooks usually hidden by more complex layers.

That’s it...

Well keep hacking till next time. Bill Kibler.
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:

<table>
<thead>
<tr>
<th>Size</th>
<th>Once</th>
<th>4+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>$150</td>
<td>$90</td>
</tr>
<tr>
<td>1/2 Page</td>
<td>$80</td>
<td>$60</td>
</tr>
<tr>
<td>1/3 Page</td>
<td>$60</td>
<td>$45</td>
</tr>
<tr>
<td>1/4 Page</td>
<td>$50</td>
<td>$40</td>
</tr>
<tr>
<td>Market Place</td>
<td>$30</td>
<td>$120/yr</td>
</tr>
</tbody>
</table>

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

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

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 pre-printed 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.


FOR SALE, Motorola Development Kits. Fuzzy logic (FLEDKT00), 68HC16 (M68HC16TK), and 68HC05 (M68HC705KICS). MAKE OFFER! Other Miscellaneous Electronics available, for a list send SASE to Gene Francisco, 10226 N. 29th St., Tampa, FL 33612. Email neelrah@cfnet.com

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

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

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 magazines, TCJ Classified it works!
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.

Subscription: $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

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 floppy
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

VERSATILE 80C32 AND 68HC11
SINGLE BOARD COMPUTERS

The DCO8002-1 includes the following:
- 1.28M Baud 80C32 processor.
- 32K of EPROM.
- A different memory map.
- Extended BASIC-80 with 28 additional commands.
The DCO8011-1 includes the following:
- IBM - MS/BIOS/11 processor.
- 32K of EPROM jumper selectable as 2, 16K or 32K.
- MS-BASIC with adjustable read and write speed commands.
All units include the following standard features:
- 8x16 pixel LCD.
- 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 5 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 units come with a 5 volt DC wall cable, 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
1843 Summer Ct.
Las Cruces, NM 88001
Ph. (505) 524-4028
$140.00 kit or assembled and tested. Add $5.00 shipping & 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.tageta.com/jfar/mvp.html

$79.95 68HC11
8K EEPROm for More
Program Space!

SBC-8K
- Small Size, 4.3" x 3.6"!
- Low Power, <60 ma!
- 1632 Byte EEPROM!
- 566 Byte RAM!
- DB-9 RS-232!
- 24-TTL I/O Bits!
- 5-A/D Inputs!
- 8-coin Slot Circuit!
- 5-MHz Clock!
- Log Data with SERIAL!


LDG Electronics
1645 Perrin Road
SL Leonard, MD 20685
410-586-2177

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

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 constitutes 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