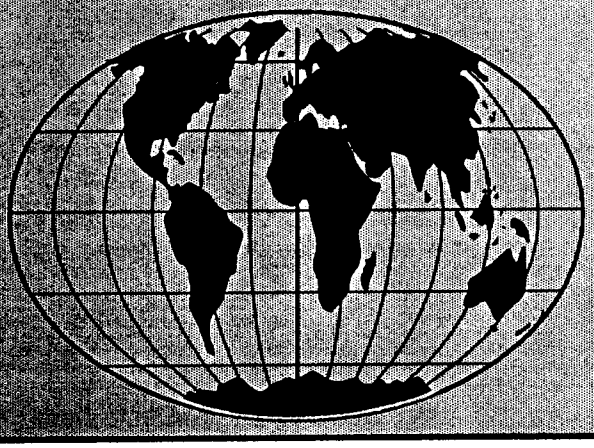
Small Scale Computing Since 1983

Supporting the *Trailing Edge of Technology*



The Computer Journal

Issue Number 80 Fall 1997 US $4.00

ISSN # 0748-9331

Hands-on Hardware and Software

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Dave Baldwin

Previous Publishers
Bill D. Kibler
Chris McEwen

Technical Consultant
Bill D. Kibler

Contributing Editors
Ronald W. Anderson
Richard Rodman
Helmut Jungkunz
Tilmann Reh

The Computer Journal is published by DIBs Electronic Design and mailed from *The Computer Journal*, P. O. Box 3900, Citrus Heights, CA 95611, (916) 722-4970.

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

TCJ The Computer Journal

Issue Number 80, Fall 1997

Editor's Column

And in this issue...

We have Win95 in the house now. My wife's laptop for work came with it and she's required to use it. The only crashes she's had involve the online connection she uses for work. Otherwise there have been no 'problems' but...

Looking at Win95 on her computer has verified my suspicions about it. The primary purpose of Win95 was to boost lagging sales by Microsoft and the other major PC software and hardware manufacturers. They made enough changes to make sure that your old software wouldn't run as well as their new software. They also made many little changes in the names of things so that finding a function that you used in Win3.1 or DOS sometimes becomes a little challenging.

Another thing is the hardware. Win95 expects certain things from the hardware and can be uncooperative when it doesn't find them. Win95 is 'designed' to be smarter than the average consumer that buys it (and I wouldn't be surprised if it is). It was not intended to cooperate with the minority of us who think we can set things up ourselves. My recommendation is that if you really need Win95, get a new machine that is intended to run it (and save your old machines for your old software). It can minimize the problems you run into.

I'm not against Windows or any other particular software or hardware. The TCJ 'philosophy' is that you use products for what they're good for. I use Win3.1 programs to publish TCJ. No DOS program that I know of will do the page layout that I need. On the other hand, I always make a note of the things that people have done that seem to benefit them more than me. I've paid (a lot) for the software I use and I don't particularly like the idea that I should pay again to do the same thing because someone else needs to sell a new product.

With the exception of Desk-top publishing and graphics, every program I use to make money runs under DOS. Well, actually, SLR's Z80ASM runs under CP/M and it still makes me some money. Obviously, most of the 'big' companies aren't making any money from me at the moment. I'm sure they'll 'get me' eventually. When I just gotta have the latest upgrade, it will be a Win95 version and I'll have to break down and buy a Win95 machine. Then I'll have every thing from my first CP/M machine (a Bigboard 1) to a Pentium here in my office. All I have to do is find a place to put all of them.

Dave Baldwin,

Editor/Publisher

PS: If you find yourself dependent, like I am, on older software, Make Sure You Have Backup copies!! Many companies are out of business and many others won't support their older software.

We lead off with **B/P Bios** by Hal Bowers and Cam Cotrill. They're still working on it and refining it. Next is my article on the **PC serial ports in ASM and C**. There have been a lot of requests for working code for the PC serial ports so here are three small terminal programs, two in ASM86 and one in C. This is one of those 'easy' things. One of my customers recently thanked me for doing a custom version for him in only two weeks. I told him it wasn't hard, all it took was ten years preparation.

Allison Parent, a new author for TCJ but an old hand at micros and CP/M, writes about adapting PC hard disk cards to other systems in **Cheap Hard Disk Controller**. Part 3 of the continuing saga of **Simplex III** is here also along with Rick Rodman's **Real Computing** column.

Another new author, André Fachat, presents his **High Speed Serial for the 65xx, Add a 16550 to a 6502**. He shows what it takes to adapt a chip to a system it wasn't designed for. I'm not sure what the 8250/16550 were designed for, but he has it working with his Commodore C64.

Ron Anderson continues with **Small System Support** and Bill Kibler with **The Computer Corner**. Helmut Jungkunz is back with the **European Beat** this issue with an article about coercing dBase to cooperate with the ZCPR TCAP.

Next Time...

Part two of **B/P BIOS** is in the next issue along with Ron Anderson's PC serial and parallel port test programs in C. There will be more PC programming articles along with embedded hardware and software in the issues to come.

Don Jhindra, author of The Little Big LAN, is working on some networking articles but I don't know when they will be ready. He's done some fascinating things with LBL. In addition to standard network cards (Ethernet and Arcnet), LBL has built-in support for both serial and parallel ports. I bought a copy but I haven't had time to install it. Since Don also provides programming support, it looks like I should be able to make LBL talk to just about anything.

About The 'Net

New Column

by Ted Deppner

Internet Myths

The Internet: Myth No More

The Internet is the latest storehouse of myths in people's minds. Some myths are heated, (pornography galore!), some are bad (those nasty Internet Police), and the bare truth about the Internet gets little air time. Myths usually talk about what the Internet is, completely ignoring what it isn't. We'll talk about both and give you a more complete view of the Internet as it is, and what it may become.

The Internet is above all else a reflection of society. There is something for everyone online. There are match-cover collectors, TTL chip directories, model railroad fanatics, crocheters and knitters, libraries, state and federal agencies, and so on. *(and TCJ, of course)* You can find the latest in golf clubs, USR's X2 upgrade, or just the drivers for your sound card. And yes, you can find pornography, political propaganda, and the latest in home built explosives. This is a reflection of society, remember?

Myth #1: Pornography and other stuff. Just like in the physical world around you, you make the choices about where you go and how you spend your time. You make the choices, they aren't forced on you. If you don't like the adult book store, you don't go in. You can just as easily choose the local electronics hut of retired hardware. It's the same on the Internet: you go where your interest are, and stay away from those "other" areas. To each his own.

Myth #2: The Internet Police and Big Brother. Those guys sure do get around. Hah. The Internet is a large collection of cooperating but separate entities—there is no centralized anything. As such, there is no central repository of information, no government affiliation or governmental control, and no police. This doesn't mean you can do anything you want and "get away with it". While there is no central control and record keeping, records are kept.

A record of where you have been, what file you have downloaded, or where you sent email to is kept, but the contents of these items is not. Somewhere deep in your service provider's logs it says "me@here.com" sent an email to "him@there.com", but what the message said is not recorded. There's a log entry that someone at there.com viewed my home page, but who that actually was isn't listed.

Time, date and origination are the extent that is kept. Your name, address, or anything identifying you as "you" are not kept, unless you choose to provide that information. However, armed with time, date, origination, and a court order, your Internet service provider could easily determine who was there at that time.

Myth #3: Etiquette and morals. Moral ineptitude is frowned on by the Internet community. If you send out the latest Make Money Fast file, expect lots of email calling you names. Email 10,000 people your latest sales flier out of the blue and expect your service provider to shut your account off.

Illegal activity is always illegal, and stuff you couldn't do in your local communities you can't reasonably expect to do online either. If you have questions, contact your service provider, they should be more than happy to help you.

Myth #4: The Internet has everything. Well, yes it does, though it may not be exactly what you wanted. Anything that is freely available in the world is online. Works by Shakespeare, Poe, and Doyle, sports scores, stock reports, and so forth. You can find plenty of home built circuit diagrams for all sorts of devices, but you probably won't find the latest schematic for your TV online. You won't find the latest book by Stephen King online, nor up to the minute stock information. You can find reviews of King's latest book, and you can get your stock information delayed by fifteen minutes. Of course, if you want to pay to get King's book you can, likewise you can get your stocks up to the minute if you subscribe to that service.

Myth #5: Buying the Unknown. Everywhere you go in society, you have offers to purchase things: Books, magazines, new computers, the latest Microsoft collection of bugs, and new "must have" widgets. In the real world, you have to pull out the flat wallet, peek through it, and put forth green or plastic. So also goes the Internet. In general, everything online is free. If it is not free, it will tell you so, and make you give your credit card numbers and name before letting you on. You won't just stumble into some pay area and all unknowing start racking up charges. Once again, it's your choice. Good companies will provide a free sample of what they offer, so you can get a taste.

Myth #6: They'll steal my Credit Cards! Recent versions of Internet browsers can encrypt what you're sending, and you can use security programs like PGP (Pretty Good Privacy) for email. Getting your numbers would be a needle in haystack kind of problem in the first place, so add to that encryption of one sort or another, and the fact that there are much easier targets for thieves, and it's not a huge threat. Obviously you shouldn't say "hey everybody here's my credit card", but then you wouldn't do that in the real world either.

The Internet is a vast and wonderful place. It reflects the society of the world around us, and each of us can personally benefit from what the Internet offers. We can each have an effect on what the Internet is, through our participation. There are email discussion groups and USENET news groups with topics that range from iguana care to how to repair TVs. You can even put your own web pages online, detailing to the world how to make the Mr. Spiffy Widget, or simply stating how you made your networked coffee machine. So step forth without fear and newly armed with the truth about these six common Internet myths.

Happy Netting! ======

— Ted Deppner, ted@psyber.com, http://www.psyber.com

Ted takes care of the technical stuff at Psyberware where the TCJ Web pages are located.

Letters and News

All Readers

MINI Articles

READER to READER

Send your letters for Reader to Reader to:

The Computer Journal

P.O. Box 3900

Citrus Heights, CA 95611-3900

From: Christopher Burian
<cburian@ll.mit.edu>

Subject: 8051 Startup code in TCJ #78

Thanks for this great article. It's the best, maybe only, article I've seen yet that implements working interrupt-driven serial and circular queues.

I'm astounded by all the sample code out there, even sophisticated-looking monitors, that use queueless polling to send and receive. It's either that or generic pascal-like junk found in next-to-useless textbooks.

Luckily, I had a college class on 8088 assy programming which included a unit on circular queues, interrupts, programming the 8250 uart, etc., but most people don't get that kind of class. I guess it's considered unorthodox and out of the mainstream to actually teach students how to write real programs that run on genuine CPUs. Entirely too much programming at my U. was taught in virtual languages which were compiled and simulated on virtual computers existing only in the perverse imaginations of CS professors. So I think lots of people are going to find your 8051 introductory article really useful, as I have.

Despite my previous exposure to queues, your sample program is a great help to me in simplifying my own code.

I wish you'd had a little more time/ space to delve into the problem you discovered wrt interrupt code on NMOS vs. CMOS chips, and perhaps given an example of code ordering which you found not to work, just in case someone decided to make modifications to your code. Unlike the hc11's protective feature (write within the first 64 clocks only) the problem you've discovered sounds more like a bug than a feature.

On my current project, I'm again reinventing the wheel, making a monitor/ hex-downloader, because just like lots of other guys, I think I can do it a tiny bit better. One of the gimmicks I want to include and haven't seen anywhere is intelligent handling of BREAKs sent from the host. Another is use of handshaking when queues fill up, to handle delays when programming FLASH during a hexfile download, for instance.

I have a couple questions about your QUE_SND routine. Why isn't a CLR C required before the SUBB? And why use SUBB instead of CJNE?

Thanks, and keep up the good work.

Eagerly waiting for #79,

Chris Burian

I'm glad you liked the 8051 article. TCJ #77 had an article with similar code for the Z80-SIO and this issue has my article on the PC serial ports.

I never did find what the original problem was. I'm sure it's a 'bug' of some sort. The Intel 8051 experts couldn't figure it out, may not have been interested. The original program I started out with was SKELETON.ASM from the Phillips BBS and it didn't work on the CMOS 80C31. I don't know whether it worked on the NMOS version. I got a copy of one of the standard, polling monitors from a friend and it worked on all versions of the chips. That verified to me that the chips were working and that I just had to do something different with my program. I went back to the books to see what might be dependent on the sequence and this program is where I ended up.

About the QUE_SND routine, the 'CLR C' probably should have been in there. After looking at the program, it's probably just an accident that it works without it. 'CJNE' tests for the wrong thing in this case (NE). I needed to know if it was equal to 10h. A TX_CNT of 0 means that all bytes have been sent. 1 to 10h means that there are bytes to be sent in the fifo and 10h is the max the fifo can hold.

Dave Baldwin, Editor

From: "Jeffrey A. Weiner"
<j_weiner@juno.com>

Subject: XT's and kindred folk

Kudos on Issue No. 79! Well worth the boost we all gave you.

I also wanted to note that its a good thing that TCJ is covering XT-class machines. Case in point: I rescued 3 ALR Barrister XT system units from the dumpster of the building I work in. ALR had scant info on the motherboard switches, and nothing on the unique combination controller card that was in the beasties; connections for 2 pairs of floppy drives, serial and parallel ports, clock/calendar, and a mysterious 50-pin header marked "HIGH DENSITY DRIVES". I'm hopeful that a someone who reads/ writes for TCJ will e-mail me some info on this card so's I can take advantage of its features.

And those kindred spirits? Well, one thing I'd like to see is some articles about doing something useful and/or different with a DEC Rainbow. I have a few (!) of them on the premises, and

since the International Rainbow Users Group has hit the bit-bucket, it would be nice to see some guidance from some more technically-adept Rainbow afficianados.

Keep 'em flying...

Jeff Weiner Chicago, IL

From: Ron Anderson
<rwilande@concentric.net>
Subject: Hard Drives

Dave,

Last July my Conner 830 Mbyte hard drive crashed. It was eleven months old. Knowing that it would take a while to get a warranty repair, and having a use for that drive in another system, I bought a new Western Digital 1070 Mbyte drive as a replacement.

Conner had just been bought by Seagate, and partly due to transposition of two digits in the Return Authorization number (my error, not theirs), it took about six weeks to get the replacement drive back. Either they sent a new drive or they replaced the tag on it, because the new one said Seagate on it.

Shortly after that, I went to "defrag" the new WD drive and an error message appeared saying that parts of the disk were not accessible. I ran "scandisk" on the drive and it found about 1 megabyte of bad sectors all in the last 150 or so tracks of the 2488 on the drive. I decided to watch the drive carefully and to call WD if more bad sectors were mapped out later. Last month the drive failed a defrag session again and I found that another 400K of sectors had gone bad.

I called WD and they asked me to download a diagnostic from their web site and run it on the drive, and call back to report the result. The next day I phoned them and they said the drive was bad enough so they would replace it under warranty. With my credit card number as collateral, they sent me a new drive and I transferred a large number of files after reinstalling DOS and Windows 3.1 on the new drive. I ran scandisk again and before I had all the software installed, I noted some bad sectors had to be mapped out, again all at the very end of the drive.

About at that time, my replacement Seagate drive quit completely. It sounded like the head servo had gone crazy, and couldn't read data from the drive. I called Seagate and after ten minutes of tone button pushing on their "automated service line" the computer gave up and informed me that there was a problem and transferred me to a real live person, who was most helpful, agreed that the drive was under warranty, and gave me a return authorization number. This time I was careful to see that I got it right, and I have sent both drives back to their respective manufacturers.

The next day, I was talking to a friend who is experiencing the same difficulties with a Western Digital drive. He reported that about 6 megabytes of sectors had been mapped out, and that the bad list continues to grow. He has run the WD diagnostic and was planning to call for authorization to return the drive last time I talked to him.

Could it be that the manufacturers in their race to cram the most data in the least hardware have pushed the technology beyond the limit of reliability? I've used Western Digital drives for years, and in fact have a couple still running that had never had a problem (an old 100 Mbyte drive and a 170 Mbyte).

I've ordered a "refurbished" 650 Mbyte drive to have around as a spare. Maybe this one was before the technology got pushed into the area of unreliability. We'll see what happens with the Seagate.

I must add that I was treated very courteously by both Seagate and Western Digital's service people. The wait was minimal (a few minutes) in both cases.

Ron Anderson

From: Ken Deboy

Dear Bill,

I figure that if I'm going to criticize Windows95 and Microsoft, I should explain some of the problems I've had with Win95 and in dealing with Microsoft, so here goes:

1. The pamphlet (no way it could be called a "manual") doesn't explain how to do anything. For example, I had to call a 3rd-party software vendor to find out how to over-write the MBR so I could install System Commander (Win95 doesn't normally allow this to be done). Microsoft was no help because they won't provide tech support to someone who obtains Win95 pre-installed on their computer. (I could really get into what I think about a software company that won't support their own damn OS, but I won't.)

2. It sometimes locks up on me, with no way to recover except turning off the computer. Of course, there is a dire warning in the pamphlet to never do this, but what else is there to do when even <CTRL-ALT-DEL> doesn't work? Never had this problem under Win3.1 (or Linux or OS/2). This isn't a "rare" occurrence, either...at least 3 times a week, sometimes even twice in one day.

3. It won't let me delete DLLs from .../WINDOWS/SYSTEM even if they're not write-protected. It also won't let install programs over-write them if one tries to install a DLL that is already there. I have to drop back to DOS to delete the file, which is fine, since I know how to do it (no thanks to Microsoft), but what about the poor soul who has a Win95 box as their first computer?

There are other minor frustrations with Win95, but these are the major problems I have with it. Sorry for the rant, but I don't feel right criticizing a product without explaining why I feel the way I do. I hope Caldera wins their lawsuit against Microsoft, but I'm not betting on it. I feel the only way to make sure MS doesn't become a monopoly is to make sure that other OS's are economically viable. I have OS/2 on one computer, and soon I'll have it on both my computers. When I need a program, I try to find a version that will run under OS/2 or Linux. If I need a program that is only available under Win95, I always make sure to write to the software company and tell them I'd be interested in purchasing a version of the program to run under OS/2 or Linux. Maybe if enough people did that, MS would lose the strangle-hold they have on the PC software industry. Well, at least I can dream...

With best wishes,

Ken Deboy

From: Steve Johnson

Dave:

Enclosed is my early renewal check for TCJ.

Detailed comments about the magazine should wait for a future message, but you should know that I always read the magazine with a day or two of arrival and I continue to use the complete set in my study. I like the practical, problem solving approach authors take and the mix of diverse hardware, operating systems and languages covered. Even when I'm unlikely to work with a particular combination of hardware and software, I enjoy the articles and learn of practical value.

Subjects of particular interest to me are small scale networking and operating systems including Linux and Forth. CP/M remains of interest and I follow 'comp.os.cpm' closely. Like many other readers, I run MYZ80 instead of nursing my (diminished) accumulation of aging Osbornes and their many accessories. During my last binge of old equipment accumulation, I picked up a dual set of Siemens 8 inch drives which interface to an Osborne Executive with quad density internal drives and dual external hard disks. It pains me to admit it, but the Osbornes are less robust than the Kaypros.

Rick Rodman is one of my favorite TCJ contributors. My at-home computing network is much less elaborate that the Kettlepond Computing facility, which serves as inspiration. I believe my next networking step will be to extend the thinnet from the second floor to my basement workshop & home brewery (well I do have other interests besides computers). That will be a natural time to get involved with home monitoring and control.

Enough comment, time to write a check.

Signed: Steve Johnson

From: Rick Bromagen

To: Dave Baldwin

Hi Dave!

I just received the last issue of my subscription to TCJ (#79), and that reminded me there was something I have been intending to do for several months now. It's time to renew my subscription! Let's have 2 more years please.

I had been trying to figure out how to add an IDE interface to my Z80 CP/M system for several years. Then TCJ published the articles from Tilmann Reh. That was great work! We all owe people like Tilmann a big thank you. Those that have the time, knowledge, and are willing to share their efforts with everyone else have my respect.

I assume there are at least a few who have built the GIDE interface by now, and have done some tinkering with BIOS driver software. Of course we all seem to have different systems and BIOS's, so sharing source code is not simple. I am planning to put a GIDE on my Eagle II mother board. It already has a SASI (early SCSI interface) drive, but I like the challenge of putting a much faster and bigger drive on this system.

That reminds me, I would like to hear in TCJ about different ways to add large hard drives to CP/M. By that I mean how to have a 40 or 80 megabyte CP/M drive "partitioned", and how the BIOS tables describing it look. I think Ampro and several other had done this. Is there a public domain description or code sample for this?

That TCJ centerfold about the P112 board looks very interesting to me. How can I get one of these? When will TCJ offer a group buy? I am busy thinking up several projects I could "embed" this and a drive into. How about an embedded household controller with floppy (ala Circuit Cellar Ink HCS II), or a portable CP/M system? I remember David McGlone of the Z-Letter once mentioned that there was a computer built in a small, flat portable case, with maybe an LCD screen. Maybe I will try pulling the Z80, and putting the P112 board with a 3.5" drive. How about a 2.5" hard drive as well? Of course, battery power will need some thought. There are a lot of high power replacement batteries for laptops available now that could be small and light weight enough. It would be fun on my next long airplane trip, to pull out a CP/M laptop, play some games, or even do some writing.

DisConnect

Dogbone McGillicutty groaned when he heard the 'snap' that told him his network fiber had popped out of the wall. He was just trying to move his workstation a little bit to make some room. He started imagining the parade of people he'd be seeing soon. First would come the Nettertons network security people to see what had happened to 'their' connection. Of course, none of them could repair the break, but they could cite you for 'Illegal Disconnect', section blah-blah-blah of the Network Code. They would tell you to get it repaired right away. If they were in a good mood, they'd call repair for you since, of course, you couldn't because you just broke your network connection.

The last time he had called repair was from a friends terminal. First, they asked why he was calling since the system was obviously operating since he was able to call them. Then they wanted to know why he was using someone else's station and did he have permission to do so. After identifying himself, his employer, and his station and waiting for them to verify that he wasn't home using his station, they finally agreed to come repair it although they warned him that he would be charged double if it was a UIP (User Induced Problem).

After (or maybe even before) the repair droids showed up would come the HW2020 (Home&Work) people to find out why he wasn't using their systemware for his each and every need. And somewhere in one of those groups would be a Copper-Checker to make sure he wasn't one of those that had found out how to blackbox the old copper network and avoid his responsibilities (and payments). And after all of those people were done, he'd have to go through a reconnect session and verify his ID and his accounts and all of his passwords which would take a half a day. Can't have any illegals on the Net.

Banked/Portable I/O System (B/P Bios) Pt I

by Harold F. Bower and Cameron W. Cotrill

Special Feature

Z80/180 Systems

Advanced BIOS

For the past several years we have attempted to address some of what we consider to be fundamental problems in the 8-bit Z80/Z180/HD64180 system software arena. Our first effort, ZSDOS, was directed toward what we believed to be architectural weaknesses in CP/M and its clones in the late 1980s. Such weaknesses included; inefficient code, inconsistent implementations of some DOS functions, numerous different and incompatible file Date/Time Stamping methods, and just plain errors (remember Function 37?). Now in the 1990s, even more effort is needed to correct the proliferation of systems designed on faulty (or at least weak) architectures, and to provide a logical and consistent path to increase the functionality of our systems.

To understand our concerns in this area, let us review the way in which CP/M 2.2, as modified by the Z-System, uses available memory. For standard CP/M and compatible systems, the only absolute memory addresses are contained in the Base Page which is the range of 0 to 100H. All addresses above this point are variable (within certain limits). User programs are normally run from the Transient Program Area (TPA) which is the space from the top of the Base Page (100H) to the base of the Basic Disk Operating System (BDOS). Figure 1 depicts assigned areas pictorially along with some common elements assigned to each memory area.

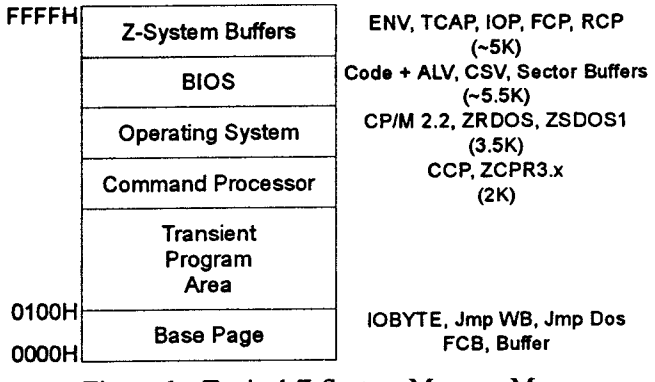


Figure 1. Typical Z-System Memory Map

The sizes depicted for the Z-System buffers is typical of many, and allows a certain functionality. It is sometimes necessary to delete some capabilities to add others, since every addition in this area pushes the other components lower, decreasing available TPA space. Likewise, any new features or more elaborate routines in the Bios decreases available TPA.

There have been some attempts at ameliorating these difficulties, but none have directly addressed the entire problem. One system in relatively widespread use is NZCOM. It allows a fairly easy method of changing systems "on the fly". The main drawback, however, is that to obtain large TPA, system features must be sacrificed by deleting or downsizing the resident Z-System segments (FCP,IOP,RCP,NDR,TCAP). To us, this method is only viable in systems which do not have extended memory capabilities. With the ever-increasing use of systems based on the Hitachi 64180 and Zilog Z180, other solutions are more attractive and offer a larger TPA without sacrificing system capabilities.

The final major factor contributing to shrinkage of TPA is the increasing commonality of large hard disk drives. Disk space is managed by a bit-mapped buffer (ALV) where each bit represents an allocation block of storage space. Typical allocation units are 2k for floppy diskettes and 4k for Hard Disk Partitions. Assuming 4k allocation blocks, a 20 MB drive needs approximately 20000/4 = 5000 bits or 625 bytes. With 80 to 100 MB drives being common these days (one B/P user reported that the smallest drive he could obtain was 850 MB!) you should see that several kilobytes are now required, further reducing available TPA space.

The first requirement to place us on the road to more powerful systems is to overcome the 64k memory limit imposed on direct access by the Z80 family of processors in a consistent and logical manner. Such a technique, generically called memory banking, means that we can access more than 64k of memory for something more than simply a RAM disk.

One of the first attempts to tackle the 64k memory barrier was Digital Research with CP/M Plus (aka CP/M3). While it banked both portions of the BIOS as well as the Basic Disk Operating System (BDOS) and included some useful additions to the BIOS, it was relatively incompatible with CP/M 2.2 in many key features. In addition, CP/M Plus made no provision for banking application code. The adoption of a CP/M 2.2 standard for the Z-System has served to widen the compatibility gap even further on the majority of our systems.

There are some internal inconsistencies in the CP/M 3 architecture as well which were never fully resolved. A prime case in point is the function to return Disk Free Space

(Function 46). The specification states that three bytes are returned reflecting the number of available 128-byte records on a disk. This equates to 2^24 * 128 or 16,777,216 * 128 = 2,097,152 kB. While we know of no one who has actually installed a single disk partition of more than 2 Gigabytes on a Z-System, it would create problems since CP/M Plus can handle disks up to eight times this size, but not correctly report free space. Simply returning free space in terms of 128-byte records is inconsistent as well since disk space is allocated in blocks which have a minimum size of 1k, with 2k and 4k commonly used. This is only one example of several, and we do not consider it a viable system for future Z-System growth; although it is still being actively installed.

Several manufacturers have attempted to bank portions of operating system software over the years, yet either locked the software into their hardware as Epson did with the QX-10, or made such changes as to limit portability of common tools as in the XLM-180. This latter system, while it used the Hitachi 64180 processor with its memory mapping capabilities, required system tailoring of much of the common Z-System software base.

The release of MicroMint's SB-180 in the early 1980s marked a decision point in Z-System development. First, it retained all standard ZCPR3 definitions, it used a CP/M 2.2 compatible BDOS, and it forced programmers to think more of portability and compatibility in system software. This was a major thrust in the development of XBIOS which placed the greatest possible amount of BIOS code in alternate memory locations outside of the primary 64k address space. Furthermore, capabilities to bank additional features (Resident System eXtensions, or RSXs) were widely supported with DateStamper, DosDisk and others requiring no sacrifice in TPA to execute. Since its last upgrade, however, several severe problems have come to light, among them are the inability to properly handle hard disk sizes greater than 32MB and sluggish performance due to banking of Console routines.

We considered this history and wanted to develop a system which included as much machine independence as possible. Not only should newer systems with the 64180/Z180 processor be included, but S100 systems with banked memory, addon boards such as Terry Hazen's MDISK for the Ampro Little Board, and homebrew systems as well. The goal here was as much selfish desire as anything else. By developing a single common architecture, only one tool for a given purpose would be needed across a variety of machines. As an example, Hal has several YASBECs, two SB-180s (one modified with static memory), an SB180FX, an Ampro Little Board, a couple of mongrel S-100 systems, and recently acquired a P112. Each system had its own Formatter, Configuration utility, Clock type, native disk formats, etc. To us, this seems out of place now, particularly with the scarcity of systems programmers in the Z-Community. It made more sense to develop a common software architecture so that more programming resources could be devoted to applications type efforts.

Another goal of the effort was to retain the maximum compatibility with existing Z-System software for the same reasons cited above. Customizing a huge number of common utilities as was done in the XLM-180, seemed to be the wrong approach. We therefore decided to retain the greatest possible commonality with CP/M 2.2 (actually our ZSDOS), and use existing Z-System segments to their greatest potential without sacrificing performance. As those of you who tracked our efforts as we developed ZSDOS know, we do not like slow systems or large code sizes (TCJ issues 37 and 38). We also decided that our architecture had to be capable of outrageous expansion and extension capabilities without invalidating previous software efforts. Further, we wanted to create a general purpose banked memory interface that allows applications programs as well as the operating system to access alternate banks of memory. The final results are the Banked & Portable (B/P) Bios.

B/P Bios attacks the memory problem in a manner which is easily adaptable to different hardware. All HD64180/Z180-based systems bank memory in 4k slices, and many S100 and addon systems bank in 16 or 32k increments. We therefore decided on an architecture which retains common memory in the upper 32k of address space (8000-FFFFH), and switches the lower 32k (0-7FFFH) among any available banks of RAM. Figure 2 displays this architecture pictorially.

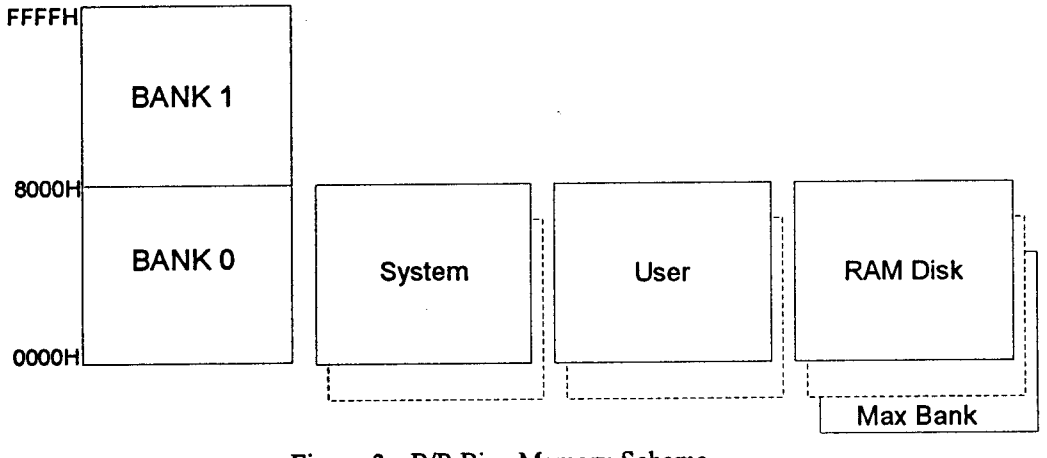


Figure 2. B/P Bios Memory Scheme

BNK1 is *ALWAYS* present in the address space and is referred to as the *Common Bank*. It contains all Z-System buffers, Common portions of the Bios, BDOS and the Command Processor as well as the upper portion of TPA. Part of the Bios which makes B/P unique is the structure which allows controlled access to other banks in the lower 32k.

At least one 32k bank is required in a minimal banked B/P system. The system bank, as a minimum, holds portions Bios and a copy of the Command Processor which speeds Warm Boots by simply copying the banked code to the Common bank and executing the warm entry. Figure 3 depicts memory use in a maximally-configured banked system. In such configurations the System bank holds banked portions of; the ZSDos2 operating System, banked Command Processor, BIOS, and Hard Disk allocation bit maps.

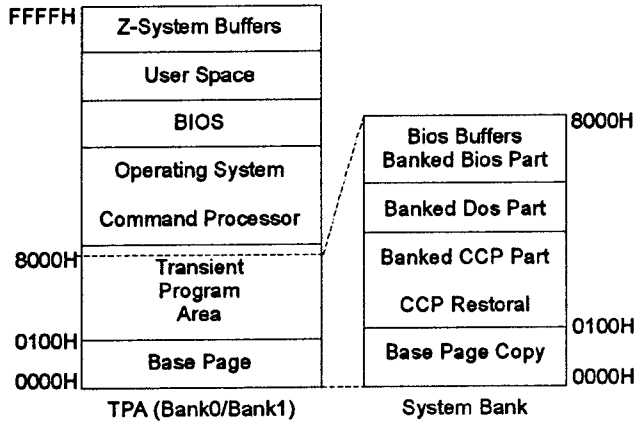


Figure 3. Fully-banked Memory Map

The B/P Bios began with one of Cam's superb architectures. He started with the standard CP/M 2.2 Jump Table, added in those from CP/M Plus with changes to correct some of the inconsistencies, then added in a new series to permit logical and easy access to new routines. The code, in assembly source form, was divided into logically functioning elements, with the greatest possible amount placed into machine independent modules. As an example, the Disk deblocker and IOBYTE decoder functions are machine independent and need no change between systems, while the actual disk and character device drivers require custom tailoring for each type of computer. Standard interfaces, in terms of register usage and value limits, result in common software requirements across vastly different hardware systems.

Each software module of the Bios includes relocation directives to the assembler telling it whether the code is to go into the main memory, or into another bank of memory. If a non-banked system is assembled, all code is placed into the main 64k area, while banked systems use the main 64k area for common code and data as well as banked code and data areas. All tools (formatter, configuration utility, etc) automatically accommodate both types of Bios without any intervention. The choice of whether to bank each section of code or not was painstakingly examined for performance and size penalties. Many of our design choices may be debated, but B/P is here and it works!

By itself, fixing the Bios is not a complete answer to our current Z-System dilemma, but it is a prerequisite. We also considered it essential to bank the Disk Operating System. Starting with ZSDOS, Hal used the new BIOS banking functions to bank significant portions of the BDOS. At every juncture, size and speed were traded off to keep the system small and fast. We corrected the flaw in the CP/M Plus Disk Size function cited above by returning four bytes containing the number of kilobytes free (a more meaningful measure). File Time/Date stamping functions for DateStamper(tm), P2DOS (CP/M Plus compatible), and Joe Wright's NZTIM are all embedded within the Operating System and are concurrently active. While ZSDos2 is still a work in progress, copies are included in the B/P Bios package to allow users to benefit from the newer system. Latest versions were posted in the BPBIOS: directory on the Ladera Z-Node until its demise, and no replacement is yet available. The most recent version of ZSDos2 in 1993 added directory hashing for true speed demons.

Also incorporated in ZSDos2 is a ZCPR34-compliant Command Line Parser. With the permission of Jay Sage, we also modified ZCPR34 to operate in the banked environment, added many common features of Resident Command Processor packages, and simplified it to use the new ZSDos2 Command Line Parser. By folding all features into the CPR, the need for a Resident Command Processor (RCP) extension in high memory all but disappears, typically adding 2k to the TPA.

As a closing note to this first installment, systems using B/P Bios, ZSDos2 and the expanded CPR are currently in operation on the new D-X Designs Pty Ltd P112, Micromint's SB180, a modified SB180 with static memory, two versions of the SB180FX, three YASBEC configurations including a laptop with VGA LCD display, Ampro Little Board (with Terry Hazen's MDISK expansion), and in non-banked mode on a Compu/Time S100 and Teletek. All tools are common across the hardware systems with a complete Z3 Environment and additional segments typically resulting in the equivalent of a "standard" 60k CP/M system. This size is without the RCP typically used to enhance the Command Processor.

The next part of this article will describe the B/P Bios interfaces and standards, while part 3 will cover proposed standards for the evolving ZSDos2, its associated Command Processor and some of the support utilities.

Cameron "Cam" Cotrill is a Senior Software Engineer with Symantec.

Harold "Hal" Bower retired from the US Army Signal Corps in 1990 and has been active in microcomputing since 1976.

Special Feature

All Readers

PC Terminal Programs

Program This!
The PC Serial Ports

By Dave Baldwin

The Chips

PC serial ports are the result of decisions made in 1980 and never changed since then. The 8250 UART from National Semiconductor was chosen for the original PC's. It is a simple, single-buffered design that is only slightly more advanced than the old AY-5-1013 hardware uart. It has a bunch of extra bus control signals designed to make it 'easy' to interface to just about any CPU. It also has all of the modem control signals available along with two undedicated outputs.

Other designs of the time were double buffered (like the 8251) or triple buffered (like the Z80-SIO) meaning that you could wait longer to read the received characters without losing data. These other chips had limits of their own, however. Neither had all the modem control signals, the 8251 has to have CTS true to transmit and the SIO is designed to work with the Z80 interrupt structure.

Newer versions of the 8250 have been introduced including the 8250A, the 16450, and the 16550. The 8250A fixed a couple of errors in the original chip, the 16450 provided a faster access time, and the 16550 included 16-byte receive and transmit fifos. There are also CMOS versions, the 82C50A and the 16C450. All of them power up to act like an 8250 (almost). The fifos on the original 16550 didn't work, so a 16550A version was introduced with working fifos. 16650 and 16750 versions with larger fifos are also available now from other manufacturers.

The Programs

Two programs are presented here, one in ASM86 and one in 'C'. A third program, SIMPLTRM.ASM will be available on the BBS and the TCJ Web

```
/* SCTERM - Simple C Terminal program, June 1997, by Dave Baldwin
    See Issue #80 of The Computer Journal for more info
    about this program.  Names of the equivalent labels in
    SIMPLTRM and DOSTERM are in parentheses here.
    TCJ Web page: "http://www.psyber.com/~tcj/"
    Phone: 800-424-8825

    Written for BorlandC 3.1 */

#include <stdio.h>
#include <dos.h>
#include <conio.h>

#include "SCTERM.H"

#define __CPPARGS
/* prototypes */
unsigned int keychk(void) ;
unsigned int getkey(void) ;
void interrupt ( *oldvect)(__CPPARGS);/* pointer to save old vector */
void interrupt com_isr(__CPPARGS) ;   /* interrupt service routine */
void getcmd(void) ;                   /* get function key cmd*/
void com_enb (void) ;                 /* com-port set up */
void com_dsb (void) ;                 /* com-port disable */
char com_in(void) ;                   /* get receive char from buffer*/
void com_out(char pch) ;                    /* send char to serial */
/* variables */
char sch ;                      /* character from serial port buffer */
unsigned int kch,kchx ;         /* characters from keyboard */
int xflag = 0 ;                 /* program exit flag */
char int_mask ;                 /* old interrupt mask */
char new_mask ;                 /* new interrupt mask */
unsigned pic_mask = 0x21 ;      /* 8259 mask port address */
unsigned pic_eoi = 0x20 ;       /* 8259 End-Of-Interrupt port address */
char recerr ;
int i,j,k ;                     /* miscellaneous ints */
char portnum = 0 ;
char com_buf[BUFSIZE] ;         /* receive buffer */
unsigned buf_hed ;              /* buffer char in index */
unsigned buf_tal ;              /* buffer char out index */
char signon[] = "SCTERM using COM" ;
char signon2[] = ", F10 to exit.\r\n" ;
char tcjmsg[] = "From The Computer Journal, (800) 424-8825\r\n" ;
char exitmsg[] = "SCTERM terminal program in C.\r\n" ;
char helpmsg[] = "SCTERM/x where x = 1-4 for COM1 thru COM4.\r\n" ;

/* ======================================================== */
int main(int argc, char *argv[])
{
/* force value for test */
comp.base = 0 ;
/* (init01) command line check here */
if (argc==2)
    {
    /* check for port number */
    i=0;
    k=0;
    while (argv[1][i])
         {
         if (argv[1][i++] != '/')
              {
              continue;
              }
         portnum = argv[1][i] ;
         break ;
         }
    }
```

```
if (portnum=='4') {
    comp.base = COM4;          /* (initab) */
    comp.intr = COM4IRQ;
    comp.imsk = COM4MSK;
    }
if (portnum=='3') {
    comp.base = COM3;          /* (initab) */
    comp.intr = COM3IRQ;
    comp.imsk = COM3MSK;
    }
if (portnum=='2') {
    comp.base = COM2;          /* (initab) */
    comp.intr = COM2IRQ;
    comp.imsk = COM2MSK;
    }
if (portnum=='1') {
    comp.base = COM1;          /* (initab) */
    comp.intr = COM1IRQ;
    comp.imsk = COM1MSK;
    }
/* default value */
if (comp.base == 0) {
    portnum = '2';
    comp.base = COM2;   /* (initab) */
    comp.intr = COM2IRQ;
    comp.imsk = COM2MSK;
    }
/* (comptr) initialize address variables for comm port registers */
comp.ier = comp.base+IER ;
comp.iir = comp.base+IIR ;
comp.lcr = comp.base+LCR ;
comp.mcr = comp.base+MCR ;
comp.lsr = comp.base+LSR ;
comp.mst = comp.base+MST ;
comp.scr = comp.base+SCR ;
comp.baudlo = comp.base ;
comp.baudhi = comp.base+1 ;
/* initialize memory copies of register values for com port */
creg.baud = 12 ;
creg.lcr = 3 ;
creg.mcr = 0x0b ;

/* (init03) clear screen, setup serial port and interrupt vector */
clrscr();
com_enb() ;
/* display signon messages */
cputs(signon) ;
putch(portnum) ;
cputs(signon2) ;
cputs(tcjmsg) ;

/* (main) ======== MAIN PROGRAM LOOP ========== */

while (xflag == 0)
    {               /* while the exit flag is zero */
    /* check for keyboard input */
    if (keychk() & 0xff) {
         kch = getkey() ;          /* get char from keyboard */
         if (kch & 0xff) com_out(kch);/* if the key input notzero, */
                                      /* send it to the serial port */
         else getcmd();            /* get function key command */
         }
    /* check for serial input */
    if (buf_tal != buf_hed)  {     /* compare head and tail ptrs */
         sch = com_in() ;          /* get char from serial buffer */
         if (sch) putch(sch) ;     /* display it on the screen */
         }
    }

/* (prgxt) ======== Program EXIT ========== */
com_dsb() ;                            /* turn off serial port */
clrscr() ;
cputs(exitmsg) ;
cputs(tcjmsg) ;
return 0 ;
}

unsigned int keychk(void) {
union REGS regs;
int c ;
regs.h.ah = 1;  /* check for key ready */
int86(0x16, &regs, &regs);
c = regs.x.flags & 0x40 ;
c ^= 0x40 ;
return c ;
}
```

pages. The source code for it is too long to fit in the magazine. I may offer all the code from this issue on a disk if people ask for it.

These programs will work on any PC from an XT to a Pentium and are simple enough that you can modify them for your own uses. The ASM programs (DOSTERM and SIMPLTRM) can be assembled with MASM 4.0 or 5.1, Borland's TASM 3.0, or the public domain Arrowsoft x86 Assembler along with the Val linker. These last two were found on the Dunfield Development web site and are now on the TCJ Web pages and the TCJ/DIBs BBS so you can download them. The 'C' version was done in Borland C 3.1.

Since MS-DOS and the PC BIOS (like CP/M) don't do anything useful with the serial ports, each of these programs does it's own serial port setup. DOSTERM sets up the port for polled operation and uses MS-DOS for keyboard and display. If you have ANSI.SYS loaded, it gives you a very simple ANSI-PC terminal. SIMPLTRM and SCTERM set the port up for interrupt driven operation on receive and polled operation on transmit. This is the only 'universal' setup for interrupt operation. Transmit and status interrupts and bugs vary enough between the chips to require some testing and special handling in the interrupt service routines.

If you decide you want to use transmit and status interrupts, you need to get a copy of 'The Serial Port FAQ' by Christian Blum which is now available on the TCJ web site and BBS. There are problems with transmit interrupts disappearing on some chips and the interrupt response for multiple interrupts is different. An interrupt service routine that takes all of these into account is shown in The Serial Port FAQ and is many times longer than the one I'm using here.

I've put a couple of special 'features' in these programs. The ASM programs are set up so that they will run as EXE or COM programs. I did this when I got tired of having to run EXE2BIN before I could check the results of the next 'newer, better' version. This also helped when I started making larger versions that needed to be EXE only programs. Also, having the three dif-

ferent versions lets you compare what's been done and see the different ways of doing essentially the same thing. I've also tried to kept the names of the routines the same in all three versions so you could compare them. Another thing is that these are all small programs. Even the 'C' version is less than 10kbytes after it's compiled.

All the programs are organized the same way. After defining some values, they check the command line to see if you want COM1 instead of COM2 which is the default port. Then they set up an address table based on the address of the comm port you're going to use, enable the comm port, clear the screen, and put up the signon messages. The 'com_enb' routine is called to initialize the serial port. Since DOSTERM uses the MS-DOS console routines, the 'clear screen' is done as an ANSI escape sequence in the signon message.

Enabling the Serial port

Before you can use the serial port, you have to program the chip and set up the interrupt vector if interrupts are going to be used. SIMPLTRM and SCTERM use the MS-DOS functions to get the old interrupt vector and save it so it can be restored when the program exits and then set the vector to point to their own interrupt service routine, 'com_isr'. Next, all three programs program the serial chip registers to the values needed for operation.

The 8250 family has 9 or 10 addressable registers. The 'Scratch' register doesn't exist on the earliest versions. In ASM86, only the 8-bit I/O addresses from 0 to 255 are directly addressable. For 16-bit I/O addresses, you have to load the DX register with the address and input or output to/from the AL or AX register. Since the 8250 registers are all byte registers, we'll be using only AL.

The first thing we do is to disable all interrupts from the chip while we're programming the rest of the registers by setting IER (Interrupt Enable Register) to 0. Then we send an 83h to the LCR (Line Control Register) to select the baud rate registers. The 8250 has an internal baud rate generator with a 16-bit divider that can divide the baud rate clock with values from 1 to 65535. Divide by 0 is undefined as usual.

```
unsigned int getkey(void) {
union REGS regs;
regs.h.ah = 0 ;  /* get key */
return int86(0x16, &regs, &regs);
}

void getcmd() {
kchx = kch/256 ;   /* if it's zero, then get extended char */
if (kchx == 68)
    {              /* is it F10? */
    xflag = -1 ;   /* then set exit flag */
    }
if (kchx == 59)
    {                   /* is it F1? */
    clrscr() ;          /* then clear the screen */
    cputs(signon) ;
    putch(portnum) ;
    cputs(signon2) ;
    cputs(tcjmsg) ;
    cputs(helpmsg) ;
    }
}

/* COM_IN */
char com_in(void) {
char tch ;
tch = com_buf[buf_tal] ;          /* get byte from current buffer
position */
buf_tal++ ;                       /* adjust buf pointer */
buf_tal = buf_tal & BUFMSK ; /*  and mask and resave it for next
                                    time */
return tch ;                      /* return the char from the buffer */
}

/* COM_OUT */
void com_out(char pch) {
int cntr = 1000 ;  /* time-out counter to prevent lockup */
char tst ;
while (cntr-)
    {                         /* check time-out counter */
    tst = inp(comp.lsr) & 0x20 ;  /* test THRE */
    if (tst)
        {                     /* if true, send char */
        outp(comp.base,pch) ;
        cntr = 0 ;            /* set loop counter to zero */
        break ;                         /* and exit */
        }
    }
}

/* COM_ENB */
void com_enb (void) {
buf_hed = 0 ;             /* clear fifo pointers */
buf_tal = 0 ;
/* save the old interrupt vector */
oldvect = getvect(comp.intr) ;
/* install the new interrupt handler */
setvect(comp.intr, com_isr);
/* disable interrupts while we set up port */
disable() ;
/* set up serial port registers */
outp(comp.ier,0) ;            /* serial chip interrupts off */
outp(comp.lcr,0x83) ;             /* access baud rate registers */
outp(comp.baudlo,creg.baud) ;     /* divide by 12 for 9600 baud, low
                                     byte */
outp(comp.baudhi,creg.baud >> 8); /* high byte of baud rate divider */
outp(comp.lcr,creg.lcr) ;    /* set 8N1 */
outp(comp.mcr,creg.mcr) ;         /* turn on DTR, RTS, and OUT2 */
/* do dummy reads to clear flags */
inp(comp.base) ;
inp(comp.lsr) ;
outp(comp.ier,1)   ;              /* serial receive interrupts only */
/* re-enable interrupts now */
enable() ;
/* set up interrupt mask in PIC */
int_mask = inp(pic_mask);
new_mask = int_mask & (!comp.imsk) ;
outp(pic_mask,new_mask) ;
}

/* COM_DSB */
void com_dsb (void) {
outp(comp.ier,0) ;      /* serial chip interrupts off */
outp(comp.mcr,0) ;      /* modem control lines off */
/* restore the old interrupt mask */
int_mask = inp(pic_mask);
new_mask = int_mask | comp.imsk ;
```

```
outp(pic_mask,new_mask) ;
/* restore the old interrupt handler */
setvect(comp.intr, oldvect);
}

/***NOTE:
    This is an interrupt service routine.  You can NOT compile this
    program with Test Stack Overflow turned on and get an executable
    file which will operate correctly. */
#pragma -N-        /* stack checking off */
/* COM_ISR */
void interrupt com_isr(__CPPARGS) {
recerr = inp(comp.lsr) ;                    /* read/clear error byte */
com_buf[buf_hed] = inp(comp.base) ;  /* read data byte */
buf_hed++;                                  /* adjust and mask pointer */
buf_hed = buf_hed & BUFMSK ;
outp(pic_eoi,0x20) ;                        /* send EOI to 8259 */
}
```

```
/* SCTERM.H - header file for SCTERM.C */
/* COM port and buffer equates */
#define COM1 0x03f8          /* for COM1 */
#define COM1IRQ 0x0c
#define COM1MSK 0x10
#define COM2 0x02f8          /* for COM2 */
#define COM2IRQ 0x0b
#define COM2MSK 0x08
#define COM3 0x03e8          /* for COM3 */
#define COM3IRQ 0x0c
#define COM3MSK 0x10
#define COM4 0x02e8          /* for COM4 */
#define COM4IRQ 0x0b
#define COM4MSK 0x08

#define BUFSIZE 0x2000  /*receive buffer size, must be power of 2 */
#define BUFMSK BUFSIZE-1     /* 1 Less for mask */

#define  BASE 0    /* Base (No real reason to have this) */
#define IER     1   /* Interrupt Enable Register */
#define IIR     2   /* Interrupt ID Register */
#define LCR     3   /* Line Control Register */
#define MCR     4   /* Modem Control Register */
#define LSR     5   /* Line Status Register */
#define MST     6   /* Modem Status Register */
#define SCR     7    /* Scratch Register */
#define RTS   2     /* rts bit in MCR */

struct comdef {     /* addresses of port registers plus IRQ and mask */
int base ;
int ier ;
int iir ;
int lcr ;
int mcr ;
int lsr ;
int mst ;
int scr ;
int baudlo ;
int baudhi ;
char intr ;
char imsk ;
} comp;

struct COMREG {
unsigned int baud ;
unsigned char lcr ;
unsigned char mcr ;
} creg ;
```

The baud rate registers actually use the same addresses as the data and IER registers, but they're in a different 'bank'. You select the baud rate registers by setting bit 7 or the LCR to a one. The standard PC baud rate clock oscillator is 1.8432 MHz. To get 9600 times 16 (153.6 KHz) you divide by 12. You put 12 in the lower baud rate register and zero in the high register.

Next we load the LCR with 03h which unselects the baud rate registers and sets the chip to 8 data bits, no parity, and 1 stop bit. The MCR (modem control register gets loaded with 0bh for the interrupt version and 03h for polled. The OUT2 bit has to be turned ON in PC serial ports to enable the interrupt gate. This is usually a 74LS125 which is in series with the INTR line from the chip and the selected IRQ line on the bus.

After that, we do some dummy reads of a couple of registers to clear any flags that might have been accidently set. For the polled version in DOSTERM, this is all that's needed. For the interrupt versions, we still have to set the IER to allow receive interrupts and set the appropriate mask bit in the 8259 PIC (Programmable Interrupt Controller).

The disable routine, 'com_dsb', does the opposite of the enable routine. It restores the old interrupt vector and turns off the serial chip interrupts in SIMPLTRM and SCTERM and it turns off the modem control lines in all of them. SIMPLTRM and SCTERM also restore the previous interrupt mask to the 8259.

The Main Loop

The main program loop checks for a keyboard input to see if you want to send anything and then it checks to see if you have received anything. DOSTERM actually polls the serial chip each time. SIMPLTRM and SCTERM check the receive buffer to see if the receive interrupt service routine has put any characters in the buffer.

If there is a keyboard input, it is checked to see if it's an 'extended' char code or a normal character. Normal characters from 1 to 255 are written to the serial

port. Extended codes are checked for the two commands, F1 and F10, that are used by these programs. F1 is the 'help' command which redisplays the signon info and F10 is the exit command.

If there is an input from the serial port, it is sent to the screen. DOSTERM lets MS-DOS handle the control characters and escape sequences while SCTERM let's the video routines built in to the Borland 'C' compiler do it. SIMPLTRM has routines to do the control code and escape sequence processing to emulate the basic functions of an ADM3 terminal and uses the PC BIOS routines for video output.

That's really all there is. Each of these programs can be used as is or easily modified to your own needs. They were left as simple as they are so they would be reasonably easy to understand.

One of the limits on these programs is speed. Don't expect to go above 19.2 KBaud (9600 for DOSTERM) without errors. In 'standard' configurations for PCs, it turned out that everything from a 4.77MHz XT to a 486DX2/66 started getting errors above 19.2Kb. The XT just plain ran out of steam while the background processing caused problems on the 386 and 486. It turns out that background programs like SMARTDRV link into to many of the standard DOS and BIOS routines and use up enough CPU time to limit the interrupt servicing. When I booted a 386 in a minimum software configuration, it would run without error at 57.6Kb with these simple interrupt routines.

An improved version of SIMPLTRM is available for download on the TCJ FTP site and the TCJ/DIBs BBS. Called DIBSTERM.COM, it let's you select which serial port you want to use, tell's you what kind of serial chip the port has, and allows you to change the port while the program is running. It also puts the modem line status at the top of the screen so you can monitor their status and lets you control the outputs with function keys. You can also select the baud rate (from 50 to 115200) and other parameters along with the handshaking mode. It's a good testing program as well as a simple terminal program.

Dave Baldwin

```
;   DOSTERM.ASM - a simple terminal emulator for PC's
;   TCJ Web page: "http://www.psyber.com/~tcj/"
;   Phone: 800-424-8825
;
;   Uses MS-DOS console functions
;   Uses the PC serial port in polled mode.
;
;   Can be assembled by MASM 4.0 or 5.1, Borland's TASM,
;    or the Arrowsoft public domain assembler and Val linker.
;   This is a 'COM' program that can also run as an 'EXE'.
;   Look for '***' to see what's required to do this.
;   There can only one segment, the code (CS) segment.
;
; EQUATES
false   equ  0    ;
true    equ  not false
cr  equ  0dh  ;
lf  equ  0ah  ;
; duplex mode
echo    equ  false     ;false = full duplex, true = half-duplex
;standard hardware handshake
rts     equ  2    ;rts bit in com_mcr
handshake     equ  true ;include handshake code if true
;COM port equates
com1          equ  03f8h     ;for COM1
com2          equ  02f8h     ;for COM2
com3          equ  03e8h     ;for COM3
com4          equ  02e8h     ;for COM4
;=============================================================
; Macros for handshaking routines
; These are used anywhere that video scroll can occur because
;  the video BIOS routines turn off interrupts when they scroll.
RTSOFF   MACRO
; rts OFF
IF handshake
    push dx
    push ax
    pushf
    mov  dx,com_mcr
    in   al,dx          ;
    and  al,not rts     ;rts OFF
    out  dx,al          ;
    popf
    pop  ax
    pop  dx
ENDIF
ENDM
;
RTSON    MACRO
; rts ON
IF handshake
    push dx
    push ax
    pushf
    mov  dx,com_mcr
    in   al,dx          ;
    or   al,rts         ;rts ON
    out  dx,al          ;
    popf
    pop  ax
    pop  dx
ENDIF
ENDM
;=================================================================

cseg     segment para public 'CODE'

    org  100h                ;COM program

    assume    cs:cseg,ds:cseg,es:cseg,ss:cseg ;COM program

dosterm proc  far          ;entry point from DOS
; *** set segment reg's for EXE/COM version, all same segment
    mov  ax,cs          ;
    mov  es,ax          ;make SS and ES = CS, leave DS = PSP
    mov  ss,ax          ;
    mov  sp,-2          ; set stack pointer
; *** save psp pointer in real stack along with a '0'
    push ds        ;
    xor  ax,ax          ;
    push ax        ;
;get command tail
    mov  bx,80h         ;point to command line
                   ; (uses DS as PSP pointer)
    mov  al,[bx]   ;get first byte which is char count
    cmp  al,0      ;if 0 (no chars),
    je   init0b         ; use default port (com2)
```

```
;check the rest of the command line
init01: inc   bx        ;next char
    mov  al,[bx]   ;
    cmp  al," "         ;
    je   init01         ;skip spaces
;
    cmp  al,"/"          ;first non-space char must be slash
    jne  init0b         ; or else use default port (com2)
    inc  bx        ;next char
    mov  al,[bx]   ;
    cmp  al,'?'         ;
    jne  init02         ;jmp if not '?',
    mov  dx,offset errmsg ; else show 'errmsg'
    jmp  prgext         ;  and exit
;
init02: cmp   al,"1"                    ;is it COM1?
    jb   init0b             ;if not then use COM2
    cmp  al,"4"                 ;
    jbe  init0c
init0b: mov   al,"2"                    ;set default comm number
init0c: mov   byte ptr com_num,al ;save ASCII comm number
; *** Now set DS = CS because we're done with PSP
    mov  dx,cs
    mov  ds,dx
; set com port address table according to 'com_num' in AL
    call comptr
;
;put up signon message
init03: mov   dx,offset signon    ;get signon message
    mov  ah,09h             ;display string function
    int  21h            ;let dos do it
    mov  dl,byte ptr com_num ;get comm number
    mov  ah,06h             ;display char function
    int  21h            ;let dos do it
    mov  dx,offset signon2   ;get signon message
    mov  ah,09h             ;display string function
    int  21h            ;let dos do it
;
    call com_enb        ;start up communications port
;=================================================================
;
;   Main program loop
;
;=================================================================

main:    mov  dl,0ffh   ;
    mov  ah,6       ;direct input
    int  21h        ;ask DOS for keyboard input
    jz   main4          ;no char, skip
    cmp  al,0       ;is it extended key code?
    je   main5          ; Yes, go process

main2:
    if   echo
    push ax         ;if running half-duplex, echo char
    call vidout         ; to PC display
    pop  ax         ;
    endif
;
    call com_out   ;
;-----------------
;(com_inp) check if char waiting at comm port
main4:
    mov  dx,com_lsr     ;get receive status
    in   al,dx          ;
    and  al,1       ;check receive data ready flag
    jz   main       ;no, loop
;
    mov  dx,com_base    ;data port
    in   al,dx          ;
    call vidout         ;write it to PC display
    jmp  main4          ;see if any more waiting
;-----------------
; check function keys
main5:
;get second byte
    mov  dl,0ffh   ;direct input
    mov  ah,6       ;ask DOS for keyboard input
    int  21h        ;
;
    cmp  al,59          ;F1?
    jne  main6          ;
; F1 (Help) message
    RTSOFF
    mov  dx,offset signon    ;get signon message
    mov  ah,09h             ;display string function
    int  21h            ;let dos do it
```

References:

Advanced MS-DOS
Ray Duncan
Microsoft Press

ASM and C programs for MS-DOS. I started out by copying the TALK.ASM program from this book.

The Peter Norton PC Programmer's Bible
Norton, Aitken, Wilton
Microsoft Press

The 'standard' refence for beginning and intermediate PC programmers.

MS-DOS Programmer's Reference, Version 5
Microsoft Press

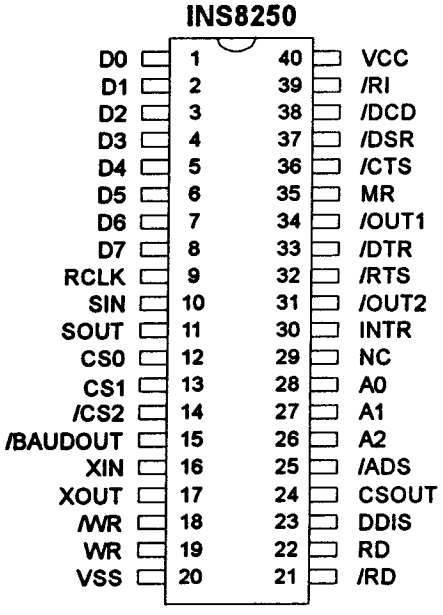
Extensive listing of all the MS-DOS functions with short sample routines and all of the structures used by MS-DOS up to version 5.0.

The_Serial_Port_FAQ
Christian Blum

Look here for routines for detecting and using the 16550 uart with fifos for high-speed communication. Available on the TCJ FTP site.

National Semiconductor Data Communications Handbook

8250 thru 16550 data sheets. Also available from their web site as Adobe Acrobat PDF files. "http://www.natsemi.com"



Pins:

1-8	D0 thru D7, data lines
9	RCLK - Receiver clock
10	SIN - serial input
11	SOUT - serial out
12-14	CS1-3 - chip selects
15	/BAUDOUT- Oscillator output
16	XIN - crystal input
17	XOUT - crystal output
18-19	WR - write control
20	VSS (gnd)
21-22	RD - read control
23	DDIS - driver disable
24	CSOUT - chip select out
25	/ADS - address strobe
26-28	A0-A2 - address lines
29	NC
30	INTR - interrupt output
31	OUT2 - used to enable PC interrupts
32-34	Modem/control outputs
35	MR - Master reset
36-39	Modem/control inputs
40	VCC - +5V

```
    mov  dl,byte ptr com_num ;get comm number
    mov  ah,06h              ;display char function
    int  21h            ;let dos do it
    mov  dx,offset signon2   ;get signon message
    mov  ah,09h              ;display string function
    int  21h            ;let dos do it
    mov  dx,offset tcjmsg ;restore message pointer
    mov  ah,09h         ;display string function
    int  21h            ;let dos do it
    RTSON
;
main5m:
    RTSOFF
    mov  dx,offset errmsg      ;get message offset
    mov  ah,09h         ;display string function
    int  21h            ;let dos do it
    RTSON
    jmp  main      ;

main6:   cmp  al,68     ;F10?
    je   mainx          ;yes, exit
    jmp  main      ;if not, ignore and loop back to 'main'
;
mainx:   mov  dx,offset exitmsg  ;print farewell message
;-----------
;exit procedure
prgext:
; *** set DS in case of error exit
    mov  ax,cs          ;
    mov  ds,ax          ;
    mov  es,ax          ;
;
    push dx         ;save message address
    call com_dsb    ;disable controller and release int vec
    pop  dx         ;
;
    mov  ah,9       ;let DOS print message
    int  21h        ;
;
    mov  ax,4c00h  ;exit with ret code = 0
    int  21h
;
    ret             ;far return just in case
    int  20h        ;just in case
;
dosterm endp
;=========================================================================
vidout   proc near      ;write char in AL to display
    RTSOFF
    mov  dl,al          ;
    mov  ah,6       ;use DOS DIRECT console
    push bx         ;save bx
    int  21h        ;call DOS
    RTSON
    pop  bx
    ret             ;
vidout   endp
;=========================================================================
; send char in AL to serial port
com_out proc near
    push ax         ;save char
    mov  dx,com_lsr     ;check tbe stat
    mov  cx,2000    ;anti-lockup
com_out1:
    in   al,dx          ;
    and  al,20h         ;THRE
    jnz  com_out2  ;send if ready
    loop com_out1  ;loop until ready
    pop  ax         ;restore char
    ret             ;
com_out2:
    pop  ax
    mov  dx,com_base
    out  dx,al          ;write char to port
com_oxt:
    ret
com_out endp
;=========================================================================
; set up comm port
com_enb proc near

    mov  dx,com_ier     ;interrupt enable register
    mov  al,0       ; no interrupts
    out  dx,al          ;

    mov  dx,com_lcr     ;set line control
    mov  al,83h         ;select baud reg, set parms below
```

```
    out  dx,al          ;

    mov  dx,com_base    ;set 9600 baud
    mov  al,0ch         ;lo byte
    out  dx,al          ;
    inc  dx        ;point to high byte
    xor  al,al          ;high byte is 0
    out  dx,al          ;
; set r/t parms
    mov  dx,com_lcr     ;
    mov  al,03h         ;unsel baud reg, no parity, 1 stop, 8 bits
    out  dx,al          ;

    mov  dx,com_mcr     ;modem controller DTR, RTS,
    mov  al,03h         ;
    out  dx,al          ;

    mov  dx,com_base    ;dummy read of serial port
    in   al,dx          ; to clear flag

    mov  dx,com_lsr     ;
    in   al,dx          ;dummy read to clear stat flags
;
    ret
com_enb endp
;=========================================================================
; turn modem control outputs off
com_dsb proc near

    mov  dx,com_mcr     ;
    mov  al,0      ;turn off all mcr outputs
    out  dx,al          ;
;
    ret
com_dsb endp
;=========================================================================
; gets base pointers and updates I/O address table
comptr   proc near
    mov  bx,offset port1     ;offset to port1
    cmp  al,'1'                  ;
    jz   comset              ;
    mov  bx,offset port2     ;offset to port2
    cmp  al,'2'                  ;
    jz   comset              ;
    mov  bx,offset port3     ;offset to port3
    cmp  al,'3'                  ;
    jz   comset              ;
    mov  bx,offset port4     ;offset to port4
    cmp  al,'4'                  ;
    jz   comset              ;
    mov  dx,offset errmsg ;EXIT because no COM specified.
    xor  ax,ax               ;
    ret
;load I/O address table for selected com port
comset:
    mov  ax,[bx]        ;get base
    or   ax,ax               ;check for zero
    jz   comptx              ;zero error exit
    mov  word ptr com_base,ax     ;
    mov  word ptr baudlo,ax  ;
    inc  ax                  ;+1 for ier/baudhi
    mov  word ptr com_ier,ax ;
    mov  word ptr baudhi,ax  ;
    inc  ax                  ;
    inc  ax                  ;+2 for lin
    mov  word ptr com_lcr,ax ;
    inc  ax                  ;+1 for mcr
    mov  word ptr com_mcr,ax ;
    inc  ax                  ;+1 for stat
    mov  word ptr com_lsr,ax ;
    inc  ax                  ;+1 for mst
    mov  word ptr com_mst,ax ;
comptx: ret
;
comptr   endp
;=========================================================================
;com port address assignments
port1    dw   com1       ;port address
;
port2    dw   com2       ;port address
;
port3    dw   com3       ;port address
;
port4    dw   com4       ;port address
;
```

More advanced programs

In more complete terminal emulations and file transfer programs, the serial code is a small part of the program. Terminal emulations require direct video routines because the PC BIOS routines are inadequate and too slow, file transfers require a lot of code for the different protocols, and both require code to control the modem.

There are several other problems that need to be dealt with on PC's. You may need a 'CPU ID' routine to tell you what kind of machine your program is running on. Plain PCs and XTs only have the low order interrupts available (IRQ0-7) while the AT/386/486/ Pentiums have IRQ's 0 thru 15 available. If your program needs to be able to use high order IRQs (9-15), you will need a slightly different interrupt service routine for your receive interrupt.

On a PC, you also need to provide a 'Critical Error Handler' routine so that disk file problems don't crash your program. The 'default' action of the Critcial Error Routine (Abort, Fail, Retry?) will abort your program and return to DOS if you don't provide a routine that gives your program control.

You also need to 'intercept' the timer interrupt to eliminate software timing loops. Since fixed software timing loops will run at drastically different rates on different PC's, using the timer interrupt will allow you to have consistent timing on different machines.

Another problem is 'handshaking' on the serial ports. Handshaking is used to start and stop the flow of data to allow the programs to do something else for a moment like reading or writing files or scrolling the screen. DOSTERM shows very simple hardware handshaking (RTSOFF/RTSON). Some situations require other modem control lines for handshaking and there are several different kinds of software handshaking.

And then there's modems. They have to be sent 'AT' command strings in the proper order and you have code to recognize the responses to know what to do next. TCJ #79 has David Goodenough's article on the AT modem commands.

```
;program uses this table after setup
com_base dw   0    ;
com_ier  dw   0    ;interrupt enable reg
com_lcr  dw   0    ;line control
com_mcr  dw   0    ;modem control
com_lsr  dw   0    ;line status reg
com_mst  dw   0    ;modem status reg
baudlo       dw   0    ;baud divisor lo
baudhi       dw   0    ;baud divisor hi
tblsiz       equ $-com_base
com_num db    0         ;ASCII comm number
signon   db   27,"[2J"        ;ANSI 'clear screen'
    db   "DOSTERM using COM$",0
signon2 db    ", F10 to exit >",cr,lf,"$"

; Note '0' at end of first line for use by 'F1' / help function.
errmsg   db   "Type DOSTERM /x where 'x' is 1-4 for COM1 thru COM4.",cr,lf,0
    db   "Display must be text mode.",cr,lf,0,"$"

exitmsg db    cr,lf,'Exit from DOSTERM terminal emulator.',cr,lf
tcjmsg   db   'From The Computer Journal, (800) 424-8825',cr,lf
    db   'Web page: http://www.psyber.com/~tcj/',cr,lf,'$'

cseg ends

    end  dosterm
```

8250 Register Addresses

	0 DLAB=0	0 DLAB=0	1 DLAB=0	2	3	4	5	6	0 DLAB=1	1 DLAB=1
register bit	RECV buffer register	TRANS holding register	Interrupt Enable register	Interrupt ID register	Line Control register	Modem Control register	Line Status register	Modem Status register	Divisor Latch low	Divisor Latch hi
0	DB0	DB0	Recv data available	'0' if Interrupt pending	Word Length bit 0	DTR	Recv Data Ready	Delta CTS	Bit 0	Bit 8
1	DB1	DB1	Transmit holding reg empty	ID bit 0	Word Length bit 1	RTS	Overrun error	Delta DSR	Bit 1	Bit 9
2	DB2	DB2	Recv Line Status	ID bit 1	Number of Stop bits	Out1	Parity error	Trailing edge Ring Indicator	Bit 2	Bit 10
3	DB3	DB3	Modem Status	0	Parity Enable	Out2 (PC INT enable)	Framing error	Delta DCD	Bit 3	Bit 11
4	DB4	DB4	0	0	Even Parity Select	Loopback	Break Interrupt	CTS	Bit 4	Bit 12
5	DB5	DB5	0	0	Stick Parity	0	Transmit Holding register empty	DSR	Bit 5	Bit 13
6	DB6	DB6	0	0	Set Break	0	Transmit Shift register empty	Ring Indicator	Bit 6	Bit 14
7	DB7	DB7	0	0	Divisor Latch Access Bit (DLAB)	0	0	DCD	Bit 7	Bit 15

Real Computing

By Rick Rodman

32-Bit Systems

All Readers

Real-time Control

August in the Nation's Capital: ragweed pollen floats in the hot, sticky air. Gnats and mosquitos greet them that dare venture out from air conditioning and HEPA filter. It is a time for reflection, for contemplation of possibilities, for antihistamines, most of all for staying inside.

The Timer

As an old saying goes, when life hands you a lemon, make applesauce. Many times we look in the wrong place for a solution to what is really a simple problem. Part of my plans and thoughts regarding my Real-Time Control System center around avoiding utility waste, and one of my pet peeves is lights being left on - particularly in the basement, but also in bathrooms. This has led me to try various timer devices, X-10 modules, timer controllers, and RTCS. But now I've found a much simpler solution.

This is a device available from Marlin P. Jones Associates, called a "SSAC TS120A-966 Solid State Timer", which the catalog says "supplies 115VAC@1A max to the load for 30 min. then power is removed from the load". What it really does is, it turns off lights after 30 minutes. (Sorry, no shorter durations are available.) It does the job perfectly, it's simple to connect and it costs only $3.75.

The MPJA catalog is a candy store for experimenters, full of neat optoelectronics, power supplies, unusual switches, connectors and other exciting stuff. Maybe I'll finally get around to building the Chaos-Pandemonium Box.

Newton

As the Chinese say, a journey of a thousand miles begins with a single stumble. Some readers will remember Apple's other personal computer, the Newton, with its crummy handwriting-recognition technology. The Newton not only cost $800, but it required a souped-up Macintosh and an expensive development kit even to do the simplest programming - and, on top of all that, used its own proprietary language, a very weird object-oriented dialect called NewtonScript. In management, and in baseball, they say a project can survive one big mistake, might survive two, but three strikes and you're definitely out.

Well, now that monied interests have moved to greener pastures, there remains fertile ground for enjoyable experimentation. Used Newtons are available cheaply - sure, they're not the latest models, but there haven't been that many changes in the line. Better yet, an inexpensive development tool is available, Steve Weyer's Newt. Newt, and its companion Slurpee, let you do typing on a PC - or a Sun or Kaypro or anything at all, really - and compile on a Newton.

Once you realize that a Newton is not suitable for any significant data entry, you start thinking that there's a lot of stuff that'd be really neat to do. One thing I want to do on it is for my car, to keep records, perform milage calculations and remind me about oil changes.

A CD of Newton programs is available from AMUG. While there appears to be no source code on this CD whatsoever, there are plenty of programs, mostly shareware, which may do what you want to do. In fact, one program, called MPG2, does exactly what I described in the previous paragraph.

Apple Computer recently released a beta of the long-anticipated Newton Toolkit for Windows, which actually requires Windows 95 or NT, and you can download it from their Web site.

Probably I won't write much more about Newton topics unless our esteemed editor prompts me, but for the TCJ reader the relevant facts are: ARM RISC processor, 128K to 512K of internal RAM, 1, 2 or 4MB flash card, graphic display with stylus input, no disk or keyboard, one RS-232 serial port.

Java

They say you can lead a gift horse to water, but don't look him in the mouth. From an informal survey of books in local bookstores, it looks like Java is far and away the most popular programming language today, outpolling Visual Basic and C++ combined. So what is it? It's a compiler/interpreter, like BASIC-E, CBASIC, and UCSD Pascal. A web page containing an "applet" causes a browser to request that "Java Byte Code" (p-code) be sent, where a "Java Virtual Machine" (interpreter) runs it. Nothing particularly innovative about that, is there? Of course it has the usual object-oriented stuff like classes and methods that are de rigeur these days.

While people always rush to say "it's a lot like C++", this is

not true. It has none of C++'s hard-to-read double colons or overloaded operators or templates or virtual functions, features which nobody in their right mind would use anyway. Some of its declarations look more like Pascal.

The most important difference between Java and C or Pascal is that Java doesn't have any pointers. Although pointers use a quirky syntax in C and are a minefield for the unwary, they are actually the language's most powerful feature. Without pointers, we're reduced to pass-by-reference/pass-by-value, like we had to do way back in the Fortran days. In fact, because of Java's syntax, I think the best alternative name for it is... Object Ratfor.

Microsoft's motto is "Good things come to those who waste", and they hope to crush Java in their deadly embrace. They have announced a package called "Visual J++", which is a version of Java modified to support what are now called "Active X Objects" (formerly called "OCXs" and "OLE Custom Controls"). The name "J++" is supposed to look like "C++" (repeating the mistake that name makes), but Microsoft, in their usual narrow vision, didn't notice that there is a very popular language called J, a derivative of APL. Anyway, Visual J++ looks like a package that can be safely avoided.

In Windows 95 and in NT version 4, the once small, quick Write word processor, which has been getting bigger over time, is replaced by a new program called WordPad. From what I can see, it has nothing that Write didn't have. But now, thanks to Microsoft's "dogfooding", it uses MFC and takes ten times as long to load. Efficiency is evidently a dirty word at Microsoft. I wonder if Bill Gates, the hotshot programmer who cajoled a full BASIC interpreter into the Altair's limited RAM, knows or cares what schlock his minions are producing.

More on Small-C for embedded systems

Frank Wilson writes to say that he has used CUG 6809 Small C to do work on a Color Computer and on a small 6809 SBC, and fixed a lot of bugs in it. "The 'logical not' code generation portion screws up the stack. Also, I think the integer divide had a sign problem. Eventually the darn thing would self compile on the CoCo, and I also got it to compile on a 68HC11." He also says: "Maybe using a PC for cross development is cheating, but it simplifies things and makes fooling with obsolete processors a low cost option." I don't think it's cheating, Frank. After all, a PC has to be good for something, doesn't it?

He also mentions some cross assemblers which are on CUG's CD. Most of these are fairly crude. The ones I like best are the Frankenstein cross assemblers. When working with assembly code, what can seem very minor syntax variations can just drive you crazy, especially when it comes to macros. In the Unix world these things are often standardized, but in the wild-and-woolly PC world, you never know what to expect.

You've seen the movie, now get the OS

The Plan 9 distributed operating system, named for the movie "Plan 9 from Outer Space", is available from AT&T in source code form for $350.

Next time

The RTCS becomes more truly real-time as it meets the TW-523 X-10 interface and the Watson voice card. Plus, time and space permitting, the difference between 'sound' and 'voice' and how to overcome it. Remember, it's not what your computer has, but what you can do with it!

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

Marlin P. Jones Associates:
1 800 652 6733 or +1 561 848 8236
Timer, item # 4381-RL, $3.75
P.O. Box 12685, Lake Park FL 33403-0685

Steve Weyer (Newt Development Environment, etc.)
weyer@netaxs.com
17 Timber Knoll Drive
Washington Crossing PA 18977

AMUG CD Inc. (Totally for Newton CD #6, $24.95)
+1 602 553 0066 or -0144 fax
745 N. Gilbert Road #124-275, Gilbert AZ 85234

AT&T (Plan 9 operating system, $350): +1 908 577 2700
2 Paragon Way, Suite 400
Freehold NJ 07728

Cheap Hard Disk Controllers

by Allison Parent

Special Feature

8085 to PC-ISA

Hard Disk Interface

Adapting a PC/ISA card to an 8085.

I needed a hard disk for a 8085 project. The common solution is to use a host interface and controller. However, boards like the WD1002HDO are becoming scarce. I considered SCSI but I'd need a SCSI interface to drive a SCSI drive and I didn't have either of these either. I opted to look at a different source for a hard disk controller for my 8085 board. What I was trying to avoid was complex interfacing and spending money. The most obvious source was cheap and plentiful supply of 8bit ISA bus cards. I settled on the WD1002S-WX2A mostly because I had two that were good. They would interface to the ST225 and ST251 drives I have laying around.

The question is, what does it take to talk to it? That was helped along when a request for information yielded an article interfacing the slightly smaller WD1002S-WX1 to an SC84 Z180 system. The WD1002 ISA boards are interchangeable for the application, the only differences being board size and surface mount instead of dips. The next step was creating a 8085 to ISA 8bit adaptor. I had several articles that described this in painful terms. I didn't need the ISA bus for general use, I just wanted a hard disk controller. I rejected creating a copy of the ISA bus as being too complex. What I wanted was the 1002HDO! The problem was, how do I make the 1002S-WX2A look like a few simple ports to the CPU?

Going to the drawing board, I pulled out one of the boards and started looking at it closely with the specifications that I had available. It was a complete controller. The interface is fully decoded, buffered and it requires a known set of TTL compatible signals from the ISA bus. Without schematics for the card, it was, for all intents, a black box. Actually this is ok since we know what the inputs and outputs are and what goes on in the middle has been done for us.

The 1002S-WX2A responds to a standard address for I/O at 320h to 323h and has Eprom at C8000h to C9FFFh with the standard jumpers in place. The control signals were address, MemRD/, IORD/, IOWR/, DRQ, DACK, IRQ, AEN and Reset. Right off, I could see that this was a subset of the ISA bus which cut the interface task down some. Looking at the card in hope of getting some inspiration, I noticed that on the edge connector many bus signals did not even have contacts. An idea, how much of the interface could be eliminated? First off was the Eprom at C8000h. The system has enough Eprom of it's own and clearly this was excess. This eliminated the MemRD\ and A10 through A19. This was a start. I then looked at how to put the board into an eight bit address space.

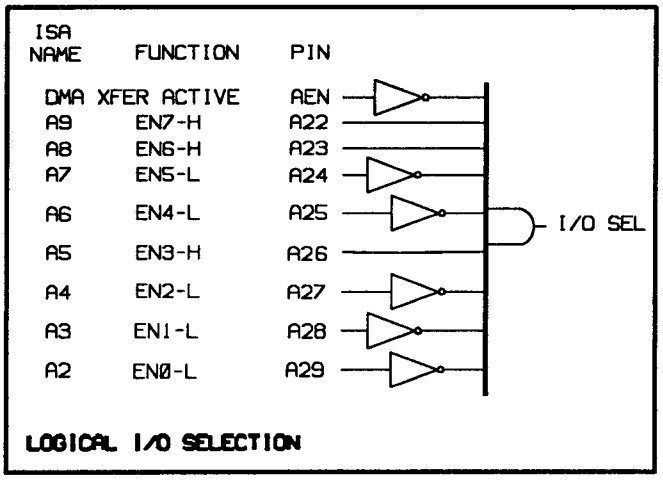


Figure.1 Logical I/O selection.

Looking at the I/O address space control lines made several things obvious. Addresses A0 and A1 are register selects. I guess-timated that somewhere on the board or in a chip the remaining eight lines enable the board for I/O. The likely decode is shown in the figure as the logical And of those eight lines, five are active low and three active high. We only need 6 of these lines to select an 8 bit address. By selecting from the available enables and forcing the unused two to zero or one as needed we can have the board appear at many different addresses. I chose to use the two active high and four active low to assign the board to C0h. The remaining active low line is grounded.

I used one of the active high lines to qualify IO operation selection off the 8085 IO/M\ line. The 8085 board does not fully decode IO/M\, RD\ and WR\ to distinct IORD\ or IOWR\. The same idea could be used in a Z80 system to decode the IORQ\ and RD\ or WR\. I decided to connect the HDC reset line to the 8251 DTR\ line as a convenient

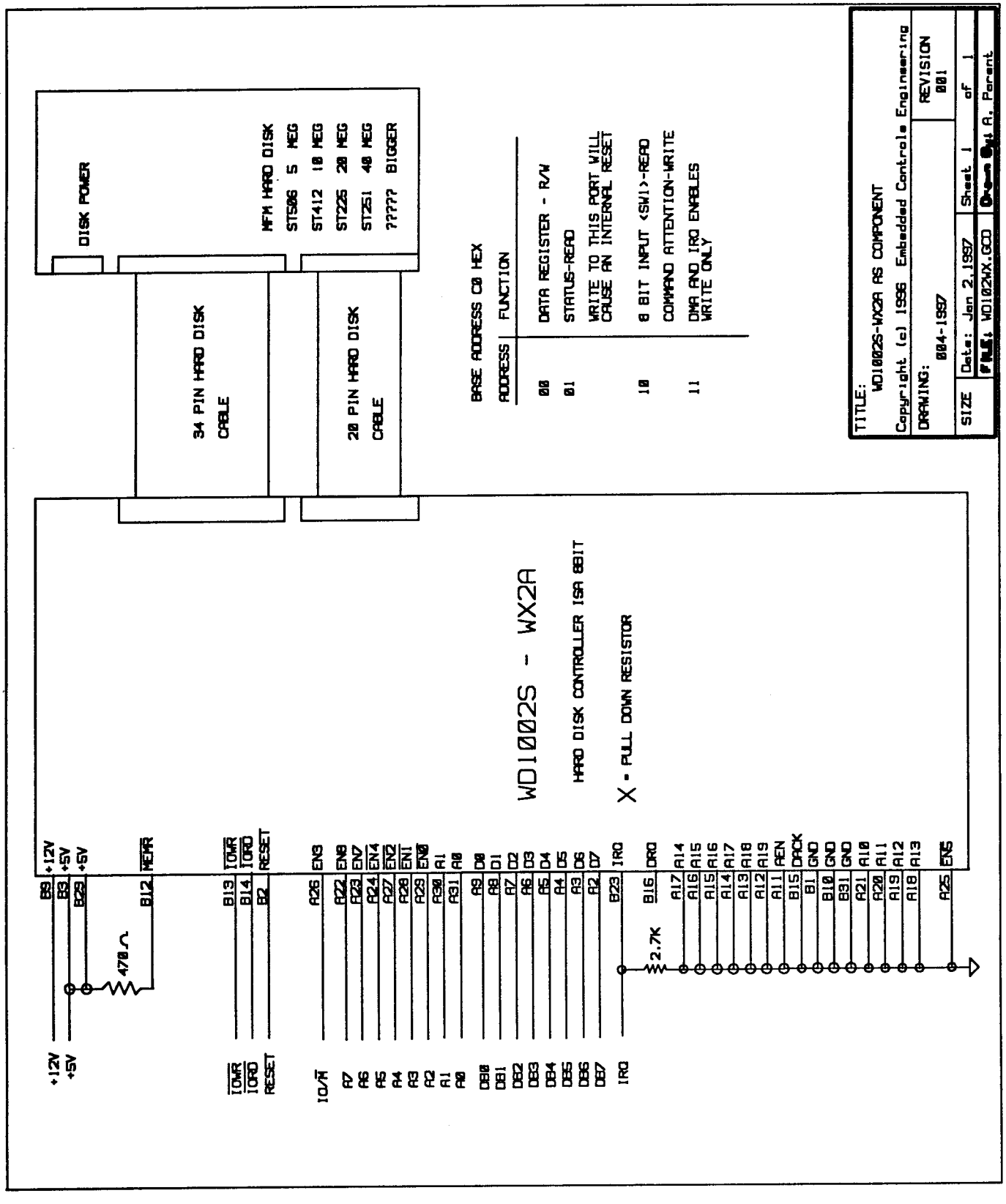


Figure.2 Pinout of the WD1002S-WX2 as component with interface names

```
; Header file for HDC
;
; Assemble using DRI ASM or equivelent.
; SIM and RIM will be inserted with DB.
;
SIM     EQU  30H    ; insert as DB SIM
RIM     EQU  20H    ; insert as DB RIM
;
; PROTO test bed card is Vt100 Printer Buffer Card
; These are slightly 8085A, with 4k EPROM, 2k RAM,
; 3 8251As, 8116 BRG and minimal glue logic.
; Mods applied are freeing up interrupt pins,
; Supplying BRG output to all 8251s and adding
; local reset generation
;
; Here we keep all the important addresses for
; code and whatever.
;
; Memory locations:
;
ROM     EQU  0000   ; ROM starts at 0000h
ROMLN    EQU     1000H    ; ROM is 4k part
RAM     EQU  2000H  ; Memory decode is 4k boundary
RAMLN   EQU  0800h  ; RAM is 2k part
;
; Magik memory addresses:
; Interrupts in hardware of 8085A.  The RIM and SIM
; instructions can monitor status of and mask interrupts.
;
INT55    EQU  02CH   ; Level trigger maskable priority 4
INT65   EQU  034H   ; Level trigger maskable priority 3
INT75   EQU  03CH   ; Edge  trigger maskable priority 2
TRAP    EQU  024H   ; non-maskable high priority 1
;
;IO decodes:
;
PORT51A EQU  000    ; data port 8251 device A
CSR51A  EQU  010H   ; CSR for 8251 device A
PORT51B EQU  020H   ; Data port 8251 device B
CSR51B  EQU  030H    ; CSR for 8251 device B
PORT51C EQU     040H    ; Data port 8251 device C
CSR51C  EQU  050h   ; CSR for 8251 device C
BAUDB   EQU  060H   ; base address for BRG
;
; The low 4 bits are baud rate.
; Baud rates below 1200 left off for brevity
; All ports use same BRG!
;
;Name        Address =        baud rate
;-------------------
B1200R  EQU  BAUDB+12   ; 16x 1200
B2400R  EQU  BAUDB+13   ; 16x 2400
B4800R  EQU  BAUDB+14   ; 16x 4800
B9600R  EQU  BAUDB+15   ; 16x 9600
B1920R   EQU  BAUDB+16        ; 16x 19200
;
; 8251A CSR Bits.
; Command bits
ENHUNT  EQU  10000000B ; Not used, sync mode
IREST   EQU  01000000B ; internal reset 1=reset
RTSF    EQU  00100000B ; 1 sets RTS pin to 0
ERES    EQU  00010000B ; Error Reset 1=reset
SBRK    EQU  00001000B ; a 1 Forces TX break
RXE     EQU  00000100B ; RX enable 1=enable
DTRF    EQU  00000010B ; 1 sets DTR pin to 0
TXEN    EQU  00000001B ; TX enable 1=enable
;
NOMCWD  EQU  RTSF+ERES+RXE+DTRF+TXEN ; typical CSR byte
;
; Note CSR51A DTR is used to control hardware reset of the HDC
; should it get lost and at power up.  CSR51A must have DTRF set
; to 1 to clear HDC reset.  This will be done in the general INIT
; code that sets up board level and branches to HDCinit.
;
; Status bits
;
DSRS    EQU  10000000B ; DSR status
SYNDET  EQU  01000000B ; SYNC det, sync mode
FERR    EQU  00100000B ; Framing error
ORERR   EQU  00010000B ; Overrun error
PERR    EQU  00001000B ; Parity error
TXEM    EQU  00000100B ; TX empty
RXRDY   EQU  00000010B ; RX reg has char if 1
TXRDY   EQU  00000001B ; TX reg can accept char if 1
;
```

and controllable single bit port. I may add that the DTR\ goes high at power on reset time so this give me a power on reset as well. This makes it possible to do a software reset of the HDC.

Further simplification was found by choosing to not implement DMA. This meant ignoring the DRQ line and forcing the DACK line to the false condition. Data is buffered on the controller card so DMA to keep up with the disk is not required. I did hook the IRQ line to the RST5.5 line on the 8085 for later interrupt use because it can also be polled using the RIM instruction. This means a polled transfer. Also lines like AEN had to be forced to false to prevent spurious deselects. AEN is used during DMA data transactions to disable the address selection logic as the DMA controller does this with the DACK signal. I might add that IORDY and IOCHK\ on the ISA bus were not used by the board. For initial testing, this is easier to debug.

See Figure 2 for the pinout of the WD1002S-WX2 as component with interface names

The target system is something I have a bunch of. They are printer buffer cards for vt100 terminals. Their basic design is 8085, 4k of eprom, 2k of ram, 3 8251 usarts with rs232 conversion and a single baud rate generator. This, with minor mods, is useful for control systems on a card that is 4.5 by 5 inches. I have a simple debug monitor with download to use on this board. It's simple and general enough in design to use for anything. This was a suitable platform to test the interface to the HDC. Other possibilities could have be 8748/9 or 8751 single component CPUs. The obvious use is on a CP/M system that needs a hard disk interface.

The interface at this point is all mechanical and interconnection to the 8085 board. The 8085 board does not have a external bus but it's easy enough to pick up the needed signals directly off the 8085 and 8251. Some would complain that I may have exceeded the fanout of the 8085 but the 3mhz part has good output drive and fairly relaxed timing. The board, drive, and controller used the same power supply since

it was big enough. Testing has progressed to the point where the interface responds and drive seeks were performed by poking data to the various ports. Programming is the next aspect of the project and is a later article.

The idea that a PC ISA bus card can be applied to systems other than PCs is not new. In most cases the goal was to create a complete bus so any card could be used. While that approach has value, it is also far more complex. As another possible application, I looked at a modem card and again saw a simplified interface that would lend itself to other applications as a black box component. My goal was to find a hard disk interface cheap. By examining a widely available PC hard disk controller and equating it to a large chip essentially made the board usable for non-PC applications. Also, there are many hard disk boards for ISA that appear the same or very similar. This would make replacements easier to come by.

Allison J Parent
<allisonp@world.std.com>

```
; HDC IO:
;
HDCB    EQU  0C0H   ; base address of controller
DATAR   EQU  HDCB+0 ; HDC data register bi directional
STATS   EQU  HDCB+1 ; HDC status read
        ; NOTE: A Write to STATS causes internal reset
EBIR    EQU  HDCB=2 ; Eight bit input, reads setup sw
        ; NOTE: A Write to EBIR causes a command attention
DMAIRQ  EQU  HDCB+3 ; DMA and IRQ enables write only
;
; DMAIRQ bits
;
; Even though they may not be used at the bus level they must be
; enabled for status IO via STATS read.
;
DMAEN   EQU  01B    ; Enable DRQ external request and status bit
IRQEN   EQU  10B    ; Enable IRQ external resquest and status bit
;
; commands are packet oriented.  The host will first write to
; EBIR to force the controller to command attention for a command
; write when REQ is asserted the host will then write a multibyte
; command block via DATAR. Monitoring the status of STATS register
; is required for protocol handshake.
;
; HDC port read bits for STATS
;
HDIRQ   EQU     00100000B       ; interrupt request
DMARQ   EQU  00010000B ; DMA request
BUSY    EQU  00001000B ; controller is executing a command
CDSTS   EQU  00000100B ; Command or data
        ; Indicates a data transfer <1> or status<0>
IOSTS   EQU  00000010B ; IO direction
        ; Indicates direction of transfer read if set, write if clear
REQST   EQU  00000001B ; Request
        ; Controller is ready for data transfer if set
;  Host buffer
;
        ORG  RAM    ;  at start of RAM
SECBUF  DS   512
;
;
; Command buffer
;
; Some commands allow 11 bits for cylinder address, the largest
; value allowed is 1024 (10 bits).  Number of heads are limited
; to 8.  Sectors per track is 17 invarient. A sector is 512 bytes.
; The largest drive is then 1024*8*17*512=71mb
;
HDCMD0  DB   00     ; Command code
HDCMD1  DB   00     ; bit 5 drive, bits 4-0 head number
HDCMD2  DB   00     ; bits 7-5 MSB of cylinder number
                    ; bits 4-0 sector number
HDCMD3   DB  00     ; LSB  low 8 bits of cylinder number
HDCMD4  DB   00     ; Sector count or interleave (command dependent)
HDCMD5  DB   00     ; bit 7 retry disable, general retry
                    ; bit 6 retry disable, ECC retrys
                    ; bits 2-0 step rate
                    ;  000=3ms, 100=200us, 101=70us
        ;
        END
```

Simplex III

Part 3

by Dave Brooks

Special Feature

Advanced

TTL Computer

Simplex-III Machine Instructions

This is the third in a series of articles decribing Simplex-III, a home-designed CPU. The machine was built in the late 1970's, using discrete TTL parts. Simplex-III was a big-endian machine, with the least-significant byte of a multi-byte object at the highest address. Data width was 8 bits, with a 64kB address space.

The overall architecture was described in the previous issue. The programmer-visible registers are shown in Table 1. These registers were implemented in a small bipolar RAM, the "SPAD".

Table 1

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

A typical instruction proceeds by reading the S register (in 2 bytes) from SPAD, incrementing it by 2, and posting the result back to SPAD and to the two bytes of the Store Address register. Two memory-read cycles follow, with the 16-bit address register auto-decrementing each time.

Since all instructions are 2 bytes long, this leaves the operation-code in the Instruction Register, and the address offset in the "R" register (a temporary register). The 8 instruction bits are broken (in most instructions) into 3 fields thus (recall that in big-endian notation, bit-0 is the most significant):

Operation	4 bits [0:3]
Operand register	2 bits [4:5]
Index register	2 bits [6:7]

The first operand is selected by the "operand register" bits, as A, X1, X2, or X3. The second operand is addressed in memory, by zero-padding the content of "R" (the address offset), and adding the content of the "index register" S, X1, X2, or X3. As before, this addresses the highest (least-significant) addressed byte of the operand. As the operation proceeds, both the memory and SPAD address registers auto-index.

Relative jumps are executed in the same way, by taking an address forward or back relative to S, and storing that address in S. The 8-bit displacement is always zero-filled to 16 bits: an instruction bit determines whether this value is to be added or subtracted from the S register.

For all registers except A, the data length is fixed at 2 bytes. A "length" register can be loaded, to define the length of A at anything from 1 to 8 bytes. Of course, this value simply defines the repeat count for operations involving A. The length value remains set until changed explicitly changed again.

A typical instruction is:

```
ADD    A,X1,DATA
```

where "A" is the operand, and "X1" the address base. "DATA" is the address of the second operand, as an offset from register X1. The effect of this instruction is to add A (at the currently set length) and DATA, storing the result in A. The condition bits will also be set.

Instruction-bytes take one of the 4 formats listed in Table 2. All instructions use Format 1, except for the SET and BC instructions.

Table 2: Instruction Formats

Format	– Function Byte — 0 1 2 3 4 5 6 7
1.	-Op-code- -Ro- -Rx-
2.	0 Length
3.	1 IT J K
4.	Condition Dirn.

Formats 2 and 3 are used by the SET instruction. When I[4]=0, the remaining bits set the effective accumulator length to 1..8 bytes, giving a working (integer) precision of 8..64 bits. When I[4]=1, I[5] terminates an interrupt, while I[6:7] are the J and K inputs to a flipflop. This flipflop drives a "signal" LED as a prompt to the user. SET instructions ignore the address/data byte. This instruction is not used often enough to justify the extra logic to count S by only one byte, and so save the space taken up by the address byte.

Format 4 is used by the Branch Conditional instruction. If I[7]=0, the address offset is added to S for a forward branch, while if I[7]=1, the offset is subtracted, for a backward branch. I[4:6] determine the branch condition, as shown in Table 3.

The machine instructions are listed in Table 4. The JI/JIL instruction has a side-effect, in that if Ro=X1, X2, or X3, S is stored in Ro (ie save subroutine link). If Ro=A, the link is discarded (ie the instruction is a simple jump). Then or otherwise, q => S (indirect jump). If this instruction is executed in interrupt level, both S registers are loaded: this intentional side-effect was used at boot-up to set up the interrupt vector (the interrupt-level S register) as explained below.

Subroutine return is typically done using the "MIR" (move & increment register) instruction. This takes a 2-byte register, adds a constant, and stores the result in the same or another register. A typical subroutine entry/exit sequence is listed in Table 5, where the "DP" (define pointer) operator is defined to point 1 byte below the target instruction.

Interrupts

Two copies of the SPAD (and condition-code) registers exist, for interrupt and base-level tasks. At reset, the interrupt set are selected. Typically, the initialisation code ends with a JI instruction, which leaves both register-sets' "S" registers pointing at the same instruction. A SET instruction then changes to base level code, which then over-writes the instruction at the JI target with a relative jump to the interrupt routine. This interrupt routine is coded as a loop, which finishes with a SET <base level>. The following instruction will jump back to the head of the interrupt routine.

Hence an interrupt switches register sets, and immediately jumps to the head of the interrupt routine. After interrupt processing is done, the SET <base level> is executed, leaving the interrupt level's S pointing at its successor instruction, ie the jump back to the head of the interrupt routine. Processing then resumes in the base-level registers.

Next issue

This article has described Simplex-III from a programmer's standpoint. The next article will cover the hardware debugging facilities.

Table 3 Branch-Condition Codes

I4	I5	I6	Condition
0	0	0	Not zero
0	0	1	Zero
0	1	0	Not negative
0	1	1	Negative
1	0	0	Not carry
1	0	1	Carry
1	1	0	Always
1	1	1	Sense switch ON (on front-panel)

Table 5 :Typical Subroutine Entry/Exit Sequence

```
SUBPTR  DP    SUBROUTINE     //Point below 1st byte of code
                                 //Calling routine
JIL     X2,X3,SUBPTR         //SUBPTR accessible from X3
                             //Link (X2) points to JIL instr.
                             //Caller continues

SUBROUTINE                   //Subroutine code: link in X2
                             //Save X2 if reqd.
        MIR   S,X2,2         //S = X2+2, ie return past JIL

        JIL   A,X3,SUBPTR    //As above, but link is discarded
                             // (no link is ever stored in A)
```

Machine Instructions

Definitions:

Ro = The operand register (A, X1, X2, X3)
Rx = The index register (S, X1, X2, X3)
I = Offset byte from instruction (zero extended to 16 bits)
Q = The effective memory operand address, defined as (rx + I)
q = Content of Q

Where the target of an operation is shown as "z, c" this means the C (condition) bits are affected, as well as loading a result into z.

The instruction assignments are as follows:

Code	Mnemonic	Function
0	ADD ro + q => ro, c	//Add memory to register
1	ST ro => q	//Sore register
2	LD q => ro, c	//LD sets conditions, unlike the 8086
3	XOR ro XOR q => ro, c	//Exclusive OR memory to register
4	AND ro AND q => ro, c	//Logical AND memory to register
5	LCP ro AND q => c	//AND, but don't store result
6	ADS ro + q => q, c	//Add register into store
7	SUB ro - q => ro, c	//Subtract memory from register
8	CMP ro - q => c	//Subtract, but don't store result
9	JI/JILq => s	//Jump indirect/link. See text for side effects
A	CTS q + 1 => q, c	//Count in store. Always 1-byte operand
B	SET	//Set values for operand length, interrupt terminate, etc.
C	RTL ro + ro => ro, c	//Add to itself, ie shift left
D	BC s +/- I => s	//Branch on conditions. Direction, and condition set by Ro, Ri bits
E	LDI I => ro, c	//Load immediate, 8-bit zero-extended
F	MIR ro + I => rx, c	//Move/increment register. Effectively moves Ro+I to Rx

Adding the 16550 UART to a 65xx system

by André Fachat

Special Feature

Interfacing

6502 + 16550A

How to add a 16550A UART to a 65xx system

I have stumbled across quite a few 6502 computers that use the 6551 ACIA for a serial interface. When I used this chip, I found it rather slow. Since I was doing a Multitasking operating system for the 6502, the interrupt latency became too large to reliably catch each and every character from the device even at 2400 Baud. This was even worse on the 1MHz my C64, into which I had built an ACIA interface in a socket under the SID 6581 (sound chip).

I had already heard of the 'FIFO-serial interface' for PCs, and even replaced some 8250's with 16550A's myself. So I decided to replace the ACIA in my C64 with the 16550A UART.

Bus differences

The 16550A UART chip was designed for a different bus than the 6502, so there are differences in how to handle the chip. When I had a first look at the interface, I really thought, 'why did they build the chip this way!'.

The Reset line is high active - which places the chip in "running mode" at power up, and then an explicit signal on the Reset line only resets the chip. The 65xx series use an active low -RESET line, such that even a simple RC-element with an additional Schmitt-Trigger suffices for a simple computer.

The Interrupt line is high active also. You can not wire-or it together with other devices (brain-damaged PC design!) as in 65xx systems for example. (That would even work if the passive state would be to leave the line, and not to pull it down to 0 Volts - but that's PC design...) And of course, there is no general clock line, but accesses take place during active RD/WR lines, as long as the chip is selected. The UART has two read (RD and -RD) and write (WR and -WR) lines, where each triggers a data transfer (i.e. not RD and -RD, but RD or -RD — which is IMHO quite peculiar, but I can think of uses for DMA, for example).

These differences in the bus system are, in my opinion, the reason that many people still use the ACIA 6551, and not the much more sophisticated UART 16550A. A more sophisticated chip involves more sophisticated software. In contrary to the simple handling of an ACIA, the UART is indeed more complicated to handle, which also seems a reason for its low usage. In fact there are many caveats in the UART design that make the thing terrible to program. See the Serial FAQ at ftp://ftp.phil.uni-sb.de/pub/people/chris/The_Serial_Port (*and the TCJ FTP site*) for more information. But then the UART has the already mentioned advantage of the FIFO, which makes it attractive.

Schematics

When I was rebuilding my ACIA interface to a UART interface, I had to reuse some sockets and chips, because I didn't really want to build everything from scratch (where I didn't have all the parts at home at that time anyway...) The old Interface card was a daugtherboard to be put between the socket for the C64 SID chip and the SID itself. An additional connector gave three signals: A9 to divide the SID address space in half (although the SID has only 28 registers, they are mirrored in a 1kByte block of I/O space in the C64), -E to enable the ACIA at all and -IRQ to signal the CPU. I soldered A9 and -IRQ to appropriate places on the C64 motherboard, while -E goes to a switch.

With the ACIA and the use of a dual 2-to-4 decoder 74LS139, I was able to do the whole thing. It even made the ACIA disappear from the memory map, when it was disabled. This didn't work with the UART, because I needed more circuitry from the decoder to manage the RD/WR handling. Well, one could surely think of something cleaner, but I wanted it quick...

Here's how it works in the C64

All pins from the SID are connected from the C64 SID socket to the SID socket on the daughterboard, except -CS (pin 8). This line is or'ed with the additional A9, to remove the SID from the upper half of its memory window, and then given to the SID. The first half of the '139 then gives the condition that Phi2 is high, -CS is active, and A9 is high. This output is then fed into the -E pin of the second decoder. The enable line (from the external enable switch) switches between the two used and the two not used outputs of the decoder. R/-W as the lower decoder address line then switches between the RD and the WR line. These outputs are or'ed with the inverted Phi2, and fed to the UART.

This arrangements has two purposes: The two decoder stages that have Phi2 in the first decoder give a delay to the beginning of the access. This is needed, as the C64 switches from video chip memory access to CPU memory access with the phases of Phi2. So the address lines need some time to

adjust. (The VIA 6522, for example, expects the address lines valid at the beginning of Phi2, and doesn't work with the C64 that way. It needs the starting transition of Phi2 to be delayed.) The ORing with the inverted Phi2 then stops the access by invalidating the RD/WR lines when Phi2 becomes inactive. This also is a reason for Phi2 being used as CS line.

Another example

I have also built a Dual UART card with two 16550A for my selfbuilt 6502 computer, and there I used a similar approach. I took a 74LS138, a 3-to-8 decoder. The select line for the I/O area goes to the decoder as enable line (-E1), as well as Phi2 (E3). -E2 is not used and set low. A0 is connected to R/-W of the system, and A1 is another address line. A2 is also set low. The outputs Q0-Q3 are then connected to the two chips.

-Q0 -> UART1 -WR
-Q1 -> UART1 -RD
-Q2 -> UART2 -WR
-Q3 -> UART2 -RD

The Software

Now that you have the UART in your computer, you have to have some software to use it. I have not yet rewritten the C64 OS to use the UART as serial interface (which I had done to use the ACIA before). Maybe I have this ready, when this issue of TCJ comes out, then you find it on my homepage http://www.tu-chemnitz.de/~fachat/. But I have already written a generic UART device driver for my selfbuilt 6502 Operating System, OS/A65.

The code shown here is part of that driver, together with a simple C64 binding. It actually follows the suggestions given in the "Serial FAQ". I have one problem, though. Because the interrupt generation is somehow buggy in the UART, the FAQ suggests to start the transmitter from outside the IRQ routine. Well, in my OS I don't have any device code outside the IRQ routine (that is called when data is being sent). But then, as the 6502 cannot directly decide where an IRQ came from, the interrupt drivers are (almost) all called when an IRQ occurs, with "higher priority" first. So the serial driver, being the one with the highest priority, is called very often, even if it is not the source of the IRQ itself. But that ensures that the IRQ routine is called and so I can check there, if I have to start transmission manually.

The listed program echos characters it receives from the serial line back to the serial line. It also takes characters typed on the C64 and sends them to the serial line. A part of the screen is used as character buffer, so that you can see something, when the program receives characters.

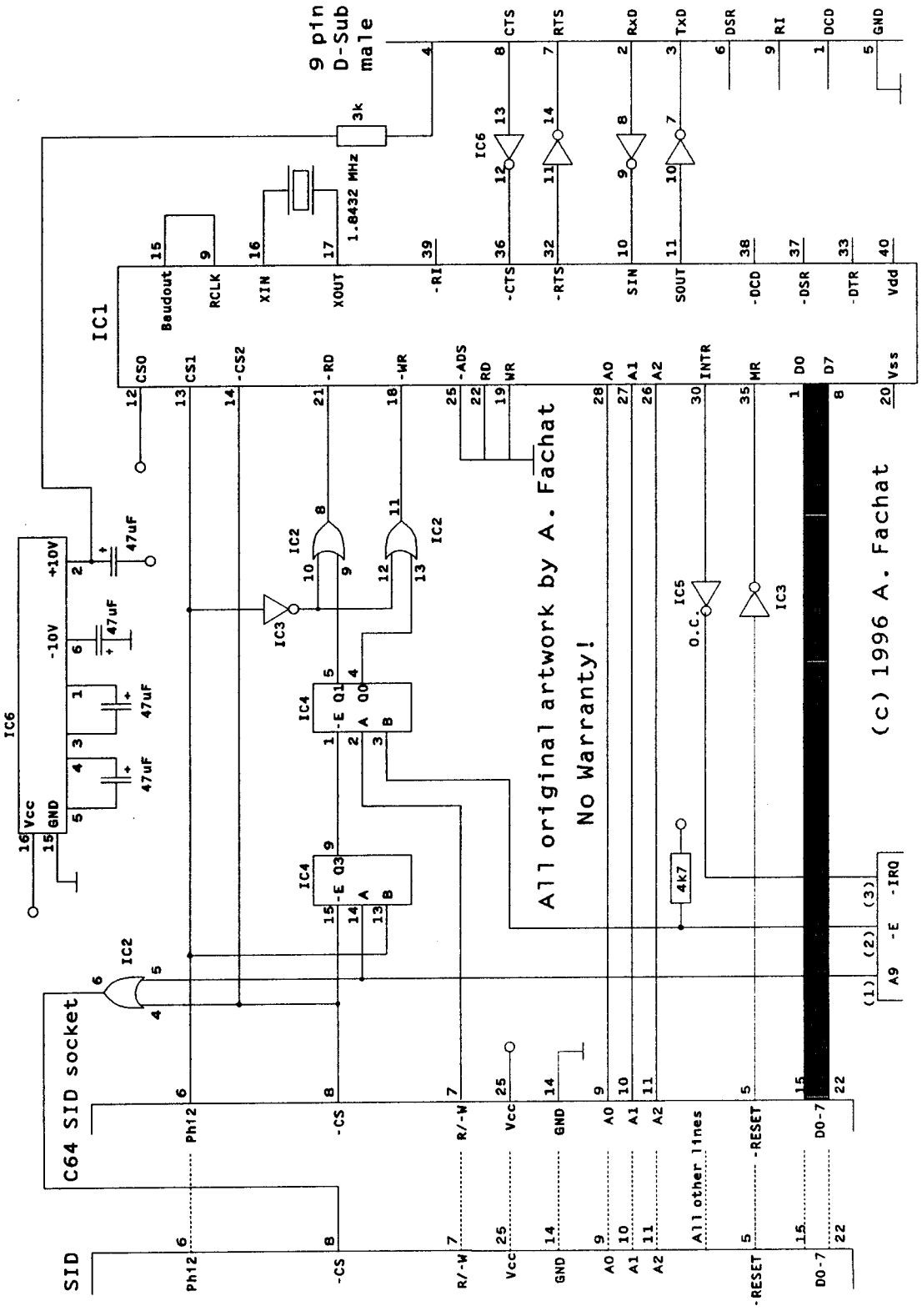
I have written a SLIP (Serial Line Internet Protocol) program for my selfwritten OS, that uses an UART. On my 1 MHz C64 with builtin UART it replies to Internet PING messages without packet loss at 9600 baud. Using the same program with an ACIA, even at 2400 baud, is completely useless due to lost characters.

Conclusion

The 16550A might not be the chip of choice for simple applications, where a high data rate is not necessary. But if you don't want to use the serial line as a terminal line only, but want to do some serious data transfer, better take an UART. Here you see a way to use the UART in 6502 based systems and how to program it.

André Fachat (a.fachat@physik.tu-chemnitz.de)

```
/*
 * (c) 1996-1997 Andre Fachat
 * fachat@physik.tu-chemnitz.de
 *
 * This is part of a UART 16550A serial line driver
 * for the OS/A65 operating system.
 *
 * The 16550 is not really an easy chip, but it has
 * 16 byte input and output FIFO buffers, which
 * allows  much higher interrupt latencies.
 *
 * routines defined here are:
 *
 * devini - initializes 16550A hardware
 * devirq - must be called in the interrupt
 *                routine
 * setspeed     - sets the speed of the transmission
 *                (yr = speed index in the divisor
 *                table)
 * txon/txoff   - switches transmitting on/off
 * rxon/rxoff   - switches receiving on/off
 *
 * routines needed are:
 *
 * GETC         - read byte from application in IRQ
 *                (via software FIFO)
 * PUTC         - write byte to application from IRQ
 *                (via software FIFO)
 * STRCMD - get status of software FIFO
 */

/* UART register definitions */

#define RXTX        0  /* DLAB=0 */
#define IER         1  /* DLAB=0 */
#define DLL         0  /* divisor low, DLAB=1 */
#define DLH         1  /* divisor high, DLAB=1 */
#define IIR         2  /* Irq ID Reg, read only */
#define FCR         2  /* FIFO Control, write only */
#define LCR         3  /* Line Ctrl Reg */
#define MCR         4  /* Modem Ctrl Reg */
#define LSR         5  /* Line Status Reg */
#define MSR         6  /* Modem Status Reg */
#define SCR         7  /* 'scratchpad', unused */

/* UART is the (memory mapped) address of the UART
 * The E_* codes are distinct error codes. E_OK is
 * 0, the others are not.
 * "status" is a local variable */

/* status: Bit  0 = 1= handshake enabled
 *              1 : 1= no ACIA found
 *              5 : 1= set RTS hi
 *              6 : 1= we are transmitting (DC_SW_TX)
 *              7 : 1= we are receiving (DC_SW_RX)
 */

#define   DC_SW_RX  %10000000
#define   DC_SW_TX  %01000000

/* 16550 divisor values for BAUD rates ?, 50, 75,
 * 110, 134.5, 150, 300, 600, 1200, 1800, 2400,
 * 3600, 4800, 7200, 9600, 19200
 */
divisor   .word  -1, 2304, 1536, 1047, 857, 768, 384,
          .word   192, 96, 64, 48, 32, 24, 16, 12, 6

/*************************************************
 * init UART
 *
 * This routine is according to the Serial FAQ by
 * Chris Blum, Release 18. The FAQ is posted
 * regularly to comp.sys.ibm.pc.hardware.comm and
 * comp.os.msdos.programmer and can be obtained by
 * ftp at ftp://ftp.phil.uni-sb.de/pub/people/chris/
```



All SID pins connected through from C64 to new SID socket, except pin 8, -CS
The usual Power Supply connections apply...

IC1	16550A
IC2	74LS32
IC3	74LS00 (or use IC5 with pullups)
IC4	74LS139
IC5	7416
IC6	MAX232

```
 * The_Serial_Port
 *
 * (These routines get called with the device number
 * in the x register. As this driver supports one
 * UART only, x is always 0 - that may change) */

devini  lda #0
        sta status

;check if there's something like an UART at all
         ldy UART+MCR
         lda #$10
         sta UART+MCR
         lda UART+MSR
         and #$f0
         bne nodev
         lda #$1f
         sta UART+MCR
         lda UART+MSR
         and #$f0
         cmp #$f0
         bne nodev
         sty UART+MCR
; check if it has a scratchpad register
; if not then it's plain 8250
         ldy UART+SCR
         lda #%10101010
         sta UART+SCR
         cmp UART+SCR
         bne dev8250
         lsr
         sta UART+SCR
         cmp UART+SCR
         bne dev8250
         sty UART+SCR
; now check the 16xxx versions
         lda #1
         sta UART+FCR
         lda UART+IIR
         ldy #0
         sty UART+FCR
         asl
         bcc dev16450
         asl
         bcc dev16550
; else dev16550A; currently only this one
; is supported

; ok, we detected a 16550A, i.e. a chip
; with working FIFO
         lda #%10000000
         sta UART+LCR
         ldx #14*2      ; 9600 BAUD
         lda divisor,x
         sta UART+DLL
         lda divisor+1,x
         sta UART+DLH
         lda #%00000011       ; 8N1
         sta UART+LCR

         lda #7         ; no FIFO enable and
                        ; clear FIFOs,
         sta UART+FCR   ; trigger at 1 byte
         lda #0
         sta UART+IER   ; polled mode (so far)
         sta UART+MCR   ; reset DTR, RTS

        clc
        rts

nodev           ; no UART at all
dev8250         ; no Scratchpad, no FIFO
dev16450        ; scratchpad, no FIFO
dev16550        ; FIFO bug
         lda status
         ora #2
         sta status
         lda #E_NOTIMP
         sec
         rts

/*************************************************
 * UART interrupt routine
 *
 * The interrupt routines are called one after each
 * other for every device in this OS.
 * If an interrupt source has been removed, then
 * E_OK is returned, E_NOIRQ otherwise. So this
 * routine even gets called, when lower level
 * interrupts occur. This is due to the single level
 * interrupt structure of the 6502 (not counting the
 * NMI, which is not really of use in this OS)
 *
 * The routine is nasty due to several reasons: The
 * UART doesn't periodically generate interrupts,
 * when the transmitter is empty, as the ACIA does.
 * So it is possible to be able to send (i.e. put
 * characters in software send FIFO) but the
 * interrupt driven driver doesn't know about it.
 * Therefore, the Serial FAQ suggests to start the
 * transmission manually when data is written to
 * the drivers internal buffers. In this OS the
 * buffers (software FIFOs) are totally independent
 * FIFOs (Streams) written to and read with PUTC
 * and GETC. To avoid lockups, I therefore check if
 * I can start sending every time the interrupt
 * routine is called, even if the interrupt source
 * is not the UART itself.
 *
 * The FAQ suggests checking if data is to be sent
 * after all UART interrupt sources have been
 * handled.  But "Start transmission by simply
 * calling the tx interrupt handler after you've
 * written to the tx fifo [software-FIFO] of the
 * program..." (These are things you can do with
 * DOS.... ts,ts,ts) doesn't work here. Instead we
 * check it in every interrupt call. */

devirq
        .(
         lda UART+IIR ; UART IIR
         lsr
         bcc intr     ; ok, found IRQ
irqend   lda #E_NOIRQ ; no irq source found
         jmp irqe     ; not this one

         ;----------------------
intr
         and #%00000011      ; interrupt mask - makes
         tay            ; four possible IRQs
         bne int_tx
         ;-------
                        ; modem status interrupt
         lda UART+MSR   ; de-locked by reading the MSR
         jmp checkint   ; do it, even if this IRQ is
                        ; not enabled...
int_tx   ;-------
         dey
         bne int_rx
tx_loop                 ; transmitter empty interrupt
         jsr tx2        ; write data bytes to UART
                        ; FIFO
         jmp checkint
int_rx   ;-------
         dey
         bne int_status
rx_loop                 ; receiver interrupt
         jsr rx2        ; get one char and save
         lda UART+LSR
         lsr
         bcs rx_loop    ; still data in UART FIFO?
         jmp checkint
int_status :-------
         lda UART+LSR   ; line status interrupt,
                        ; de-locked by reading MSR
         ;-------
checkint                ; still another IRQ pending?
         lda UART+IIR
         lsr
         bcc intr       ; irq still pending
         jsr nobyt      ; check if we are still
                        ; allowed to rx (i.e. stream
                        ; has not been closed from
                        ; reader)
         ;-------
irqok    lda #E_OK      ; irq source removed
irqe     pha            ; we get here no matter if
                        ; UART IRQ or not
         lda UART+LSR   ; check if we are allowed to
                        ; start sending manually
         and #$40       ; (THRE = LSR bit 6)
         beq nbyt       ; no then end
         jsr tx2        ; otherwise fill FIFO
         jsr nobyt      ; check if we are still
                        ; allowed to receive (this
                        ; ensures to check even if no
                        ; traffic)
nbyt
         pla
         rts
         :----------------------
         ; Receive a single byte
rx2      .(
         lda UART+RXTX; load data byte
```

```
          bit status    ; are we receiving?
          bpl rx2end    ; no, then end
          jsr PUTC      ; and save in software FIFO
          bcc rx2end    ; no error -> end
          cmp #E_SLWM   ; stream below low water mark?
          bne test      ; this happens most and is
                        ; caught by nobyt anyway
rx2end    rts

          ; check stream (software-FIFO) status
&nobyt
          bit status    ; are we receiving?
          bpl rx2end    ; no -> end
          jsr STRCMD    ; get state of stream
          bcc rx2end    ; ok -> end
test
          cmp #E_NUL    ; stream has been closed from
                        ; reader?
          bne tstwater
          jmp rxoff     ; yes, then shut receiver off
tstwater
          cmp #E_SEMPTY ; stream is empty?
          beq wl        ; ok, ensure that RTS is low
          tax
          lda status    ; we want RTS hi?
          and #1
          bne wh        ; yes, then high
          txa
          cmp #E_SFULL  ; stream full
          beq wh        ; then RTS high
          cmp #E_SHWM   ; stream above high water
                        ; mark?
          bne twl             ; no then branch
wh        ldx #0
          jmp rtsoff
twl       cmp #E_SLWM   ; stream below low water mark?
          bne rx2end    ; no then end
wl        ldx #0 ; otherwise RTS low
          jmp rtson
          .)
          ;---------------------
          ; Fill transmitter FIFO
tx2       .(
          bit status    ; are we transmitting?
          bvc txrt      ; reading IIR should clear
                        ; this line
          lda UART+MSR  ; are CTS and DSR ok?
          and #%00110000
          cmp #%00110000      ; cts or dsr inactive
          bne txrt      ; yes then end
          ldy #15             ; number of byte
txloop
          jsr GETC
          bcs test2
          sta UART+RXTX ; send new data byte
          dey
          bne txloop    ; fill up FIFO
          bcc txrt
test2     cmp #E_EOF    ; we got a stream End-Of-File?
          bne txrt
          jmp txoff     ; yes, then shut transmitter
                        ; off
txrt      rts
          .)
          .)
/*************************************************
 * support routines */

dtroff    lda UART+MCR
          and #%11111110
          sta UART+MCR
          lda #0
          sta UART+IER
          rts

dtron     lda UART+MCR
          ora #%00000001
          sta UART+MCR
          lda #3
          sta UART+IER
          rts

rtsoff    lda UART+MCR
          and #%11111101
          sta UART+MCR
          lda status,x
          ora #%00100000
          sta status,x
          rts

rtson     lda UART+MCR
          ora #%00000010
          sta UART+MCR
          lda status,x
          and #%11011111
          sta status,x
          rts

/*************************************************
 * control handling
 *
 * This is rather OS specific, so maybe it's not
 * necessary here. One thing to mention is, that
 * rxoff and txoff just reset their bit in the status
 * byte. rxoff then sets RTS high (rtsoff). After
 * that, both check if transmitter or receiver are on.
 * If both are shut off, then DTR is set high
 * (dtroff).
 *
 * If the transmitter or receiver are enabled, both
 * set DTR low (dtron). The receiver also sets CTS
 * low (ctslo).
 */

rxon      jsr rtson
          jsr dtron
          lda #DC_SW_RX
          bne o2a

txon      jsr dtron
          lda #DC_SW_TX
o2a       ora status
          sta status
          bne ok

rxoff     lda status
          and #DC_SW_RX
          beq devoff
          jsr rtsoff
; signal  eof to software FIFO here
          lda status
          and #255-DC_SW_RX
          sta status
          jmp checkdtr

txoff     lda status
          and #DC_SW_TX
          beq devoff
; signal  close to software FIFO here
          lda status
          and #255-DC_SW_TX
          sta status,x

checkdtr  and #DC_SW_TX+DC_SW_RX
          bne active
          jsr dtroff
active    jmp ok

setspeed  tya
          and #%00001111
          asl
          tax
          beq ok
          lda UART+LCR
          ora #$80
          sta UART+LCR
          lda divisor,x
          sta UART+DLL
          lda divisor+1,x
          sta UART+DLH
          lda UART+LCR
          and #$7f
          sta UART+LCR

ok        lda #E_OK
          .byt $2c
devon     lda #E_DON
          .byt $2c
devoff    lda #E_DOFF
          .byt $2c
onotimp   lda #E_NOTIMP
          cmp #1
          rts
```

Regular Feature

All Users

dBase II & TCAP

The European Beat

by Helmut Jungkunz

Z-BASE - A way to use ZCPR's TCAP for dBASE II

When CP/M is mentioned, one often hears or reads the term "standard program". The most popular so-called standard programs then were WordStar, Multiplan and dBASE. I always liked dBASE, for it offered even blunt idiots like me an easy interface to screen-oriented programming, even if in very rudimentary form.

What annoyed me, was the fact, that dBASE had to be installed for each and every terminal. When ZCPR came about, I first thought:

Wow - finally no more installations of terminals! Well, that goes for ZCPR - aware programs and tools only. When you use dBASE or any other non-ZCPR-program, you would still have to install the program first.

One of the nice things in UNIX and in ZCPR is the use of a so-called TCAP (Terminal CAPabilities) file, that, once loaded, will provide a given screen environment, usable by all applications for that system.

dBASE by itself does not offer any possibilities to directly make use of the ZCPR TCAP, but it has a powerful, built-in interface to PEEK and POKE memory addresses and thus to execute machine code!

The theory of all that may be taken from the program in BOX 1.

"CALL <memory variable>" will branch processing to the address specified in a SET CALL TO <address> command. The address must be decimal. When you reach the called address, the register pair H-L points to the first byte of the memory variable; this

```
BOX [ 1 ] "DBASPOKE.INF".
-------------------------

dBASE II 2.3B
Technical Support Note # 25
Copyright 1983, Ashton-Tate, All Rights Reserved
General Distribution

Assembly code interface information                    28 Mar 1983
for dBASE II 2.3B running on CP/M 2.2 .

The syntax for PEEK, POKE, and CALL is as follows:

PEEK( <address> )
POKE <address>, <byte list>
SET CALL TO <address>
CALL <memory variable>

       POKE  stores  a list of values into a specified location in the
computer's memory.   PEEK is a numeric function that returns the value
of a specified byte in the computer's memory.  Both PEEK and POKE use
decimal addresses and values.

Examples:

STORE PEEK(128) TO value
STORE 128 TO location
? PEEK(location), PEEK(location+1)

* Subroutine..: LEADZERO.LIB
* Description.: Replaces leading blanks with leading zeroes.
SET CALL TO 42000
POKE 42000,  34, 37,164, 70, 35,126,254, 32
POKE 42008, 194, 33,164, 54, 48,  5,194, 20
POKE 42016, 164, 42, 37,164,201
STORE "   123" TO number
CALL number
? number

; LEADZERO.ASM Replacing leading blanks with leading zeroes.
; Written by Luis A. Castro, Ashton-Tate, 1982.
;
ORG     42000   ;Load in dBASE free area.
SHLD    SAVEHL  ;Save HL registers.
        MOV     B,M     ;Store length of string to counter.
;
LOOP:   INX     H       ;Skip to next character
        MOV     A,M     ;Fetch character
        CPI     ' '     ;Is it a space?
        JNZ     DONE    ;Yes, done.
        MVI     M,'0'   ;Replace space with zero
        DCR     B       ;Decrement counter
        JNZ     LOOP    ;Repeat until done
DONE:   LHLD    SAVEHL  ;Restore HL registers.
        RET
;
SAVEHL  DS      2

* Program.: HEX-DEC.CMD
* Author..: Luis A. Castro.
* Date....: 9/14/82, 7/12/83.
* Notice..: Copyright 1982,  ASHTON-TATE.
* Notes...: To convert an assembled program's HEX file
*   into a dBASE POKE sequence.  This program works only
*   with HEX dumps that look like the following:
*
*       :10A410002225A446237EFE20C221A4363005C21484
*       :05A42000A42A25A4C9D7
*       :0000000000
```

```
*   The file this program generates is a .LIB file and can
*   be added to a dBASE command file using an editor
*   or word-processor.
*
*   You will need to create HEX-DEC.DBF with the structure:
*       HDUMMY1    C    001
*       HLENGTH    C    002
*       HADDRESS   C    004
*       HDUMMY2    C    002
*       H11        C    002
*       H12        C    002
*       H13        C    002
*       H14        C    002
*       H15        C    002
*       H16        C    002
*       H17        C    002
*       H18        C    002
*       H19        C    002
*       H20        C    002
*       H21        C    002
*       H22        C    002
*       H23        C    002
*       H24        C    002
*       H25        C    002
*       H26        C    002
*
SET TALK OFF
STORE "123456789ABCDEF" TO hexvalues
* Macros to convert hex values to decimal values.
STORE [@($(Hlength,1,1),hexvalues)*16 +;
@($(Hlength,2,1),hexvalues)] TO Mhexlen
STORE [@($(Haddress,1,1),hexvalues)*4096 +;
@($(Haddress,2,1),hexvalues)*256 + @($(Haddress,3,1),hexvalues)*16 +;
@($(Haddress,4,1),hexvalues)] TO Mhexaddr
*
ERASE
? "CONVERTING A HEX DUMP TO DECIMAL"
?
ACCEPT "Enter hex file....." TO filename
IF filename=" "
RETURN
ENDIF
IF @(".",filename) > 0
STORE $(filename,1,@(".",filename)-1) TO filename
ENDIF
STORE !(filename)+".HEX" TO infile
STORE !(filename)+".LIB" TO outfile
ACCEPT "Enter description.." TO notes
?
? "Copying structure to HEX-DEC...."
USE Hex-dec
COPY STRUCTURE TO Hex-dec.$$$
USE Hex-dec.$$$
? "Appending from &infile......"
APPEND FROM &infile SDF
GO TOP
*
* Generate the POKE sequence.
ERASE
SET ALTERNATE TO &outfile
SET ALTERNATE ON
? [* Subroutine..: ]+outfile
? [* Description.: ]+notes
STORE &Mhexaddr TO address
? [SET CALL TO ]+STR(address,5)
? [*           0   1   2   3   4   5   6   7   8   9  10]+;
[  11  12  13  14  15]
DO WHILE .NOT. EOF
STORE &Mhexlen TO linelen
STORE &Mhexaddr TO address
IF linelen <> 0
STORE " " TO decimals
STORE "11" TO item
DO WHILE VAL(item)-10 <= linelen
STORE decimals+","+STR(@($(H&item,1,1),hexvalues)*16;
+ @($(H&item,2,1),hexvalues),3) TO decimals
STORE STR(VAL(item)+1,2) TO item
ENDDO
? [POKE ]+STR(address,5)+[,]+$(decimals,3,LEN(decimals)-2)
ENDIF
SKIP
ENDDO while .not. eof
SET ALTERNATE OFF
CLEAR
DELETE FILE Hex-dec.$$$
QUIT
* EOF hex-dec.cmd
```

is the length byte for character strings. The <memory variable> must be a string and may not be lengthened while you are putting data into it. Execute a machine language return to get back to dBASE. dBASE will handle saving the registers. All machine memory from A400H up to the bottom of the CP/M BDOS is available, but will be overwritten when a SORT is done. It is recommended that a MOVCPM utility be used to create protected memory areas for assembly routines above CP/M.

To convert an assembled program's hex file into a dBASE POKE sequence you may want to run the dBASE program HEX-DEC.CMD provided for you at the end of this documentation.

This was released some time ago over the internet and tells the story. I now sat down and took a look at the ZCPR environment. Each TCAP sits in memory, once ZCPR has started. Considering the latest TCAP version, the address of the graphic characters used by the terminals (GOELD) is at a defined offset of 13 from the start of "TCAP". "TCAP" is found at the page just above the Z3 environment address ENV.

To find these addresses, issue from NZCOM or Z3PLUS the command ENV. This macro will peek the corresponding page of Z3ENV for your system. Take down the hexadecimal address at the left top! That is all you need (plus the tool ZDDTZ.COM or a Hex-Calculator). In my case, I have the address F580 for ENV. I start ZDDTZ and enter:

```
DDTZ v2.5 by C.B. Falconer.
CPU=Z80
-hF580   (the - is the prompt, the
         rest is entered from the
         keyboard)
F580 62848    (This is what we
        want - the decimal form of
        the address!)
^C
A:>
```

(done)

Now we take down 62848 on a piece of paper. We use ZDE to edit our dBASE "TCAP.CMD" and enter this decimal address as the value for ENV. Everything else is calculated automatically.

Now bring up dBASE and issue the command:

DO TCAP

The screen will be erased and after a while, GOELD will appear at the top of your Monitor, followed by a colon ":". This is called by a fake "Strike-any-key-when-ready" command —>

```
accept to dummy
```

You may leave this out, but for debugging purposes, I think it is a great help. Once you strike your "ANY" key (okay, bad joke), the graphical characters from your extended TCAP will be displayed.

If you have an idea, how the HEX-TO-DEC routine shown here can be used to immediately return the ENV address in decimal to dBASE, I would certainly appreciate this.

Just to be complete, I also include a dummy extended TCAP, that can be used on all non-graphical terminals to use standard characters like "+","-","!" and so on.

This is an example, of how a Televideo terminal with no graphical characters may be turned into a "Graphical Terminal". This is a legal hoax that works nicely to convince ZCPR, your terminal can do it. And as a matter of fact, it does.

Regards from Germany

Helmut Jungkunz
helmut.jungkunz@metronet.de

BOX [2] TCAP.CMD

```
*Routine TCAP.CMD by Helmut Jungkunz, ZNODE 51, Germany
erase
store 0 to count
*hex TO dec CONVERSION routine NEEDED
* Here I enter the address of my Z3ENV in decimal manually
* to get this address, issue the command ENV from NZCOM or Z3PLUS
store 62848 to ENV
store (ENV+128) to ENV
store (ENV+13) to GOELD
? "GOELD =",GOELD
*the nect line is a "wait for any key pressed ..."
accept to dummy
store GOELD+17 to count
store count+16+peek(count) to count
store chr(peek(count)) to UL
*UL is Upper Left of a box
? count,UL
store (count+1) to count
store chr(peek(count)) to UR
*UR is Upper Right of a box
? count,UR
store (count+1) to count
store chr(peek(count)) to LL
*LL is Lower Left of a box
? count,LL
store (count+1) to count
store chr(peek(count)) to LR
*LR is Lower Right of a box
? count,LR
store (count+1) to count
store chr(peek(count)) to HL
*HL is Horizontal Line of a box
? count,HL
store (count+1) to count
store chr(peek(count)) to VL
*VL is Vertical Line of a box
? count,VL
store (count+1) to count
store chr(peek(count)) to FB
*FB is the character Full Block
? count,FB
store (count+1) to count
store chr(peek(count)) to HB
*HB is the character Half Block
? count,HB
store (count+1) to count
store chr(peek(count)) to TI
*TI is the character Top Intersection
? count,TI
store (count+1) to count
store chr(peek(count)) to BI
*BI is the character Bottom Intersection
? count,BI
store (count+1) to count
store chr(peek(count)) to MI
*MI is the character Middle Intersection
? count,MI
store (count+1) to count
store chr(peek(count)) to RI
*RI is the character Right Intersection
? count,RI
store (count+1) to count
store chr(peek(count)) to LI
*LI is the character Left Intersection
? count,LI
store (count+1) to count
enddo
```

——— end of TCAP.CMD ———

BOX [3] DUMBTERM.Z80

```
; Z3TCAP file:  Z3TCAP.Z80
ESC     EQU  27         ; Escape character
;
; The first character in the terminal name must not be a space.  For
; Z3TCAP.TCP library purposes only, the name terminates with a space
; and must be unique in the first eight characters.
;
TNAME:  DB   'TVIDUMMY      '     ; Name of terminal (13 chars)
GOFF:   DB   GOELD-TNAME   ; Graphics offset from Z3TCAP start
;
; Terminal configuration bytes B14 and B15 are defined and bits
; assigned as follows.  The remaining bits are not currently assigned.
; Set these bits according to your terminal configuration.
;
```

```
;       B14 b7: Z3TCAP Type.... 0 = Standard TCAP  1 = Extended TCAP
;
;       bit: 76543210
B14:    DB   10000000B ; Configuration byte B14
;
;       B15 b0: Standout....... 0 = Half-Intensity 1 = Reverse Video
;       B15 b1: Power Up Delay. 0 = None           1 = Ten-second delay
;       B15 b2: No Auto Wrap... 0 = Auto Wrap      1 = No Auto Wrap
;       B15 b3: No Auto Scroll. 0 = Auto Scroll    1 = No Auto Scroll
;       B15 b4: ANSI.......... 0 = ASCII           1 = ANSI
;
;       bit: 76543210
B15:    DB   00000000B ; Configuration byte B15
;
; Single character arrow keys or WordStar diamond
DB      'K'-40H         ; Cursor up
DB      'V'-40H         ; Cursor down
DB      'L'-40H         ; Cursor right
DB      'H'-40H         ; Cursor left
;
; Delays (in ms) after sending terminal control
;strings
DB      0           ; CL delay
DB      0           ; CM delay
DB      0           ; CE delay
;
; Strings start here
CL:     DB    'Z'-40H,0         ; Home cursor and clear screen
CM:     DB    ESC,'=%+ %+ ',0   ; Cursor motion macro
CE:     DB    ESC,'T',0         ; Erase from cursor to end-of-line
SO:     DB    ESC,'G4',0        ; Start standout mode
SE:     DB    ESC,'G0',0        ; End standout mode
TI:     DB    ESC,'G0',0        ; Terminal initialization
TE:     DB    ESC,'G0',0        ; Terminal deinitialization
;
; Extensions to standard Z3TCAP
LD:     DB    ESC,'R',0         ; Delete line at cursor position
LI:     DB    ESC,'E',0         ; Insert line at cursor position
CD:     DB    ESC,'Y',0         ; Erase from cursor to end-of-screen
;
; The attribute string contains the four command characters to set
; the following four attributes for this terminal in the following
; order:     Normal, Blink, Reverse, Underscore
;
SA:     DB    0                 ; Set screen attributes macro
AT:     DB    0                 ; Attribute string
RC:     DB    ESC,'?',0         ; Read current cursor position
RL:     DB    0                 ; Read line until cursor
;
; Graphics TCAP area
;
GOELD:  DB    0           ; Graphics On/Off delay in ms
;
; Graphics strings
;
GO:     DB    ' ','H'-40H,0     ; Graphics mode On
GE:     DB    ' ','H'-40H,0     ; Graphics mode Off
CDO:    DB    ESC,'.0',0        ; Cursor Off
CDE:    DB    ESC,'.3',0        ; Cursor On
;
; Graphics characters
;
GULC:   DB    '+'         ; Upper left corner
GURC:   DB    '+'         ; Upper right corner
GLLC:   DB    '+'         ; Lower left corner
GLRC:   DB    '+'         ; Lower right corner
GHL:    DB    '-'         ; Horizontal line
GVL:    DB    '!'         ; Vertical line
GFB:    DB    '*'         ; Full block
GHB:    DB    '#'         ; Hashed block
GUI:    DB    '+'         ; Upper intersect
GLI:    DB    '+'         ; Lower intersect
GIS:    DB    '+'         ; Mid intersect
GRTI:   DB    '+'         ; Right intersect
GLTI:   DB    '+'         ; Left intersect
;
;  Fill remaining space with zeros
;
 REPT   128-($-TNAME)
DB      0
 ENDM

        END
;
; End of Z3TCAP
```

Regular Feature

68xx/68xxx Support

C & Assembly

Small System Support

by Ronald W. Anderson

C Class Notes 8, Address Program in C

This time let's digest a typical short program in C and see just how it works. The program reads a data file of names and addresses (and other information if desired) and prints each record that contains a string that matches the string given on the command line. It ignores case. If I give it the command:

```
address jones
```

It will print all the records containing "jones" or "Jones" or any combination of caps and lower case. It doesn't just print the line but the whole record. The data file is simply a text file that can be edited with any editor. Records are entered just as they will appear in the printout:

```
* John Jones 1234 5th st. Elko NV 78901
   Phone: (000) 111-2222 *
```

The program will list any record that has a match to the search string on the command line. For example it would match the above record to "elk" or "nv". Let's look at the program and then go over it line by line. This program has been left in the style of ANSI C with function prototypes and function arguments defined in this new style.

```
/* address program in C - uses data file of name
   and addresses with records separated by lines
   with asterisk in the first column.  Matches a
   search string and prints each record that
   contains the string. */
#include <stdio.h>
#define TRUE 1
#define FALSE 0
// function prototypes
   int read_record(void);
   int match(char *matchbuf);
   int ch;
   char recbuf[600];
   FILE *infile;
void main(int argc,char **argv) {
   if (argc ==1)
   {
    puts("address needs a string to match");
   }
   if ((infile = fopen("c:\\addr.dat","r"))== NULL)
   {
       puts("can't open address data file\n");
       exit(1);
   }
// the next line is the main loop of the program.
// It calls the two functions in turn, uses the
// results to produce an output and loops until
// the end of the data file
   while (read_record()) if (match(argv[1]))
   {
   puts(recbuf);
   }
   fclose(infile);
   exit(0);
}

// this function gets a record in recbuf and
// returns TRUE until end of file is detected at
// which point it returns FALSE.
int read_record(void) {
   int k=0;
   while ((ch =getc(infile)) != '*')
   {
      if (ch == EOF) return FALSE;
      recbuf[k++]=ch;
   }
   recbuf[k] =0;
   return TRUE;
}

// this function searches recbuf for a match to the
// search string and returns TRUE if it finds one,
// otherwise returns FALSE
int match(char *matchbuf) {
   int m=0,r=0;
   while (recbuf[r])
   {
     while (tolower(recbuf[r++]) ==
                        tolower(matchbuf[m++]))
     if (matchbuf[m] == 0x00) return TRUE;
     m =0;
   }
   return FALSE;
}
```

You might think this is a very short program for what it does. It makes rather full use of the shortcuts available in C. It combines asignment statements and function calls. It uses function call returned values in tests.

You might note that the data file path and name are hard coded into the program, so you must use the filename addr.dat and it must be in your c:\ directory (i.e. the root directory). You can hard code a different name or path if you wish.

Note that we put the main() function first. This is more in line with standard C programming practice. The gurus all say that main is really an outline of your program calling out the various functions in order, and if you are to do top down programming, you should start there. You can block out the called functions to be empty, i.e.:

```
void function() { }
```

Now you can write the functions one at a time and test them, perhaps with a lot of print statements in the function or in main() to test the operation of the function.

While all this sounds good on paper, speaking from expe-

rience, I can say that main() by the time I am done, doesn't resemble main() that I started with! As I look at each function I get ideas for improvements and main() changes accordingly. That doesn't, however, make the approach wrong or invalid. It is always a good idea to block out a program before working on the details. Advocates of top down programming would simply say that I don't spend enough time in the planning stage. I start coding too soon. I would counter that I could spend a very long time planning and my code still would require debug.

I sometimes violate this rule if I know that one function in the program is going to be particularly difficult. I might try to do that one first, doing only a dummy main program that calls it and allows testing of it. Later when the hardest part has been debugged, I'll go back and finish main and then fill in the other, hopefully easier functions.
Well, this has not been a complete item by item discussion of C, but over the last year and a quarter, we have covered the main features of the C programming language. Hopefully at least, this has been done in a way that will allow you to continue with a good textbook on C to learn more.

Whether you have a very small C, one of the dialects of the original Small C, or something more complete like Borland Turbo C or Microsoft Quick C, you will find the differences small. The Small C versions don't have float or double data types, but you can do all of the things we've done in this series. That is, you can manipulate files and do integer arithmetic quite nicely. If you are running a 6809 with FLEX you might be able to find a copy of Windrush C by James McCosh. It has float and double data types and it is not super complicated or large out of line with the resources of a 6809. If you find a later version it will include my scientific function package that I wrote for it and gave to James. If you get an earlier version and want the "scipack" package, let me know and I'll send it to you. I have new and improved sci function algorithms done since then.

If you are an old time reader of '68' Micro Journal you might remember a product called "Duggers Growing C". I was asked to review it way back when, and I found bugs by the dozen. I had access to the source code (in 6809 assembler) of the runtime library and was able to fix a number of bugs, but I couldn't review the thing and give it a "thumbs up", nor could I fix bugs in the compiler proper, since I didn't have source code. I am not talking about bugs in that the compiler didn't meet the then standard (actually one didn't exist then), but just plain operational errors, particularly in the math package.

At the time I offered indirectly to fix it for Duggers. I talked to them and they said "Well, we wrote, compiled and ran a few programs and they ran fine". I guess if that was the extent of their testing, I don't wonder why it was buggy. They did some dumb things, for example testing a 16 bit integer for zero by summing the two 8 bit parts of it. Any value whose upper and lower bytes were 2's complements, like for example, 00000001 11111111 (byte values decimal 1 and -1 in signed arithmetic) when added would produce 8 zeros plus a carry and the carry was ignored so this number and a lot of others tested true for zero! Of course the bytes should have been ORed so that a 1 in any position would show up in the result and test non zero. I spent a month of evenings fixing numerous such errors and sent Duggers my result. I received a phone call thanking me for my efforts on the runtime package. Though I offered to work on the compiler proper for a copy of the source code, my offer was turned down and the ad was dropped from '68'.

If you have questions, feel free to write. I am still by no means an expert in C, but I can find answers to most questions and would be happy to do so. In the past few weeks I've helped three other programmers with answers to questions about C. I think I am past the beginner stage at last.

Assembler

Let's look this time at some of the more interesting instructions in the 6809 set. For starters how about the MUL instruction. It multiplies the value in the A accumulator by the value in the B accumulator yielding the product in the D accumulator.

```
LDA #5
LDB #6
MUL
```

The D accumulator now contains: 0000000000011110
This is of course the value 30. If you know that the result is not going to be larger than 8 bits you can consider the answer to be in the B accumulator, i.e. just the 00011110 part of the above. The problem with a multiply is that the result can be so large as to require twice as many bits to represent it.

The beauty of this instruction is that it takes 12 clock cycles and is much faster than the add and shift method that would have to be used otherwise to multiply two 8 bit integers. It is left to the user / programmer how to interpret the result. Assuming unsigned bytes works best or at least is easiest to keep track of. You can assume integer multiply in which case the binary point is at the right end of the result, or you can assume fractional multiply in which case the result must be shifted left one place and the binary point is at the left end of the result. You can handle signed arithmetic by negating negative numbers and keeping track of the sign, re-negating the result if the sign is calculated to be negative. It is a pain for all but the simplest uses, but it can be useful. For example a compiler writer might use it to calculate the offset to the location of a byte or integer in a two dimensional array as the starting location plus the product of the two array subscripts (times 2 for an array of 16 bit integers). It is my belief that some of the less common instructions in the 6809 set were included for writers of compilers to use.

I might mention here that the 68000 instruction set includes both an integer and a fractional multiply called IMUL and FMUL to handle some of the above details for you. It also includes a DIV instruction which the 6809 does not.

I thought it strange when I first saw the 6809 instruction set that it didn't have more instructions that would work on the D accumulator. It has complement and shift instructions

for A and B but not for D. For your thought I offer some subroutines to do some of these instructions.

```
COMD    COMA
        COMB
        RTS
NEGD    COMB
        COMA
        ADDD #1
        RTS
ASLD EQU *   ;This is a way of giving a subroutine
             ; two names

LSLD    ASLB
        ROLA
        RTS
ASRD    ASRA
        RORB
        RTS
LSRD    LSRA
        RORB
        RTS
```

We probably haven't discussed the difference between Arithmetic shifts and Logical shifts. There is no difference when the shift is to the left. A Zero is shifted into the low order bit:

```
LDA #1
ASLA
 or LSLA
result: 00000010
```

The 1 has shifted one place to the left.

Shifting to the right is different, however. The reason is signed arithmetic. When a register contains a signed value, the leftmost bit is the sign bit. When a negative number is divided by 2 the sign bit must be preserved, thus:

```
1111110  (-2)
shifted right one place (ASR)
11111111 (-1)
```

The value has correctly been divided by 2. If we didn't preserve the sign bit as in the LSR instruction, the shifted value would be:

```
01111111 or 127
```

If we are dealing with signed numbers we need to use the ASR instruction but if we are dealing with something like a bit mask we probably want to use the LSR. For example, here is a code fragment to find the highest bit that is a 1 in a byte value:

```
FIND    CLRA LOOP  CMPB #0
        BEQ DONE2
        LSRB
        INCA
        BRA LOOP
DONE2   DECA
        RTS
```

A call to FIND would be:

```
LDB VALUE
BSR FIND
```

On return, A contains the number of the highest order bit that was a 1. Bits are counted right to left, the lowest order bit being bit 0 and the highest being bit 7. The algorithm is to shift the value until the remaining value is zero, when all the bits have been shifted away, counting the shifts. If the VALUE happened to be 0 we get -1 back indicating an error. If the value is 1 A is incremented to 1 for the single required shift but the DECA instruction makes it 0 again which is correct because the low order bit (bit 0) was the only one on. Though this might look like a contrived example, I assure you I have done things like this in the past.

Saving and Restoring Registers

There is a trap you can get into very easily using assembler. Suppose you are going along in some code and have values in both accumulators but you want to go to a subroutine and do something else. If the subroutine needs to use the accumulators to do its thing, the values in A and B that you had in your main program will be lost.

Most of us write subroutines so they save the values from whichever registers they are going to use, and restore these values on exit. The easiest way to do this is to use the system stack and the PSHS and PULS instructions. Subroutine:

```
NAME    PSHS A,B
        ...
        instructions using A and B
        ...
        PULS A,B
        RTS
```

Of course if you do this you can't return values in A or B (or D). You must return results in a variable somewhere or perhaps in the X or Y registers.

The use of PSHS and PULS instructions requires some great care. Remember that the return address for the subroutine is on the stack. If you Push more bytes than you pull or vice versa, the return address will be corrupted and the program will go off into the wild blue and lock up or start executing garbage!

These are some of the hardest bugs to find and some of the easiest to introduce in a program. Not saving and restoring the contents of a register that needs to be saved is also a hard bug to find in an assembler program.

Had you not guessed, PSHS and PULS instructions are followed by a register list. You can save one register or all of them. The order of the list is unimportant. The processor handles the order so they don't get scrambled by the process.

Of course with multiple instructions you can reverse the order and effectively swap the contents of two registers:

```
PSHS A
PSHS B
.....
PULS A
PULS B
```

This will SWAP the contents of A and B. You must PULL the last thing you PUSHed first. If you DON'T want to swap you must do it this way:

```
PSHS A
PSHS B
.....
PULS B
PULS A
```

If you just want to swap two registers the EXG instruction can do that:

```
EXG A,B
EXG X,Y
```

If you want to copy the contents of A to B or X to Y you use the TFR instruction:

```
TFR X,Y
TFR A,B
```

This leaves the contents of the first (source) register intact and copies it into the second (destination) register. Note that for either of these instructions the registers to be exchanged or those involved in the TFR must be the same size. You cant EXG A,X. The assembler will flag this error for you.

We ought to discuss the logical AND and OR instructions. First let me say that the operations such as ADD, SUB, AND, OR, EOR etc. all leave the results in the register that is being used.

```
LDA #5
ANDA #$FE
five = 00000101
$FE  = 11111110
RESULT  = 00000100
```

The AND result is left in A after the execution. The effect of an AND is to turn individual bits OFF. Since our "MASK" had the low order bit off, the result would be to make the contents of A an EVEN number because we have turned the 1's bit off.

```
LDA #15
ORA #$10
15 = 00001111
$10 = 00010000
RESULT= 00011111
```

The effect of the "MASK" in the case of the OR is to turn individual bits ON.

The AND instruction can be used to see if the contents of A or B have a particular bit ON:

```
MASK EQU $04  00000100
LDA VALUE ANDA MASK BNE ...
```

Whatever VALUE happens to be, the branch will be taken if bit 2 is a 1 and will not be taken if bit 2 is a 0.

There is a test that doesn't destroy the contents of the accumulator. It is the BIT instruction

Suppose in the above example VALUE = 7 After the AND instruction A would contain 4 because we have ANDed away the other bits.

```
LDA VALUE BITA MASK BNE ...
```

The result is the same as in the previous case where we used AND, EXCEPT that A now still contains the original value of 7.

The effect of the BIT insruction is to AND the MASK or whatever with the contents of A but NOT to write the result into A. We could do multiple tests in a sequence without having to reload A with VALUE again:

```
LDA VALUE BITA #$80 BNE SEVEN BITA #$40 BNE SIX ...
```

We could test each bit with a different value to find the highest order bit that is on, as in one of the examples above. This obviously is the hard way to test all 8 bits of a byte value, but if we only wanted to test two or three, it might be fast and short.

There is a similarity in the situation of the SUB and The CMP instructions. We could test for A containing the value 17 a couple of different ways

```
LDA VALUE  (assume 17)
SUBA #17
BEQ ...
```

We would branch if the value is 17 because the subtract would make the contents of A zero. The condition code register zero bit would be set. The BEQ instruction looks for the zero bit to decide whether the EQUAL condition is true.

```
LDA VALUE
CMPA #17
BEQ ...
```

This does exactly the same thing. The CMP instruction subtracts the immediate value and the BEQ test checks the zero bit of the CCR, which is set as the result of the CMP. The difference is that the value in A remains 17. That is, the result of the subtraction is not written back into A.

General

I've recently been given an almost antique 286/12 computer to play with. A friend of mine wanted to upgrade his computer that was considered quite a machine in 1988 when he bought it. It has a 360K 5.25 inch floppy drive, a 1.44 Mbyte 3.5", 640K of memory, a 145 watt power supply, a 40 Mbyte hard drive (IDE), and a reasonably nice AT style keyboard. It also had an EGA display board.

We decided after a good look that about the only parts of any value for an upgrade would be the keyboard and the 1.44 Mbyte floppy drive. He decided he might as well buy a new one and told me to keep it with an eye toward giving it to someone or some organization who might be able to use it. Yes, things surely do get obsolete quickly. The sad part is that the computer runs fine. With the addition of an old VGA board, and by plugging my monitor and mouse onto it, I checked it all out and loaded some software on it. It works flawlessly except that it is about 15 times slower than my present 486 system. If I needed only a fairly simple word processor system, I could have one very inexpensively with a computer like this as a start. Actually it would work admirably for what I am doing right now, using it as a

simple word processor to write this column. If I wanted more storage, I would probably only add a larger hard drive and some more memory. The motherboard has four simm sockets so it looks as though I could add up to 4 Mbytes, though there is no documentation and it might be a problem to get it all running again.

My present system is most likely 2 to 3 times slower than the new pentium systems hitting the market currently. I've always been happy with a computer until I've used a faster one for a short while. The old 6809 system used to be great fun to use. Now when I compile a program and have to wait five minutes, I writhe in the pain of boredom. I can compile the equivalent program on my latest computer in a few seconds.

When I started using 68000 based systems with real MFM hard drives, I was in heaven. Those 5 or 10 minute compiles could be done in 2 or 3 minutes. Hard disk data transfer, while vastly faster than with floppy drives, was still a bit slow.

The 286 based system I bought (12 MHz) seemed to be a whiz until I got my hands on a 386-40. Then came the 486 system. Each time, as I said, a few days use of the newer and faster system made the old one seem like it had been filled with molasses! Actually, my 486SLC50/2 is built on a motherboard that will accept and run a 66/2 as well. The clock frequency is changed by moving one jumper. Just for fun I decided to try running the processor (clearly marked as a 50 MHz one) at 66. It worked fine. Someone at work reminded me that the 66 version of this board has a heat sink on the processor chip. I switched back to 50 temporarily and went and bought a heat sink with fan for $10 at the local computer store. I put it on top of the processor with some heat sink compound between, and anchored it in place at the corners with a little hot glue dribbled from corners to circuit board. I discovered that the hot glue didn't bond very well to the board and later added some clear silicone to hold the assembly together. I might explain that the heat sink was intended for a 486DX and thus didn't just clip in place. I noted and appreciated the speed increase of 4/3.

The best data transfer rate I saw on the 68000 system was about 90 Kbytes per second from its MFM hard drive. My present system hits 1,250,000 bytes per second. Loading an editor or a compiler is VERY fast. Now the computer does something and waits for its next instruction from me rather than the other way around.

Though the new computers are great and I wouldn't trade mine for something slower when I use it for program development or word processing using Windows TruType fonts and printing in graphics mode, there are numerous reasons why I would rather play with the old 6809 or 68000 system when I don't need the speed. The biggest reason is the simplicity of the old computers (and the software that runs on them). I have found that I can run versions of that old software on the new machine and it runs like lightning. For example my screen editor that I wrote years ago for the 6809 system is now running in a new and improved version (translated from PL/9 to C) with more features on the PC. WordStar for Windows takes a full minute to fire up. It must for some reason completely obscure to me, have to read 100 disk files before it can run. My old PAT editor loads as fast as I can type the command and get my finger on and off of the ENTER key. The .exe file is right around 50K of code. I suspect with Windows and WordStar running, I am using most of the 4 megs of RAM I have on the PC system.

You've seen over the past several issues how programming in Assembler is done. It looks (and is) rather easy for a small program, say a utility program. When programs get really large, one appreciates a faster computer. Our company has been using the 6809 for about 12 or 13 years now. We never found it inadequate to perform its intended purpose UNTIL our programs grew to the point that we were crowding the memory limit. Then it began to be a headache to try to squeeze code into the 48K or so that we could use for program code (We need some stack space and the I/O uses a bit of the memory address range).

Our reason for moving on was NOT the lack of capability of the small processor, but the lack of availability of sufficient memory for some of our larger applications. We began using multiple computers in some of our products, etc. The use of one of the new computers has given us a huge memory space, and a vast improvement in performance which we didn't know we needed until we had it. By that I mean that all the extra computing power has brought to mind things that we can do in our computer that we wouldn't have thought of earlier.

I guess the bottom line is pretty much the philosophy of The Computer Journal. If you need to do something complex use the latest 80X86 based 100 MHz computer. If you want to learn something about how computers work, go get an old microcomputer and dig in!

The TCJ Store

Regular Items

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

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

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

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

SOLD OUT

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

Special Items

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

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

CP/M Kaypro Catalog

TCJ has taken over Chuck Stafford's Kaypro business. Here's our current catalog.

Upgrades
- Advent TurboRom
  - K4-83 ..... $35.00
  - K10-83 ..... $35.00
  - Kx-84 ..... $35.00
- MicroCornucopia Roms
  - Pro 8 ..... $35.00
  - 884 Max $35.00
  - 884 Max (Lo) $35.00
  - Character ROM $35.00

Add-ons
- HandyMan $75.00

Disk Drives
- Dual Density TEAC FD-55BV $15.00
- Quad Density TEAC FD-55FR $15.00
- Pair $25.00
- ST-225 20 MByte MFM HD ..... ??

Disk Controllers
- WD-1002-05 HDO $75.00

Tech Data
- Kaypro Technical Manual $25.00
- *Microcornucopia* Schematics with Theory of Operation
  - K-II/4 83 ..... $15.00
  - K-10/83 ..... $15.00
  - All-84 ..... $15.00
  - Any two $25.00
  - All three $30.00

Software
- Advent Harddisk Formatter $25.00
- TurboRom Applications Patches $10.00
- TurboRom Developers Diskette $10.00
- Kaypro 10/83 Tinker Kit $10.00
- Kaypro 2,4/84 Tinker Kit $10.00
- Kaypro CP/M 2.2H Autoload set 8 diskettes for K-10/84 ..... $40.00

Other Stuff
- Keyboards $30.00
- Video - CRT and board $40.00
- Kaypro Carrying bags $75.00

Kaypro machines
- K-II, K-2, K-4, K-10 available in various condition.

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

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

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

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

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

Regular Feature
Contact Listing

SUPPORT GROUPS FOR THE CLASSICS

TCJ STAFF CONTACTS

TCJ Editor: Dave Baldwin, (916)722-4970, FAX (916)722-7480 or TCJ BBS (916) 722-5799 (use "computer", "journal", pswd "subscriber" as log on), Email: tcj@psyber.com,.

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

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

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

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

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

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

REGULAR CONTRIBUTORS

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

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

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

Helmut Jungkunz, Munich, Germany, ZNODE #51, 8N1, 300-14.4, +49.89.961 45 75. Email: helmut.jungkunz@metronet.de.

USER GROUPS

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

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

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

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

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

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

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

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

Adam International Media, Adam's House, 1829-1 County Road 130, Pearland, TX 77581-5040. Contact Terry Fowler for information. Web: "HTTP:// WWW.Flash.Net/~coleco"

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

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

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

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

Vancouver Island Senior ADAMphiles, ADVISA newsletter by David Cobley, 17-885 Berwick Rd. Qualicum Beach, B.C., Canada V9K 1N7, (604)752-1984. dcobley@qb.island.net

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

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

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

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

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

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

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

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

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

OTHER PUBLICATIONS

The Z-Letter - has ceased publication.

The Analytical Engine, by the Computer History Association of California, 4159-C El Camino Way, Palo Alto, CA 94306-4010. Home page: http://www.chac.org/chac/ E-mail: engine@chac.org

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

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

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

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

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

Amstrad PCW SIG, newsletter by Al Warsh, 6889 Crest Avenue, Riverside, CA 92503-1162. $9 for 6 bi-monthly newsletters on Amstrad CP/M machines.

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

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

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

Update Magazine, PO Box 1095, Peru, IN 46970, Subs $18 per year, supports Sinclair, Timex, and Cambridge computers. Emil: fdavis@holli.com.

SUPPORT BUSINESSES

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

Sydex, PO Box 5700, Eugene OR 97405, (541)683-6033. Sells several CP/M programs for use with PC Clones ('22Disk' format/copies CP/M disks using PC files system).

Elliam Associates, PO Box 2664, Atascadero CA 93423, (805)466-8440. Sells CP/M user group disks and Amstrad PCW products. Email:??

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

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

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

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

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

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

MicroSolutions Computer Products, 132 W. Lincoln Hwy, DeKalb, IL 60115, (815)756-3411. UNIFORM Format-translation, CompatiCard and UniDos products have been discontinued. Web page: http://www.micro-solutions.com.

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

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

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

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

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

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

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

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

The Computer Journal Back Issues

Sales limited to supplies in stock.

Volume Number 1: Issues 1 to 9

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

Volume Number 2: Issues 10 to 19

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

Volume Number 3: Issues 20 to 25

- Designing an 8035 SBC
- Using Apple Graphics from CP/M
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller
- Extending Turbo Pascal: series
- Analog Data Acquisition & Control
- Programming the 8035 SBC
- Variability in the BDS C Standard Library
- The SCSI Interface: series
- Using Turbo Pascal ISAM Files
- The Ampro Little Board Column: series
- C Column: series
- The Z Column: series
- The SCSI Interface: Introduction to SCSI
- Editing the CP/M Operating System
- INDEXER: Turbo Pascal Program
- Introduction to Assembly Code for CP/M
- Ampro 186 Column
- ZTime-1: A Real Time Clock for the Ampro Z-80 Little Board

Volume Number 4: Issues 26 to 31

- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: SCSI Adapter Software
- Inside Ampro Computers
- NEW-DOS: The CCP Commands (continued)
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: DoubleDOS
- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation
- Feedback Control System Analysis
- The C Column: Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program in Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying the CP/M Disk Parameter Block for Foreign Disk Formats
- Build an A/D Converter for the Ampro Little Board
- HD64180: Setting the Wait States & RAM Refresh using PRT & DMA
- Using SCSI for Real Time Control
- Open Letter to STD Bus Manufacturers
- Patching Turbo Pascal
- Choosing a Language for Machine Control
- Better Software Fitter Design
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 1
- Using the Hitachi hd64180
- 68000: Why use a new OS and the 68000?
- Detecting the 8087 Math Chip
- Floppy Disk Track Structure
- Double Density Floppy Controller
- ZCPR3 IOP for the Ampro Little Board
- 32000 Hackers' Language
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 2
- Non-Preemptive Multitasking
- Software Timers for the 68000
- Lilliput Z-Node
- Using SCSI for Generalized I/O
- Communicating with Floppy Disks: Disk Parameters & their variations
- XBIOS: A Replacement BIOS for the SB180
- K-OS ONE and the SAGE
- Remote: Designing a Remote System Program
- The ZCPR3 Corner: ARUNZ Documentation

Issue Number 32:

- copies still available -

Issue Number 33:

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

Issue Number 34:

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

Issue Number 35:

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

Issue Number 36:

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

Issue Number 37:

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

Issue Number 38:

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

Issue Number 39:

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

Issue Number 40:

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

Issue Number 41:

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

Issue Number 42:

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

Issue Number 43:

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

Issue Number 44:

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

Issue Number 45:

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

Issue Number 46:

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

Issue Number 47:

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

Issue Number 48:

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

Issue Number 49:

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

Issue Number 50:

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

Issue Number 51:

- Introducing the YASBEC
- Floppy Disk Alignment w/RTXEB, Pt 3
- High Speed Modems on Eight Bit Systems
- A Z8 Talker and Host
- Local Area Networks—Ethernet
- UNIX Connectivity on the Cheap
- PC Hard Disk Partition Table
- A Short Introduction to Forth
- Stepped Inference in Embedded Control
- Real Computing, the 32CG160, Swordfish
- PMATE/ZMATE Macros
- Z-System Corner, The Trenton Festival
- Z-Best Software, the Z3HELP System

Issue Number 52:

- YASBEC, The Hardware
- An Arbitrary Waveform Generator, Pt. 1
- B.Y.O. Assembler...in Forth
- Getting Started in Assembly Language, Pt. 3

- The NZCOM IOP
- Servos and the F68HC11
- Z-System Corner, Programming for Compatibility
- Z-Best Software
- Real Computing, X10 Revisited
- PMATE/ZMATE Macros
- Home Heating & Lighting, Pt. 3
- The CPU280, A High Performance SBC

Issue Number 53:

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

Issue Number 54:

- B.Y.O. Assembler
- Local Area Networks
- Advanced CP/M
- ZCPR on a 16-Bit Intel Platform
- Real Computing
- Interrupts and the Z80
- 8 MHZ on a Ampro
- Hardware Heavenn
- What Zilog never told you about the Super8
- An Arbitary Waveform Generator
- The Development of TDOS

Issue Number 55:

- Fuzzilogy 101
- The Cyclic Redundancy Check in Forth
- The Internetwork Protocol (IP)
- Hardware Heaven
- Real Computing
- Remapping Disk Drives through Virtual BIOS
- The Bumbling Mathmatician
- YASMEM

Issue Number 56:

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

Issue Number 57:

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

Issue Number 58:

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

Issue Number 59:

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

Issue Number 60:

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

Issue Number 61:

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

Issue Number 62:

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

Issue Number 63:

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

Issue Number 64:

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

Issue Number 65:

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

Issue Number 66:

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

Issue Number 67:

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

Issue Number 68:

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

Issue Number 69:

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

Issue Number 70:

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

Issue Number 71:

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

Issue Number 72:

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

Issue Number 73:

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

Issue Number 74:

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

Issue Number 75:

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

Issue Number 76:

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

Issue Number 77:

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

Issue Number 78:

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

Issue Number 79:

- PC serial port in Forth
- Program This! - AT Modem Commands
- Center Fold - P112 Z182 board
- Simplex III - Homebuilt TTL CPU
- Real Computing - Real-time Control
- Small System Support - C and Assembler
- Embedded Development Choices

Regular Feature

Editorial Comment

Going Hi-Tech

The Computer Corner

By Bill Kibler

Welcome back to TCJ! I say that only because time is moving on and each issue is now taking some time to happen. Dave would love to move things faster than he is, but the rate of money coming into TCJ is rather slow. When I was editor, I would just dig deep into my pocket and find an extra thousand or so and keep on moving. Of course I never did get that money back and had to drop out as editor because of it. Dave however has no pockets to pull from and thus must print when he finally has the money.

I would love to help out and have a few times, but my own expenses have grown and I must make up for three years of losses. That of course explains why I am working on UNIX systems and doing C. I have almost recovered from the loss of sleep, free time, and money, only to look back and wish I could have made it work. Our time as the magazine supporting collectors is still around the corner, but systems like NT have people considering the old alternatives.

WINNT

Needing to pay for food and such has caused me to do regular programming on UNIX platforms. We use Win95 and WinNT platforms to run our tools on. These tools enable us to talk to the HP-Unix platforms and do our programming tasks. It gives me a chance to compare and understand better the advantages of simple and complex systems.

To show how much I have been converted to Unix, my fingers now work very automatically to move the cursor around in VI, the Unix editor. Vi is a little simple but very straight forward editor. On almost all platforms every Unix programmer or administrator must know how to use it. Some even love it, I am not there yet. It is however some of these little Unix tools that can give you a good feeling about using that platform. The tools usually work (hear that M/S), a manual is online, and when you usually have a problem, someone has already created a tools for you. It also explains why so many Unix tools are ending up on PC's.

Now I have moved my own box at work from NT3.51 to NT4.0. I liked 3.51 better, mostly for it's regular windows setup (like 3.1). The new win95 desktop I find a drag, especially all the layers one must go through to change something. That of course depends on if you are allowed to change them at all. They have changed many tool names, removed some, replaced others. I have tried to add a parallel port hard drive to the box several times now, all without success. It says it is there, works, is running, but I am unable to get a drive letter.

Now under DOS I use the fdisk program to find out if I had a drive that was not setup correctly and thus was the reason it wasn't being seen. Fdisk is no where to be found in NT, nor can I find a way to test for an unformatted hard drive. It may be there, they just don't want you to find it without having gone to one of their special thousand dollar schools.

At this point I say give me Linux or plain old DOS, even CP/M would be better than this nonsense. But I have one item to top even this mess.

MMX

Well Intel has their new CPU (the MMX or Pentium Pro, which is more Media Hype than horsepower upgrade) out and got one in the wings as well. I was reading an article in EETimes that caught my eye about the new CPU's. The units have power management built in, not to mention millions of transistors. This means they can idle back and save power when possible. These units also are run on anywhere from 1.8 to 3.5 volts depending on CPU speed, current needs, and tested performance at a given voltage. To do that a 4 or 5 bit signal is supplied to tell the power supply what voltage to select.

Two things came to mind, one the added complexity and a problem talked about in the article. I felt the complex-

Figure 1: 5 Bit voltage Code

Voltage	D4	D3	D2	D1	D0
3.50	1	0	0	0	0
3.40	1	0	0	0	1
3.30	1	0	0	1	0
3.20	1	0	0	1	1
3.10	1	0	1	0	0
3.00	1	0	1	0	1
2.90	1	0	1	1	0
2.80	1	0	1	1	1
2.70	1	1	0	0	0
2.60	1	1	0	0	1
2.50	1	1	0	1	0
2.40	1	1	0	1	1
2.30	1	1	1	0	0
2.20	1	1	1	0	1
2.10	1	1	1	1	0
2.05	0	0	0	0	0
2.00	0	0	0	0	1
1.95	0	0	0	1	0
1.90	0	0	0	1	1
1.85	0	0	1	0	0
1.80	0	0	1	1	0

*note D4 all ones for backward compatiblity with older cpu types.

ity is partly to enable a better yield on the parts, as some chips were being chucked that fell outside some of the design parameters. However if the voltage is kept a bit low or hi these once rejected chips will now work just fine, or at least they think so. I am not sure it's worth the extra hassles and possible component costs and failures.

Those cost and problems were of course the main concern. You see the part can drop current usage to around 300 mills. But hit a key and zap your up to 12 Amps, yup 12 AMPS! That is a big change for any power supply, not to mention a whole lot of amps for a single CPU chip. I am not sure what gets me more, the switching from nothing to full power, or the fact that full power is 12 amps. You need some pretty big traces to handle those amps, not to mention the BIG spike such a current change will cause elsewhere in the system. Talk about a few design nightmares.

The Real World

Reading about all these changes makes me feel good about some of my other projects. I am still doing some 68HC11 work on the side. It is mostly C and moving along rather smoothly. I got a good laugh out of an article in a C support magazine. The writer was talking about doing embedded C++ and how a reduced set was needed. The funny part was how he started the article. He mentioned how most embedded products are too limited for C and especially C++.

This of course didn't stop him from continuing on talking about using C++ for embedded work. It is true a few organizations are using C and C++ for embedded systems, it is just those embedded products are very expensive and usually have DOD stamped all over them. I still have problems with people who use the same term for million dollar projects and those in which the whole product will cost less then ten dollars. We need a new list of terms so people can understand there is a big difference between these two extremes of the concept.

To me embedded systems generally have a cost of well under $100 and are not a major design concern or cost. Industrial real time control systems on the other hand are typically a major part of the cost. Maybe another idea is one that Dave keeps using, it's embedded if one person can do the design work, while it is an industrial controller if it takes a full staff to design and support the product. What terms might you think are best to separate these two very different fields of work. Send me an e-mail and I will list the ideas next time.

Product Failure

For those who wonder what has happened with our own CDROM project, I can say it is moving along slowly. I now have up and running a CCS and a GodBout S-100 system with 8 inch disks. Between the two I hope to start going back through all my old disks and some recently acquired disks looking for items of interests. My main interest is source to BIOS and utilities. So far I have a good collection, but few have been moved to my hard drive for cataloging.

Over the years I have heard many stories about users turning on their old systems only to get nothing. Well it finally happened to me the other day. I hadn't used several units in about three or four months and moved them into position to help with the disk cataloging. I tried three of the Z100's that had worked earlier and one was still connected to the 8 inch drives I was also going to use for copying. Well you can probably guess that not a one of the units would boot up. A couple had video problems, one just wouldn't boot no matter what. I finally gave up for now and dug out an older terminal and used that to test the S-100 systems with.

My idea had been to use a Z-100 for a terminal with the occasional copy when one of the S-100 systems wouldn't read the 8" disk. The Z-100 also can copy from CP/M disk to MSDOS disk using one of their own utilities. Later checking showed the problem to be water damage that eaten away some of the traces and messed up one of the sockets. Where the water came from, I have no idea, but what a mess. I now have a better understanding of the frustration so many of you have told me about. However don't think using PC's will keep it from happening.

I got a panic call from an old friend who has some knowledge of computers. His major skill set is two-way radio which he did till he retired recently. He has played with computers some, but does not yet have the full grasp of what can go wrong. I have tried to help him resolve problems with windows which only frustrated him more as we often hit one wall or another.

This problem had to do with adding a second printer, in this case one of the newer Epson. When attempting to send data to it, he got back a paper out message, no matter what. My friend had gotten a second parallel port card for his PC. When trying that card, nothing work, and thus a phone call to me. I re-checked all he did and drug the printer home where it tested just fine.

I went back the next day and tried a few more things. First I checked the new parallel card and decided all the jumpers needed to be opposite from what the tech at the store told him. I used MSD and had a different, or OK status report on the two printer ports. Now that we knew the PC tech was wrong and one problems was conflicting port addresses, we tried the printers again.

After a few minor tests, all started to work properly. The problem it seems starts with his old parallel port which gives a paper out error on the Epson no matter what, a mis-configured second board, and not enough experience to know that most computer store techs have trouble using the correct end of a screwdriver. I guess at this point I need to do my normal complaint and say how bad it is and all, but I did that far too many times already, so I will just let it pass.

Till Next Time

Speaking of passing, I just ran out of space, so hope your trials and tribulations are minor ones, and keep hacking. Bill.

TCJ CLASSIFIED ADS

CLASSIFIED RATES!
$5.00 PER LISTING!

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

Commercial Advertising Rates:

Size	Once	4+
Full	$150	$90
1/2 Page	$80	$60
1/3 Page	$60	$45
1/4 Page	$50	$40
Market Place	$30	$120/yr

The Computer Journal
P.O. Box 3900
Citrus Heights, CA 95611-3900

MAGAZINES AND BOOKS

Historically Brewed. The magazine of the Historical Computer Society. Read about the people and machines which changed our world. Buy, sell and trade "antique" computers. Subscriptions $14, or try an issue for $3. HCS, 3649 Herschel St., Jacksonville, FL 32205.

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

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

Remember, TCJ has all Back Issues available. See the Back Issues page for a list of the articles in each issue.

FOR SALE

TRS-80 - MODELS I, 2, III, IV, 12, 16, POCKET COMPUTER, AND COCO: Software, hardware, internal and external disk drives (360K/720K), hard drive's (both complete and bubles), replacement motherboards, floppy drive controllers, video boards, RS-232 boards, keyboards, and more. Send 4x10 SASE for list.
Pete Bumgardner, Rt. 4 Box 36-H, Fort Payne, AL 35967-9408, (205)845-6581.

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

SUPPORT / INFO WANTED

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.