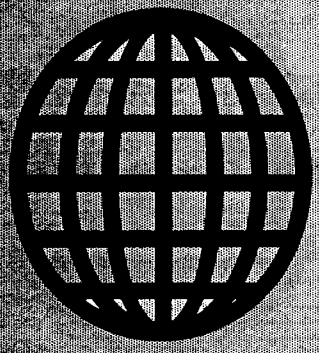
Providing Support Around The World



The Computer Journal

Issue Number 66 March/April 1994 US$4.00

Small System Support

Z-System Corner

DR S-100

Real Computing

Support Groups

PC/XT Corner

Little Circuits

Connecting IDE Drives

Multiprocessing Part III

Centerfold - Mr. Kaypro

The Computer Corner

ISSN # 0748-9331

TCJ - For Having Fun With Any Computer!

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Bill D. Kibler

Technical Consultant
Chris McEwen

Contributing Editors
Herb Johnson
Charles Stafford
Brad Rodriguez
Ronald W. Anderson
Helmut Jungkunz
Dave Baldwin
Frank Sergeant
JW Weaver
Richard Rodman
Jay Sage
Tilmann Reh

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

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

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used trademarks are acknowledged, and we apologize for any we have overlooked.

Apple II, II+, IIc, IIe, Lisa, Macintosh, ProDos; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. DateStamper, BackGrounder ii, Dos Disk; Plu*Perfect Systems. Clipper, Nantucket; Nantucket, Inc. dBase, dBASE II, dBASE III, dBASE III Plus, dBASE IV; Ashton-Tate, Inc. MBASIC, MS-DOS, Windows, Word; MicroSoft. WordStar; MicroPro International. IBM-PC, XT, and AT, PC-DOS; IBM Corporation. Z80, Z280; Zilog Corporation. Turbo Pascal, Turbo C, Paradox; Borland International. HD64180; Hitachi America, Ltd. SB180; Micromint, Inc.

Where these and other terms are used in The Computer Journal, they are acknowledged to be the property of the respective companies even if not specifically acknowledged in each occurrence.

TCJ The Computer Journal

Issue Number 66 March/April 1994

EDITOR'S COMMENTS

Welcome to issue 66, our eleventh year. This issue has regulars, specials, and hot items as usual. All this to support the exploding (well growing at least) field of collectible computers.

We start this issue with a bag full of reader to reader comments. We get several reports on choosing a language, some languages that people have had experience with, a few comments on the ZX81 article, and a request for help. This group of letters went over on pages, but it will be nothing like next issue, where I will attempt to catch up on some very interesting letters that have not been printed due simply to lack of space. Several of *TCJ*'s writers are taking a break, and with that I am going to take the chance to fill the void with our readers pearly words of wisdom (and a few complaints).

Next on the list of must reads is Frank Sergeant and his PC/XT corner. Frank is still gearing up for answering your letters about the old machines, and thus takes this chance to talk about an old project of his (a cheap PLD programmer) by explaining what Programmable Logic Devices are.

Jay Sage concludes his Failsafe Scripts topic next. Jay's work has doubled, not unlike many of us, and he has decided to cut his writing schedule down from a regular series to hitting special topics as time permits. Remember that Jay still can be reached if needed and if you send him letters, we will be printing those and his responses in the reader to reader section.

Next on deck is Rick Rodman and his status report on TCP/IP for small systems. Rick says it should be working by the time you get this issue! We added a little explanation to help fill in those gaps of understanding what OSI and all those layers amount to (a non-editable cake if you ask me...).

Our next bite of information is the conclusion from Tilmann Reh of his IDE interface article series. Now there is still some comments to be made later about actual BIOS implementations and such, but for now you have enough information to start trying it yourself. As you will see others have some hot IDE information that will help fill in the gaps that might still exist.

This issues centerfold is two fold (pun is..well..), we get some schematics dear to many Kaypro users, the Advent adapter board, and Chuck Stafford, Mr. Kaypro, gets center billing. Chuck starts a multi part article on building your own Advent decoder board. This is a good first time construction project with many options to be explored and skills to be developed.

Speaking of building, Brad Rodriguez gives us his updated 6809 multiprocessor II board. This improved version corrects a few minor problems, and...Well, Brad explains all the why and wherefores in his article.

Dave Baldwin is becoming a regular with his tips and circuits. This time he explains battery backups and other powerful topics (now how is that for a charged up pun...).

Ron Anderson takes a few stabs at C, and then answers some questions about 6809 software. He concludes with a few remarks about old systems and some explanations that may have been overlooked by beginners.

Herb Johnson answers the SPRING mail bag, but don't bounce too high or you'll miss the comments about a single chip IDE S-100 project. I have seen the early draft of Claude Palm's article about his work, and it all should be in the next issue (including schematic). Imagine, 240 megs of hard drive on a single S-100 card!

Our Support group listing has a few new entries as JW takes time to repair his broken Kaypro. He promises to be back next time and fill us in on what happen and how he resurrected the beast.

For all those who said what is a PLC (and there were many, very very many at that!) I present a quick explanation of PLC concepts with promises of more. Actually, I will be explaining all to the local Forth group, and hopefully have some Forth code next month.

Life at *TCJ* has been hectic the last few months. The holidays turned out to be a time hog and not a time off. Thus I must apologize to any would be writers and some readers as well for falling behind on the paper work end of things.

Since I attempt to get *TCJ* out on time each issue, it takes precedents over all else. That means phone calls, letters, and E-Mail get set aside often. I have some options to be explored down the line, but alas nothing will happen to improve things for the next few issues.

So if you have sent in orders for back issues or know someone waiting for a trial issue, please explain and understand that you will get your request answered, it just might take awhile.

If your wondering what happened to slow things down around here, how about this, three birthdays, wedding anniversary, valentines day, all in the first three weeks of February. Now if that's not enough how about a five year old son, still not convinced, try ten Llamas on ten acres. Let's go big and change jobs too. One last topper, *TCJ* was only three days late to the printer (who by the way moved as well)!

So despite all the odds, here is number 66 for your enjoyment. Happy Hobby Hacking! Bill Kibler.

READER to READER

Letters to the Editor

All Readers

MINI Articles

Bill...

It seems I am destined to embarrass myself publicly! Just two weeks before *TCJ* #65 arrived, I received a brand new catalog from Jameco Electronics. Surprise! They've added _more_ components -- over 50 pages worth -- and now have _no_minimum_order_! It's the "February-April 1994" catalog...which may hint at catalogs more than once a year. I've always had good dealings with Jameco; only their shrinking inventory and rising minimum order put me off. Happy days are here again! Jameco Electronics, 1355 Shoreway Road, Belmont, CA, 94002-4100, phone (415) 592-8097.

I also followed up some ads in *Nuts & Volts*, and received a flyer from Debco Electronics, 4025 Edwards Road, Cincinnati, OH, 45209, phone (513) 531-4499, orders (800) 423-4499. They've got a good selection of logic, linear, and microprocessor ICs, plus various other parts, at competitive prices. No minimum order! I bought some TTL ICs and delivery was prompt.

I'm glad to hear (from *TCJ* #65) that some distributors are taking small orders. I'd heard before that Dallas Semiconductor -- smart people! -- were taking orders from individuals. Now that Maxim is doing likewise, and in view of Dave Baldwin's excellent article on reset circuits, let me highly recommend the MAX690 reset circuit. I believe it's comparable to the TL7705, and probably more easily obtained. (We fixed a _lot_ of problems with MAX690's.)

Regards,
Brad Rodriguez

Thanks Brad. Got a few others who commented on how "hobby" computing is coming back in style and with it the need for selling again to the little people like us. Thanks. Bill.

Hello, and thanks for the comp. copy of *TCJ*.

I'm impressed with your editorial quality. I noticed that you would be issuing a special on the ZX-81. As a former member of a timex/sinclair user group, I thought you might be interested in the following info:

One support group still functioning in support of the ZX-81, along with the 1000, 2068, and QL, is L.I.S.T. (Long Island Sinclair Timex, which also includes NYTSE, a group which formerly met in New York City. The group has public-domain cassette tapes for the various machines, and has assembled two volumes of technical data, program listings, etc., for the ZX-81. The group may be contacted c/o Mr. Harvey Rait; 5 Peri Lane; Valley Stream, NY 11581. The group meets monthly on Long Island and publishes a newsletter.

Has *TCJ* considered regular support for the Timex and Sinclair machines? There are still quite a few of them out here (this message comes via a QL.)

Another message on another topic to follow.

Thanks Stoney for that note. Yes I heard about LIST a few days after sending 65 to the printer. I heard about QL's and always wanted to get one, but never saw where or how to do it. How did you get yours and how well does it work? Since TCJ supports all old machines, I guess you could say we support it, but were really trying to teach people how to do their own support. Thanks for writing. Bill Kibler.

Dear Mr. Kibler,

Thank you for printing my letter in Issue 64. After reading the other letters about Small-C I decided I had a few more things to say.

The choice of a 'standard' language for *TCJ* to use is a difficult decision. Not because there are so many good choices, but because there aren't any good choices.

There are only a few languages that could be considered for a magazine devoted to small systems (assuming nobody wants to create yet another language); assembler, Basic, C, Compiled Basic, Forth, Macro Assembler, Micro C, Pascal, Small-C, Structured Basic. I'll discuss each of them in turn. I'll also discuss compilers, interpreters, and translators. In my previous letter I discussed outputting P-Codes instead of assembly so that a single compiler could be used on all machines. I won't discuss that further here.

Assembly isn't really much of an option. There are simply too many different CPU's. If there were just two or three different CPU's then maybe. However, *TCJ* supports ALL old computers, to some degree or other.

Basic. Yeah, right. And I've got some ocean front property in Kansas I'll sell you. Cheap. I suppose it could be done,

but it wouldn't be pleasant. There are many variations of Basic, but most of them are similar and have similar flaws. That's to be expected since one company wrote most of them; Microsoft. Basic is slow, it.... Well, the old saying 'Basic and Cobol cause brain damage' comes to mind. It is however, the most widely available language on the old computers.

C. ANSI C and C++ are out of the question. They are too large. You'd have to stick with K&R. Most computers have C compilers or at least cross compilers. A full K&R C compiler is fairly large. The 6809 OS9 C compiler for the Color Computer is over 64K. It can be divided into modules, like is done with 6809 OS9 C (see my letter in issue 64). I've heard that the old PCC compiler used to be publicly available. This was 'the' official C implementation back when 'they' were practically giving C and Unix away just for the publicity. I don't know what its status is now. Probably locked in a vault somewhere.

Compiled basic could be somewhat useful. It wouldn't have to be a full basic, just a subset. Simple Basic compilers can be fairly small, and written in any language available. Still though, it would be Basic.

Forth has been implemented on just about every computer in existence. It's small reasonably fast, and extensible. It's also hard to learn. I've spent a year laboriously implementing a 6809 Forth for OS9 in assembler, and I still haven't been able to get the knack of Forthing. Of course, in all honesty, I haven't spent as much time trying to learn Forth as I did C or Pascal. I've had other things to do. Most *TCJ*ers are going to want a language they already know. I certainly would. Also, algorithms aren't as 'obvious' when written in Forth as compared to C or Pascal. Then there's the problem of a standard Forth. The saying 'If you've seen one Forth, then you've seen.... one forth' comes to mind. Forth 83 would be the logical choice to standardize on, but some are still using Fig-Forth simply because it's readily available. ANSI Forth is still in the future and will be several years before its out.

Macro Assembler would be interesting. Given a decent macro language you could come up with about any type of language you want. There are two catches thought. One is the phrase "Given a decent macro language'. The other is trying to find a macro assembler in source code. I've looked for six months and haven't been able to find one for any processor. I suppose you could use a macro preprocessor of some sort. <shrug> Not really practical, I just thought it was a novel idea.

Micro C is a C subset written by Dave Dunfield in Canada. I think his current version is 3.0 and contains structures, typedefs, multiple pointers, multiple dimension arrays, etc. Almost a full C. I'm not sure how much he sells his current version for. He sold his previous versions at $50 for the compiler and a generic P-Code output routine. You could also buy processor specific output routines for an additional $50. All in all, it sounds fairly good. You've got somebody already dedicated to small processors (Z80, 6809, 8051, etc.). And the product is already available. The only problem is the price. If somebody talked to him about it, he might be willing to do a reduced price version to TCJ subscribers, on the condition that we tell him of bugs etc. I have not talked to him about this and I'm not trying to obligate him. I just thought I'd mention it as a maybe. You could also try his old shareware version. I picked up a copy of V1.20 from a BBS for the cost of the phone call. It's got full source code and documentation. Of course, it doesn't contain structures, typedefs, and other niceties. Since the basic compiler package comes with a p-code output routine, that could allow generic 'assembly' and program transfers. Even the compiler could be distributed this way in case the target doesn't have a working C compiler. The compiler is available for a large number of CPUs, including the 8051, Z80, 6809, 8088. The only requirement is that the CPU has a 16 bit data register that can access the lower 8 bits, and a 16 bit index register. Even those could be changed some if you were willing to keep the 'registers' in memory.

Pascal could be a serious choice. ISO level 0 Pascal isn't worth using (I know, my OS9 Pascal09 is ISO level 0 with a few MINOR extensions), but most versions add a number of extensions. The only real problem is finding a compiler. Since there are so many versions, some standardization would need to be made (such as no strings or no recursion, or no nested procedures or some such). Many would already have one, but some would need one. I know of a couple in source code, but only one is self compilable. It is also very large, about 64K of compiled code (it uses sets heavily throughout the compiler.) It also generates P-Codes, although that could be changed or handled with assembler macros. With some effort, It might be possible to shrink the compiler to 32K or so, but I doubt it. A smaller compiler would need to be found. Of course, with source available we could always add extensions to make it more useful. If you could live with out self-compilation and some other things you could try Facilis, or even the Pascal/S. Pascal doesn't have the low level support that many would want, but still, Pascal is an easily understood language that is widely available. If we could find a nice generic Pascal compiler in source the we could add extensions.

I think I covered Small-C fairly completely in my letter in Issue #61.

Structured Basic overcomes many of the problems of Basic. Of course, not all computers have structured Basic. It's also still Basic. When I looked at the structured Basic (Basic09) that came with my OS9 Level 2, my first impression was that they tried to make Basic look like Pascal, and that if somebody was going to use a Pascalish language, then they might as well use Pascal.

As for interpreters, compilers, and translators...

An Interpretive language has many good features. Ease of use, ease of debugging, etc. It is S-L-O-W though. Also, most

interpreted languages don't support recursion well, or have parameters for the subroutines.

Compiled languages usually have much more power and are much more expressive than interpreted languages. Of course, debugging and ease of use suffer. Also, the compiler can be rather large. In this respect, Forth is excellent. It's a compiler and an interpreter and it's small. Of course, you have to understand Forth, and many don't.

A translator could bridge the gap between an understandable language and the usefulness of Forth. Several times I've thought about converting Small-C to output Forth instead of assembly. I never got around to it, but it was an interesting idea. The same could be done with Basic, Pascal, etc. The problem with this is that you now need two levels of languages, the translator and the Forth. Also, it wont be as efficient as natural (or would that be 'un'natural?) Forth. Of course, you can say the same thing about any compiler as compared to native code (assembler). This does give the users two chances to be able to run a program. Either using their own C or Pascal compilers, or the translators and whatever language they produce.

My suggestion would be to see if you can find a full C compiler. If not, try a group rate Dave Dunfield Micro C. If not, try Pascal. If that's unsuccessful, try for Small-C and Pascal to Forth converters. If that fails, you are either going to have to put up with Forth, or forget about a universal language for *TCJ*. If you do get a compiler, I suggest that it output P-Code that can be translated or interpreted by the user. This way, porting to a new computer would be just a matter of writing the translator and using the already p-code compiled compiler. That also allows generic 'binaries' to be distributed.

What every reader needs to do is send in a note saying what they have and what compilers are available. That way you'd know which to lean towards. Also, if anybody happens to know of generic, portable compilers, translators, Forths, etc, then they should drop you a note. That way we would know what is available to use.

Of course, having said all of that, I have to admit that in my case it wouldn't make that much difference. For my CoCo I've got a K&R C compiler, a (almost) working Forth 83, and an ISO level 0 Pascal compiler. I can already handle about anything you publish that I might want to try.

Sincerely, Carey Bloodworth.

Thanks for the short letter Carey...Actually your comments are very much appreciated as we consider how TCJ is to treat code examples.

After recent work on OS2 C, I must admit a reluctance to suggest C in any form. If Forth gets blamed for write once coding, C then must be write never. My experience proved that you can make any code impossible to read after being written. The use of Macros and procedures if left uncommented (the norm) can force you to search code, books, and files in the elusive battle of "what does it do and how do I talk to it????" chase. The ultimate example is the yearly contest to put as much as possible in one line of C code. I've seen the totally unreadable results and would hate to be a newcomer to computing trying to figure it out.

By the way, Forth ANSI standard is completed, several Forths in public domain do comply with the standard. I suspect that within six months the standard issue problem of Forth will be a thing of the past. Speaking of the past, I remember trying F83 (the Laxen and Perry version) that had both a Pascal and Basic converter. How well they work, alas I do not remember. Also a friend of mine is writing a C compiler in Forth (a nasty project due to C's oddball way of doing things) and hopes to be done soon.

The overall objective is not to use only one language at TCJ. The objective is how to get the most educational explanation of what the program is doing. Ideally I want to develop a universal working environment for all small systems, but that is secondary to teaching and providing "how-to" information, today.

Possibly we need to look at our problem in a different manner, as you have so aptly explained, the language choice is very complex and not solvable in any reasonable time frame. Maybe Lee Hart's comments will help us see a different direction.

Thanks again Carey for both of your letters. Bill Kibler.

Dear Bill,

Well, I found a house at last! The new address is in the letterhead above. Please update your files accordingly.

My check for $24 is enclosed for a *TCJ* subscription renewal. Keep up the good work!

On the choice of a standard programming language for *TCJ:* Language is intended to promote communication. Computer languages are designed to communicate with computers; they are NOT good for communicating with people. No matter what computer language you pick, many readers will have trouble fully comprehending it. After all, comments are necessary in program listings because even skilled practitioner can't understand the code without them. The comments form a pseudo-language that documents what is going on for people.

Essentially, your articles are trying to communicate with two different audiences; the computer and the reader. For the computer, any standard language appropriate for the platform under discussion is fine. If it compiles / assembles, it is good; the computer understood it. Just avoid non-standard or peculiar dialects. (This is why BASIC is a bad choice; there is no standard dialect.)

For the human-readable portion, I favor the use of a pseudo-language, such as ALGOL, when the goal is to illustrate an idea or algorithm for people. It doesn't really matter to people if the code is

correct, as long as the comments clearly describe the intent. I REALLY hate trivial or meaningless comments like "load accumulator" or "tweedledee ... tweedledum ... around and around ... until we're done."

On your coverage of the IBM PC as a classic computer. *TCJ* seeks to teach people. Teachers must be especially vigilant in choosing good examples. But the PC teaches...

-that bad hardware design can be hidden with enough megabytes and megahertz,

-that only Asians can build hardware,

-that only experts can write software,

-there is no point in learning to program; someone will write it for you,

-copy protection, shrink-wrap warranties, and selling known defective software will make you filthy rich,

-that stealing software is preferable to writing it,

-to spend more on advertising than product development,

-to use standards to block competition, not advance the state of the art,

-that connectors should be unlabeled, and never used for their customary purpose,

-that every program should include hardware specific I/O,

-and it's normal for a computer to "crash" every few hours.

I don't mind using the PC as an example, as long as it's explained to be a BAD example.

If we're going to promote old systems, how about the MAC? For $50 or $100 you can get a single board 8 MHz 68000-based computer with 128K to 1meg of RAM, 64K to 128K of ROM, two serial ports, video output, serial keyboard and mouse inputs, dual floppy disk controller, real-time clock, 256 bytes of non-volatile RAM, and 8-bit D/A converter. Later boards have SCSI port (but it only takes one chip to add SCSI to older boards). Mac boards are well designed and built, fast, reliable, easy to mount, and take a fraction of the power of a PC.

Here's a project: Can one of *TCJ*'s 68000 gurus provide a ROMable FORTH to replace or enhance the apple ROMs? These boards have two ROM sockets that accept either 32K (27256) or 64K (27512) ROMs or EPROMs. Early Macs had 64K of ROM (two 32K chips); later it increased to 128K (two 64K chips). The 128K ROMs are highly prized by Atari and Amiga owners to make their machines MAC compatible, so you can often buy a Mac board with empty ROM sockets for a song.

Suppose we burned a new pair of 26512 EPROMs, with Apple's 64K code in the lower half, and FORTH in the upper half. We add a patch so at power-up you can either run FORTH, or boot Apple's operating system normally. FORTH gives us Lee Felsenstein's dream of a hacker's Mac; all the power of the 68000 environment without Apple's repressive operating system. FORTH gives you total machine access, yet you can still use the Apple ROM's toolbox routines to use the clock, mouse, keyboard, screen graphics, disks, etc. Now suppose those 27512's are flash EPROMs, so they provide non-volitile read/write storage. 32K of FORTH can do amazing things. You have a very powerful disk less controller.

I loved the Timex/Sinclair stuff. I have always considered it the ultimate expression of simplicity; the Zen of computer design. A complete, working computer in 4 chips for $100. If you think we have progressed technologically, show me its equivalent today!

Clive Sinclair had a genius for this kind of work. He also built a watch, a calculator, a portable TV, and a multimeter with the same kind of ultimate simplicity and low cost. As for those who see it as a failure; not so! Sinclair entered a crowded market, and went from 0 to first place in sales in 4 months flat. He introduced millions to computers, and made millions of dollars doing it. That rates it as a success in my book.

Alas, the ZX81 is with us no more. But companies are like flowers. The chips are planted on a board. If they are of good stock, and the market is fertile, and is illuminated by the light of brilliant software, it blossoms into a thing of beauty. Then just after it blooms most brilliantly, it dies. But like flowers, it leaves behind the seeds for the next generation.

One has to wonder what a Sinclair computer would look like today?

Yours Truly, Lee A. Hart, Robbinsdale, MN.

I now understand why so many of my readers keep asking for you to do articles, Lee. Thanks for those great comments and you have said it better than I could.

Your comments on a language, made me consider that what I have been looking for is not a language as such, but really a proper and satisfactory way of commenting code. Your so correct in thinking that if we could just comment it in such a way that it expressed what was going on and needed to happen, that the actual language doesn't matter.

This idea also falls in with many FORTH programmers view, that FORTH is their own private language and tools for solving problems. Whether or not anyone uses the language is of no concern, what is important is that the language works and fits my personal style of programming. The problem then is really how do I comment and explain the language such that someone else (or myself) if needed, could convert it to another language on say another platform(especially since there will always be direct calls to the hardware I/O).

I really have learned to hate programmers who do not comment their code. The worst seem to be those who are making all those direct calls. Most likely they don't comment because they really do not know what it is they are doing. So, say nothing, or make funny com-

ments, and hopefully no one will notice you really can't program your way out of a paper bag....

I must confess that I have never seen ALGOL (that I remember) and would challenge you to do an article explaining it for us. That article might try to show how to comment properly (like "load the accumulator with the base address of table X", a bit more informative) and provide some comparisons to other languages for the same ideas(a comparison of C, BASIC, Forth, Assembler, Pascal, and ALGOL).

Confess again I must, that I had intentionally skipped over the Macs, under the idea that they were still too expensive (good hardware doesn't drop in price like a brick or PC) and that other 68000 machines were more available (I have Atari STs, one cost $25). I would love to do a series of articles on any of these 68000 based machines. For raw computing power and least amount of money, old Mac's, Atari's, Amiga's, and Sun workstations are great. Oh yes, I have a few readers suggesting some articles and projects using the old Sun 68000 workstations, since they too can be had for a song. TCJ's official position is that any of the older classic systems are better learning platforms than new PC's of any vendor.

Your idea of the ROM in Mac's is great and a bit like my idea of an universal operating system for these older machines. Think about taking an old Mac or Atari and plug in new ROMs and up comes the system running the same programs without converting or recompiling the code.

Yes I do know of someone just like Clive Sinclair, who is in fact building single chip computer systems with blinding speed, Chuck Moore. I commented on his newest MPU21 last issue. Chuck is really now breaking new ground in showing how the old stuff shirts and marketeers are just wrong in which way computing research and design should go. Chuck get's my Sinclair award for keeping it simple but extremely powerful.

Thanks for the comments and how about some articles on a regular basis? Your fans await you! Thanks again. Bill.

To answer Robert Edgecombe's letter in *TCJ* #63:

Even when you run the SYSGEN/ MOVCPM from the distribution disk, things can go wrong. I think it has got something to do with CONFIGUR utility but I'm not sure.

Anyhow, there is a fix for MOVCPM, I dug it up some time ago and since I was having the same troubles on my 820-II. I have tried it and it works! I even imported the Kaypro MOVCPM to the XEROX, patched it, and it still ran OK.

The fix comes from the INFO-CPM digest issue 29 in 1990. The source remains unnamed, but we have to thank Marc Wilson for bringing it out into the open.

Part of Marc's message follows:

Note that this information is specific to a particular copy of MOVCPM. Your patch point is almost guaranteed to NOT be in the same place. But, in looking at over a dozen copies of MOVCPM from as many manufacturers, I found that:

1)The code around the patch point always looks the same (that portion is DRI's, not the vendor's).

2)The patch point has always been within 80h bytes of the point specified in this file.

Also... I did NOT write this patch. I found it on a local BBS. Many moons ago. It's not my fault if you screw up your copy of MOVCPM. Do NOT do this on an original disk!

MOVCPM.FIX:

I recently tried to help a friend generate a new system on my machine, using his copy of MOVCPM, and we were greeted with, "SYSCHRONIZATION ERROR" followed by the machine quitting. After talking to another friend, I was informed that the problem was caused by a serial number mismatch between my system and his copy of MOVCPM. My friend further stated that there was "NO WAY" around this protection. After pondering the problem a while I decided to start disassembling MOVCPM with the help of the "L" command in DDT. What follows is the result of my efforts.

Beginning at 2C0 I found the following code:

```
-L2C0
02C0 POP  D
02C1 LXI  D,1200
02C4 LHLD 037A
02C7 MVI  C,6
02C9 LDAX D
02CA CMP  M
02CB JNZ  025A
02CE INX  H
02CF INX  D
02D0 DCR  C
02D1 JNZ  02C9
```

I then did the following substitutions:

```
-S2CB
02CB C2 00
02CC 5A 00
02CD 02 00
02CE 23 .
```

After the above changes do a SAVE 40 MOVCPMNU.COM and you have a version of MOVCPM that will run on any machine.

With the above changes under my belt I decided to do some more poking around in MOVCPM.COM and came up with the following addresses that might arouse your curiosity.

B5F, 1200, D28.

Best of Luck, A HACKER.

Actually "HACKER" is a real person, but just didn't want his name used. So thanks "Hacker" for your note, and sorry for the delay, your letter got mixed in with some other mail.

What you are doing is "NO OPing" out the "jump on NOT ZERO" of the compare of the six byte serial number.

MOVCPM compares the serial number byte at a time and if any byte is not the same, jumps to the message output routine and HALTS the system (actually I think it is a tight loop, but the results are the same, a locked up system.) If you do some looking you will find the serial numbers and it is possible to "ZERO" them out and solve the problem as well(most of my CP/M's have no serial number).

Good tip and thanks again. Bill.

Dear Mr. Kibler

I wish to extend my subscription which expires with issue 65. I also wish to purchase 1 to 58 to bring me up to date.

Now for the meat of the matter. When the new S-100 machines first came out I drooled for them. Later when they became "obsolete" I started running to hamfests looking for one. I finally found got my first last week. I found a Vector Graphics and am trying to get it running. The terminal seems to want 16 Volts it ain't getting. The hard drive spins, the floppy spins and seems to read a disk from my TI99-4A. I've got my voltages and clock and everything seems to run.

I need information as to dip switches for the following Vector boards if you can get it: ZCB, Bitstreamer, 64K Ram board, Flashwriter, Floppy/hard disk controller. I also need to know about the switches on a Televideo 925.

I must applaud you on the way you are running your magazine. Some of your letters in May/June issue have tempted me to write a small bit on the STD bus. I'm slowly working on it. Can you read a disk written on a TI 99-4A? If you want to wire wrap it, I've got a STD backplane I can let some one have.

Sincerely Roger Wykoff, 938 W. Outer Drive, Oak Ridge TN 37830.

Thanks Roger for those kind words about TCJ. First I would suggest talking to Herb Johnson about your Vector boards. It does sound to me like you might have something as simple as a non-working terminal or just wrong baud rates. I am not sure about the not getting 16 volts statement, as you do know that the regulators in S-100 are on the individual cards. That means the 12 volt bus voltage is typically 18 Volts.

As to the STD bus, yes I would like the backplane myself, and I am sure that my readers would love to hear about the STD bus, as unlike the S-100 it is still going strong. Chips like the 80451 and 68HC11 are taking away a lot of STD BUS jobs, but so many people are still using STD products everyday, I doubt the newer chips have any real impact on STD BUS applications.

Again thanks and do call Herb. Bill Kibler.

Dear Mr. Kibler;

Just received the renewal notice, about the same time as issue 65, thanks for the reminder.

I see that you are working with PLC's now. As a former boss said to me, "Welcome to the world of REAL computers". Which PLC's do you "specialize" in?

Along with the check for my renewal, I am enclosing a qualification form for "Personal Engineering & Instrumentation News". They usually have an article or two of interest every issue, a little "bleeding edge" stuff in most issues. Overall a nice balance, at least for me.

I picked up a DecMate II CP/M machine recently, and am learning how much I have forgotten. Now, to find some documentation, in case it breaks.

Good luck on your new job, let me know if you need any help in the mid-West or South.

Yours, Bobby Yates, Jonesboro, AR.

Thanks Booby, and yes it is nice to be back in the real computing world. I work on Omron PLC's right now, and doing mostly simple ladder stuff, with a very complex interface to non-PLC compatible 8051 systems. I have to use the BASIC interface and that brings back old memories of why I learned Forth - to get away from industrial BASIC! The PLC does things FAST, the interface module does thing in assembly FAST, then you must talk to the BASIC module and everything just about stops! I sure wish these vendors would drop their BASIC interfaces and go to FORTH, then I wouldn't have to curse at them so much.

Anyway Bobby, hope you find the DecMate documents, and I am already a subscriber to "PE&I", and yes they are mostly too bleeding leading edge most of the time. Their review of CAD and Schematic Capture programs was very good, which is why I still like getting it. Thanks for renewing and the letter. Bill.

Dear Bill,

Thanks for sending the free issue of *TCJ*. I was surprised to see your magazine headquarters so close to where I live - Lincoln is tiny! Anyway, I thought there was good information in it, especially the section on the old ZX80 and ZX81 clone. I had hoped to see something about these machines since I own a couple. I actually have a Microace computer which was another ZX80 clone but with 2K instead of just 1K. That was my first computer, but I later got a ZX81 for the extended BASIC and the non-blinking screen.

About 4 years ago I got a hold of a Tree Systems Pluri-Fourth chip - but I don't even know if it works because I could never get the ribbon for the keyboard back into its slot, and it eventually frayed and became unworkable. From the same distributor I also got another computer, a Memotech 512, which is like a C64 but with a Z80 microprocessor. Unfortunately it is not working at the moment. I have a magazine called SYNC, which was published for about 3 to 4 years and contains numerous articles and construction projects for the ZX80/1 and Timex computers. I want to keep the magazines (I have every issue but one, vol 3 #2) but I thought maybe some of your readers might be interested in photo-copies of

some of the articles, you could copy them from me and then distribute them to those who were interested.

One of your readers mentioned the ASZMIC rom and offered copies - is there any way to get an Eprom copy from you? I don't own an IBM or clone. And finally, at the risk of losing all possible assistance from you, I must say I was hoping for more construction articles. I don't subscribe to any magazines currently, I usually buy ones that contain something I can build. But thanks again for the free issue, and if your interested in copying any of the SYNC magazines give me a call or send me a letter.

Chris Ball, Yuba City, CA.

Thanks for the offer Chris, but currently I have my hands full producing TCJ. If anyone contacts me I will give them your name and address. There is a local (Auburn) copy business I use that can do those for a very cheap price if there isn't one in Yuba City.

I am not sure what happened to the ASZMIC ROM, as I did indicate a willingness to copy and distribute it. I imagine I filed away the information and have just forgotten to get back to the person to get it. I guess I better hunt around and find out what happened. You have found the main reason I didn't like the ZX81, the bloody ribbon cable, I replaced one with wires, and better yet a new keyboard!

I am trying to get more construction articles but it isn't easy these days. Those who are building are so busy they don't have time to write. I have actually had several offers, but the writers just don't get the time to put it down on paper. I will keep trying! Thanks again. Bill Kibler.

Dear Mr. Kibler!

My name is Alex Shakhnovich, and I am a computer hobbyist from the former Soviet Union. Now I live and work in the States (I am a chemist). I am in love with 8080 and Z80-based machines since 1984, recreating them, writing my own software, etc. Right now I am about to finish the construction of my third machine, that combines Sinclair-128 and SHARP MZ-700 (MZ-100 and MZ-800 are not probably know in the USA, but are quite popular in Russia because of good design and easy upgradability). By the way, lot of software, including several disk operation systems and a lot of applications has already been written by Russian programmers. I can share the information I have with other Z80-hobbyists. In exchange I need original Sinclair/Times software, and schematics/technical manuals for the computer and periperhals. So if you know some crazy people, doing the same things that I do, can you please give them my address? Thank you.

Alex Shakhnovich, 10001 Katelyn DR. Charlotte, NC 28269, SHAKHNOVICH@A1.malard.sandoz.com.

Ok Alex, I think I just gave your name to a few people who are interested in working with Z80's, all my subscribers. Hope the over load will not be too much for you.

I have been seeing a lot of information from Russian programmers lately, mostly from Forth Interest Group (FIG). Seems computing hardware was a bit limited, so your programmers had to be very innovative. That made Forth their number one language.

Since you said you are doing Z80 projects, how about some articles, especially the construction side of what you have been doing. I wouldn't even mind an article on the Sharp machines.

Well keep up the good work and welcome to the land of fun and plentiful Z80 machines. Bill Kibler.

Dear Sirs,

HELP! I have this here monstrocious AM-Varityper 3510 phototypesetter (advertised in *TCJ*) which uses 8 inch HARD SECTORED disks.

Continued on Page 50.

Articles Nededed

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

The subject matter of interest are mostly those which explain and teach readers how to perform intermediate and advanced improvements and modifications to their systems.

All of *TCJ*'s readers are not intermediate in skill, many are beginners. Articles need to take any reader of any skill level through your project, as if they were begining on this subject for the first time.

Areas of current interest are using older and obsolete systems for new embedded control situations.

Embedding operations in ROM and running the entire operations for remote sensing over a telephone line would be a great article of interest to our readers.

First hand reports on the history of early and classic systems is always a topic which our readers enjoy.

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

Short reports on projects that are currently under way, belong in our Support Groups section, where letting others know of what is being done has become a major focus.

Send your letters to:

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

Regular Feature

Intermediate Users

OLD PC/XT and PALs

PC/XT Corner

by Frank Sergeant

Day-Old Computing

Old PCs, Programmable Logic, XYZ Tables, and 68HC11

Introduction

Since the *theme* of this column is related to old PC/XT machines, I feel obligated to say a few words about them. I'll do that then jump right to a hodge-podge of topics that have been cluttering my brain.

XT Motherboard

I wanted an old XT motherboard to experiment with, as I have discussed putting Forth in ROM or setting up two-way communication using just the keyboard connector. Since I talked to you last, I bought one XT motherboard sight unseen mail order for about $15. This was a mistake. Arriving with *No* RAM was bad enough, but it didn't even have a BIOS ROM or CPU. I should have spent another $30 to $50 to buy an old '386SX motherboard instead.

Real Old PCs

All is not lost, however. We dug out some original PCs to use in the Small Scale Computer Systems lab where I teach. These aren't even XTs. Some have 256K and some 512K of RAM, one or two working floppies, a monochrome card and monitor and parallel port, no serial port. Suddenly I'm surrounded by old PCs. No one wants them. I even get to put one on my desk in the TA (teaching assistant) office. Wow! It takes an hour just to boot from a floppy (I'm exaggerating). By the time the date/time prompt appears I forget what I wanted to do with it. Oh, I remember. We are using them as terminals to talk to little 68HC11 boards we are building. For that we need serial ports. We bought some serial cards for about $7 each that didn't seem to work.

Hoist by My Own Petard

For years I've been using and recommending a quick and dirty serial interface for the PC. Technically, it violates the RS232 specs, but has always worked fine in practice. The idea is to use CMOS hex inverters to shift between the levels needed by the 'HC11 (5 volts = "1" bit, 0 volts = "0" bit) and the levels from the PC's serial port (about +12 volts = "0" bit and about -12 volts = "1" bit). The inverter's 5 volt output makes a satisfactory RS232 "0" bit, and, although the inverter's 0 volt output is 3 volts too high for a legitimate "1" bit, it worked on all serial cards I've tried. Until, we got these $7 serial cards. These cards work just fine if you give 'em real RS232 levels, but they DO NOT WORK with my cheap interface. I've finally been bit by taking a short cut. Let that be a lesson to me. Meanwhile, I took a few of my own serial cards to the lab. They work just fine with my cheap interface.

PC's Keyboard Interface is Bi-directional

I wouldn't have believed it if I hadn't seen the schematic. (Bill was kind enough to send me a schematic.) Even then I didn't believe it. I got out two more schematics. They all agree. Generally, the keyboard runs the interface, toggling the clock and data lines. But, these lines are "wire-OR" with pull-up resistors on them and open-collector drivers attached. Either end can pull those lines low. This may turn out to be our simplest way of talking to a bare board PC without having to add any cards to it at all. I'll let you know when I've had a chance to actually try this out.

Programmable Logic Devices (PLDs)

Anyone can make a friend, but only AMD can make a PAL (PAL is a trademark of AMD).

We are surrounded by programmable logic. Much of it is very inexpensive. In a previous article, I mentioned the possibility of introducing it to the computer architecture labs I teach at SWT (Southwest Texas State University). I'd like my students to use it because it saves chips compared to standard TTL and might let them do more work at the *logic* level if they had less physical wiring to do. It might also help them get jobs in the "real world." But, I worried PLDs would hide too much inside a single chip. Students might do better if they could probe the connections between standard TTL chips. One reader thought bringing internal logic points out to spare pins on the PLD would solve this problem. My compromise last semester was to demonstrate AMD's PLD compiler program PALASM by entering the equations and burning a PAL or two for them. Other than that, they used standard TTL. I'm planning to do about the same this semester.

Even if most of my students didn't learn much about PLDs, I had a lot of fun playing with them. They are easy to work with. The only real problem is the

cost of the PLD programmer, and I think I am getting closer to a solution to that.

What is a PLD?

All sorts of things can be called PLDs, including microcomputers (since they can be programmed to implement logic functions) and EPROMs. Generally, though, the term PLD refers to a programmable chip that has groups of AND gates feeding into OR gates. Each OR output is connected to an output pin, sometimes directly, and sometimes through a flip-flop and/or an inverter. Programming consists of selectively connecting or not connecting various inputs to the gates. If you can program both the AND array and the OR array, the PLD is called a PLA (programmable logic array). If the AND array can be programmed but the OR array is fixed, it is called a PAL (programmable array logic). If the device contains flip-flops, it is said to be "registered."

These various small PLD devices usually come in 20- or 24-pin DIPs. The name PAL suggests the array is programmed by melting fuses, as with PROMs. EPLD refers to PLDs with quartz windows which are programmed much like EPROMs (or are one-time programmable). EEPLDs (electrically eraseable PLDs) are programmed similarly to EEPROMs. GAL stands for Generic Array Logic and refers to an EEPLD which can emulate most of the fuse-based PALs. Essentially, they are all just collections of AND, OR, and NOT gates which can be connected as you see fit.

Truth Table

Take any number of input variables and fill out a truth table for an output function. Whatever the function, it can be expressed as a number of AND gates feeding into a single OR gate. This is called a "sum-of-products" form, because the OR gate produces a Boolean "sum" and an AND gate produces a Boolean "product." For example, you can express the exclusive-OR function of two variables as the sum of products /A*B + A*/B, which might be expressed in Forth as
A NOT B AND A B NOT AND OR.

Example of a Common PLD: The 22V10

The 22V10 has 12 dedicated input pins and 10 pins that can be either input or output or bidirectional. The OR gate connected to each output pin is fed by 8 to 16 AND gates. Each AND gate has about 44 possible inputs. The PAL (such as 22V10) directly implements the classic sum of products Boolean logic equations. Thus, each output consists of a group of AND gates (each with many possible inputs) feeding into a single OR gate. The programming determines which particular inputs will go into which AND gates. You have no choice as to which AND gates go into their corresponding OR gate. Furthermore, the output has a D-type flipflop which can be used or bypassed, and the output can be left true or inverted, or the output can be fed back to serve as the input to some other AND gate. It may sound complicated, but it all follows directly from converting your truth table into a sum of products.

How to Choose the Right PLD

It doesn't matter! They are all essentially interchangeable. The important differences are whether they are eraseable and reprogrammable and how much power they consume. In rare cases, speed might be a consideration, but for most glue-logic replacement purposes, the slowest are fast enough. For large volume production, you might want the "best" one. Otherwise, use whatever is available and eraseable.

A Cheap PLD Programmer

I've been wrestling with the idea of building a PLD programmer. Here are my latest thoughts. The main problem in using PLDs is the cost of the programmer. It looks like $500 is about the cheapest, and they go up from there. Why not just stick with TTL or CMOS standard logic chips? An extra chip or two at $0.25 each seems like a bargain. Besides, PLD chips are expensive. The AMD PALCE22V10 costs perhaps $6 to $7. Smaller PLDs are cheaper, and the ICT PEEL 18CV8 is now available from several distributors in the under $2.00 range. It is not entirely clear exactly how many off-the-shelf logic chips can be replaced by a single PLD. It depends on your particular circuit. Even if a PLD could replace 20 ICs in a certain circuit, if *your* circuit only needs one or two standard TTL chips, that PLD only replaces one or two chips!.

What's your price? If programmers fell to $200 would you jump on the PLD wagon? $100? One correspondent said we needed a $50 programmer. I've been trying to figure out how to build one, or a kit, and sell it cheap and still make a little money. I'm beginning to think $900 for a professional, commercial programmer would be money well spent. It is a very major project to build and support a "universal" programmer. There are a *lot* of devices out there, with new ones coming along rapidly. Each device type has its own programming specs. Some device manufacturers do not want to release this information unless you are already an established programmer vendor. Some of the programming algorithms are rather complicated, requiring super voltages at different times on up to 4 or 5 separate pins.

Even if it's worth it, you might not have $900 to spare. Is there a way out of this mess? Yes. Maybe. I've been thinking I might offer a kit for about $100 that would program only one or two types of PLDs (plus a bunch of EPROM types). It might even be a "semi-kit" where all you have to do is plug a chip or two into a socket. This puts it into the price range of an EPROM programmer, but with the PLD bonus. This is a one-size-fits-all approach and makes it easy to choose which PLD to use. You choose the only one the programmer supports! I'm leaning toward the ICT PEEL22CV10 which has programming specs I like, is electrically erasable and reprogrammable, and is a superset of most of the available small PLDs. I might also support the PEEL18CV8. I'd also like to support something like the

GAL 16V8, but that hinges on obtaining the programming specs.

But, even with a *free* programmer, why should you change over from standard logic chips to PLDs? Because every chip you save, also saves power, board space, holes to be drilled, and chances of wiring mistakes. Further, since the PLD is reprogrammable, you can correct or change your circuit without rewiring or making a new circuit board.

Example: Glue Logic on a 68HC11 Board

For example, before I started working with the PLDs, I wire-wrapped a 68HC11 computer with 32K bytes of external RAM. I used 3 glue-logic chips: a 74HC573 latch (same as a 74HC373, except for the pin-out), a 74HC00 quad 2-input NAND, and a 74HC4049 hex inverter. I suppose that is about a dollar's worth of chips. But, since I was wire-wrapping, it also cost me 3 wire-wrap sockets. They cost more than the chips. So maybe I'm up to about $3. Plus there is some wear and tear on me to wire it all up and decide where to put the chips on the perf board. I think it would have been worth spending an extra $3 or $4 to use a single PEEL22V10 to save the extra work and board space. As the number of chips which can be replaced grows, the advantage easily swings to the single-chip solution.

Inventory

Another benefit often attributed to PLDs is inventory simplification. Instead of keeping track of bunches of different TTL and CMOS chips, we could keep a single part in stock. Too many 7402s on hand? Not enough 7411s? With a PLDs, one size fits all. I'm not sure this is very important to us as hobbyists.

PLD Compilers

Another barrier to the use of PLDs may be their strangeness. Exactly how do you use them? Several chip vendors offer free PLD design software (PLD compilers). ICT (the manufacturer of PEELs) has APEEL, AMD has PALASM. I think National and TI and Intel also offer free PLD compilers. If you can draw your circuit using AND and OR and NOT gates, you can input those circuits easily into the PLD compilers. In PALASM, for example, if you wanted to enter a circuit of variables A, B, and C AND'd together and followed by an inverter, you could say /C = A * B * C where the slash indicates the complement (the NOT) and the asterisk indicates the AND operation. PALASM will take your raw equations and perform logic minimization for you. I'm not sure if APEEL does the minimization for you or not. Once you have compiled your logic equation, the software produces a "JEDEC file" to be downloaded to the programmer. Presto, your PLD is programmed.

EPROMs are PLDs

Let us not forget that a 2K x 8 EPROM (a 2716), which can be had for under $1.00, is a programmable logic device. It can generate any 8 functions of up to 11 input variables each. A $3.00 32K x 8 EPROM (a 27256) can generate any 8 functions of up to 15 input variables each. The address lines are the logic inputs and the data lines are the logic outputs.

As an example, consider using a 32K x 8 EPROM as an ALU (arithmetic logic unit) which operates on 4-bit operands. 4-bits for the first operand, 4-bits for the second operand, plus a carry-in would use up 9 of the 15 input lines. The other 6 input lines could choose which of 64 logic or arithmetic functions to perform. The output could be 4-bits plus a carry-out, plus perhaps a zero-flag, overflow-flag, negative-flag, for a total of 8 output bits. If speed of operation were no object, we might see how few EPROMs it would take to build a minimal computer for instructional use.

MC68HC11 Microcomputers

B.G. Micro at (214) 271-5546 in Dallas has been selling MC68HC11A1 chips in a 52-pin PLCC package (i.e. a square, package designed to be surface mounted) for around $7 each. Still available? The 'A1 has 256 bytes of on-board RAM and 256 bytes of on-board EPROM plus timer, serial port, I/O ports, and 8 ADC (analog-to-digital converter) pins.

The best price I've found for 'HC11s is from Beall & Glenn Enterprises at 1-800-874-4797. They have 68HC11A1 chips at $2.85 and 68HC11E1 chips at $3.00. For a $20 order they pay shipping. Otherwise, add $2.00.

Marvin Green at 821 SW 14th, Troutdale, OR 97060 has some nice looking blank printed circuit boards for prototyping 'HC11s. They include a small wire-wrap area and come with suggested schematics. 3 blank boards for $17. I have some, like their appearance, but haven't soldered one up yet. I'm balking slightly because I enjoy using the cute little blue ceramic resonator (from DIGI-KEY) and Marvin's boards are set up for a real crystal instead.

PLCC Wire-Wrap Sockets

I had been reluctant to use PLCCs for prototyping because of the difficulty of connecting them, compared to using DIPs. However, I have now tried two different ways to connect them that work well. Both methods use a PLCC socket, which converts the PLCC to a pin-grid-array style pin-out. The first way, which I tried on a 68-pin PLCC DSP (digital signal processing) chip, was to use a Radio Shack perf board with separate copper pads on each hole. I stuck the socket into the board and soldered 30 gauge wire-wrap wire to each socket pin. The copper pads help: you just strip about 1/8 inch of insulation from the wire, stick it into the hole, and solder the wire, pin, and pad all at once. The first twenty pins or so were very frustrating, but after that I had the hang of it and the rest went smoothly. The trick is to pay no attention to the destination of the wires. Cut each wire long enough so you can wrap it anywhere on your board. You end up with a mess of wires, so you need to prepare a column and row "map" (like a spreadsheet, row A, column 7, for example) to make it easy to identify each pin.

The second way, pointed out to me by a correspondent, is to take DIP wire-wrap machine pin sockets and cut them into

single strips of the right length. Stick the strips onto the pins of the PLCC socket. Presto. You have a wire-wrap PLCC socket. I think it takes 6 strips of 7 pins and 2 strips of 5 pins for the 52-pin PLCC used by the 'HC11.

If you have $19 to spend, the CGN1001 wire-wrap module works well. We are using them in the Small Scale lab. Just plug in an 'E1 or 'A1 chip and add power and ground and a serial interface. They come with crystal, socket, jumpers to put chip in special bootstrap mode, and pull-up resistors on the interrupt and reset pins. CGN Company: (408) 720-1814.

Obviously, I've been fooling around with the 'HC11 lately. I used the 'D0 version in a 40-pin DIP for the Bare Bones EPROM Programmer, mainly because it came in a DIP. This makes breadboarding it in a plastic breadboard easy. Now that I know how to wire-wrap to a PLCC, I'll probably switch over to using the PLCC 'A1, 'E1, etc. The latest board I wire-wrapped has the 'A1 plus 32K bytes of external static RAM. It talks to my PC over a serial line. I can download programs to the internal or external RAM and set or read its I/O pins, all from the comfort of Forth on the PC.

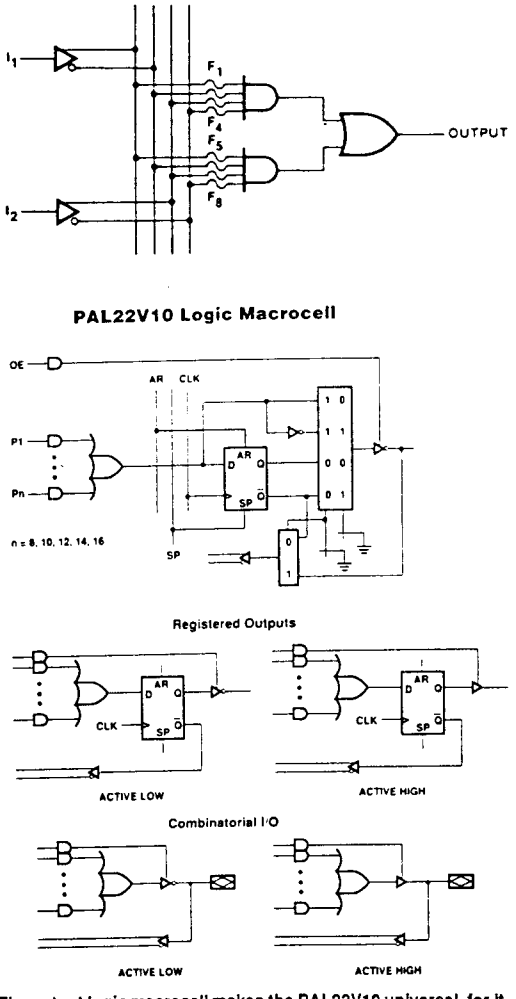
Different Versions

There are some interesting differences between versions of the 'HC11. The 'E1, for example, is nearly identical to the 'A1, except it has 512 bytes of on-board RAM instead of 256. The 'E2 has 2K bytes of EEPROM, instead of 512 bytes. The 'D0 has only 192 bytes of RAM, no EEPROM, and no ADC (analog to digital converter). All the versions have a bootstrap mode which allows you to download a program to the on-board RAM and execute it. This is one of the nicest features of the 'HC11. As far as the instruction set, the 'HC11 is essentially a 6800 with extensions.

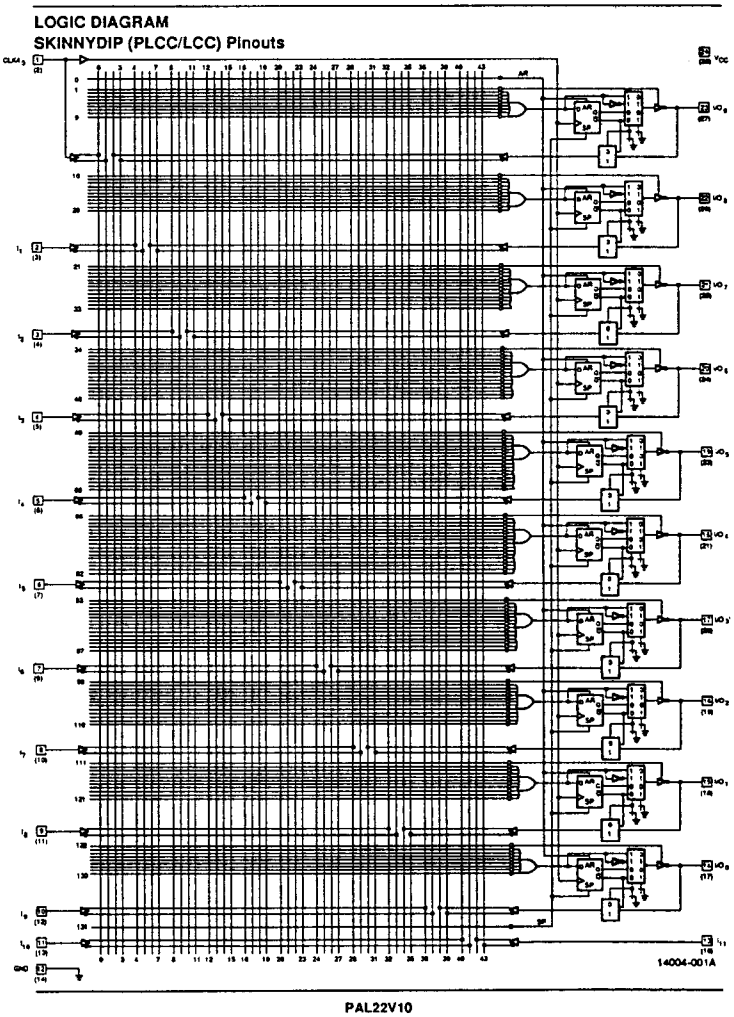
More to Say

I've got more to say about XYZ tables, making printed circuit boards, CAD packages for drawing schematics, and computing the convex hull of a finite set of points (huh?). I'll try to say it all next time, and even include a little about PC/XTs. As always, I am delighted to hear from you, especially by email to F.SERGEANT on GEnie or fs01@academia.swt.edu on Internet.

An unprogrammed PAL device has all fuses intact.



The output logic macrocell makes the PAL22V10 universal, for it can substitute for virtually all of the standard 24-pin PAL devices on the market.



PAL22V10

Some examples to help you understand PALs.

Regular Feature

ZCPR Support

Failsafe Scripts Part 3

The Z-System Corner

By Jay Sage

Techniques for Running Unattended

Part 3: The Command Scripts

Two columns ago I presented the control scripts I developed to allow my DOS computer to run a long sequence of tasks unattended over a lengthy period of time, during which the computer might suffer a power failure or have to be rebooted for some other reason. In my most recent column (a couple of issues back) I showed how PMATE (ZMATE) macros could be used to analyze the results of my circuit simulations so that 4DOS batch command scripts could carry out complex analyses without human intervention. This time I will present an example of those command scripts. Although I am describing my specific application to the simulation of an electronic circuit, the techniques have very broad applicability, both in the context of unattended computing and in other situations.

4DOS Release 5

As usual, I first have an aside. Since my last column was written, JP Software has released a major update to the 4DOS command processor that makes possible the command scripts I have described in the past and will describe in this column. Before getting down to the main subject, I would like to mention briefly a few important additions in 4DOS 5 (currently at revision level 5.0D). I would also like to reiterate my strong recommendation: if you use MS-DOS at the command prompt with any frequency, get and install 4DOS in place of COMMAND.COM.

Error Handler

One of the few areas in which ZCPR3 (Z-System) maintained a superiority over 4DOS was in error handling. Those of you who are familiar with Z-System know that when you enter an incorrect command in a command line, the command processor does not just complain about it, skip over it, and continue running whatever comes next, possibly with catastrophic consequences. Instead, it brings up a special program called an error handler that allows the user to deal with the problem. Most often, the error handler displays the problem command and allows the user to edit it.

Some time ago in TCJ I mentioned that I stopped by at the offices of JP Software in nearby Arlington, Massachusetts, and spoke with 4DOS author Tom Rawson about this issue. The results of that meeting can be seen in Release 5. Now, when the 4DOS command processor cannot execute a command, it will invoke an alias with the name UNKNOWN_CMD, passing to it the offending command line as an argument. This gives the user the hook needed to perform whatever special processing is required. I have only begun to play with this new capability. I let UNKNOWN_CMD pass the command processing task off to a batch file by defining it as "CALL C:\4DOS\ERROR.BTM". The CALL command is used to invoke it so that it will work properly even with errors encountered inside another batch file. Without CALL, control would not return to the original batch file after ERROR.BTM ran.

My first cut at the implementation of ERROR.BTM (I'm sure it can be improved on) is shown in Listing 1. It assigns the bad command line to an environment variable, displays a message to the operator, and then allows the operator to edit the command line, using another nice new feature in Release 5. The INPUT command, which gets a string from the user and assigns it to a specified environment variable, used to initialize that variable to a null string. Now it supports the "/E" option to allow editing of an existing value. Finally, the revised environment variable is issued as a command.

The DO Loop

Another extremely handy feature added in 4DOS Release 5 is the DO command for use in batch scripts. It was always possible to use IF and GOTO statements to accomplish the same things, but the DO command makes programs easier to write and to read. It comes in the following versions:

```
DO n
DO FOREVER
DO var = start TO end [ BY n]
DO WHILE condition
DO UNTIL condition
```

The end of the loop is designated by the ENDDO command. The commands LEAVE and ITERATE allow one, respectively, to leave a DO loop entirely or begin a new iteration immediately. You will see an example of a DO loop later in my NOISESWP script.

My Circuit Simulation Problem

Now on to the main subject! Just this past month I was faced with the need to study noise margins in the new circuit I

have been developing, a digital shift register using what are called Resonant-Tunneling Diodes or RTDs. I took an earlier version of simulation schematic and added a new voltage source to represent noise induced in the circuit. I wanted to determine the range of this voltage over which the circuit would continue to function correctly for various choices of other circuit parameters.

Since each simulation run takes considerable time, I wanted the calculation to proceed automatically, without my having to examine the results at each step and then decide how to change the noise voltage. Moreover, I wanted the computations to run both day and night and in the background while I was doing other work. Since the computer was not going to be left unattended, there was no need for the fail-safe facilities I described in previous columns, but I did want to make use of the automated evaluation technique I described last time.

Overview of Approach

Here is how I organized a given simulation run. The main circuit schematic and the simulation specifications were contained in a file that we will call NOISE.CIR. To make it easy to vary certain parameters, their values were defined in a separate "include" file called NOISE.PAR that the main file was told to read in. It is very easy, as you will see in a moment, to have 4DOS "write" that file.

The PSPICE program was then invoked to run the simulation, after which PMATE (ZMATE), using a macro called NOISE.MAT, would analyze the result and determine whether or not the circuit performed properly. The noise margins were determined by running a sequence of these simulations with different values of the noise voltage. A 4DOS script controlled this process, adjusting the value and determining the limiting values for proper circuit operation.

Command Script 1

I generally try to approach programming tasks in a modular fashion. Therefore, I first wrote a script to run a single simulation with specific values assigned to the key parameters, including the noise voltage. This script is shown in Listing 2. It consists mostly of lines that use the ECHO command with output redirection into files ('>' to create a new file and '>>' to add to a file). Some of the lines generate the NOISE.PAR file that contains PSPICE statements to define the values for the parameters. Other lines add lines to the text file NOISE.RES that documents the results of the simulation runs.

A few lines do something more sophisticated, but I will explain only a few of them here. One is the line that generates the ".TRAN" statement in the PAR file. This PSPICE command tells PSPICE to run a transient signal analysis over a specified time interval (second argument) with output values printed at specified subintervals (first argument). For my problem I want the simulation to print output voltages at each half cycle of the clock and to run for a total of seven half cycles. A half clock cycle is three times the rise/fall time of the clock as given by the parameter rftime.

4DOS supports not only environment variables but also environment functions. In this line we encounter two instances of the powerful @EVAL environment function, which evaluates arithmetic expressions involving addition, subtraction, multiplication, real and integer division, and modulo operations on integer and real numbers.

Here are a few other items worth commenting on. In most DOS batch files, comments use the REM (remark) command. Somewhere I read that a double colon is a more effective way to enter comments. It certainly makes for easier reading. In several places in the script an ECHOS (echo string) command is used to echo a line without a carriage return and linefeed so that additional text can be added to the line. In one case it is combined with the TIMER command to generate a line that looks like the following:

03-06-94 Timer 1 on: 22:03:12

It shows the date and time when the timer was started. Later a TIMER OFF command records the ending time and the elapsed time for the simulation run.

Command Script 2

The second command script is where 4DOS really shines. The function of this script, called NOISESWP.BTM (noise sweep), is to vary the value of the noise parameter until the most positive and negative values at which the circuit works correctly have been determined. See Listing 3, which contains a somewhat abbreviated version of the script.

The script begins, as most of my scripts do, by checking the syntax and presenting a syntax message if their is a problem. If that test has been passed, then the work can begin. The SETLOCAL command pushes the system state -- including the values of all environment variables and aliases and the current subdirectory -- onto a stack. When the batch file ends (or an ENDLOCAL command is executed), the original state is restored. The command-line parameters are then assigned to named environment variables, to make the script easier to read and to allow the values to be modified.

Next some information is written out to a file called NOISE.SUM that records a summary of the final results of the simulation runs. Another of 4DOS's three timers is used to measure the elapsed time for a series of runs. The utility script FORMAT.BTM takes the name of an environment variable that holds a numerical value and puts it in a format occupying a specified character space and with a specified number of digits after the decimal point

Two environment variables -- Nneg and Npos -- are used to store the magnitude (absolute value) of the largest negative and positive noise voltage levels for

proper circuit operation. They are initialized to zero.

The first simulation run is made with the noise voltage (represented by the variable TSTVAL) set to zero. After all, it makes no sense to scan for circuit margins when the circuit doesn't work at all! The subroutine TEST initiates the actual PSPICE run and sets the variable FLAG to GOOD to indicate success or BAD to indicate failure.

Now the script gets more interesting. To find the limiting value for the noise voltage to high precision but quickly, a binary search approach is followed. It is implemented in the subroutine BINARY. The environment variable SIGN is given the value POS or NEG to indicate whether positive or negative noise voltages are being tested. The value of TSTVAL is always positive. This seemingly overly complex approach was adopted because of a bug in 4DOS that prevents numerical comparisons in conditional tests from working correctly with negative numbers (I've reported this to Tom Rawson and hope he will issue a Release 5.00E with a fix). BINARY is called twice, once to get the positive limit and once to get the negative limit. The results are formatted to two decimal places and written out to the NOISE.SUM file.

The BINARY subroutine starts out with a noise voltage step size DELTA of 0.32, an initial value VALUE of 0.00, and a scan direction DIR of UP. The variable BIN is set to zero to indicate that the code is not yet in the binary mode in which the value of DELTA can be halved at each step. The initial step size has to be maintained until the first circuit failure is encountered.

Now we begin a DO FOREVER loop that is eventually terminated by a LEAVE command. If the scan direction is UP, then the step DELTA is added to VALUE; otherwise it is subtracted. If SIGN is POS, the variable TSTVAL is set directly to VALUE; if NEG, TSTVAL is set to the negative of VALUE. Then the subroutine TEST is invoked to perform the simulation.

If the result (indicated by variable FLAG) is bad, we set variable BIN to 1 to show that the binary search mode can be followed. If BIN is still 0 at this point, we start the next iteration immediately. Otherwise, we cut the step DELTA in half. If this makes it less than our desired precision of 0.01, then we are finished and leave the DO loop. Otherwise, we set the step direction to DOWN if the circuit failed at that noise voltage or UP if it performed correctly. We then continue with the DO loop.

Each invocation of the script NOISESWP determines the noise margins for one set of circuit parameters. Typically I write one more level of scripting. This one invokes NOISESWP with many different sets of parameter values. Sometimes I use the FOR command (and even nested FOR commands) to sweep one or more parameters. Once this master script is running, I can forget about it and turn my attention to other work. When the whole series of series of series of simulations is finished, I just have to plot the results (you don't suppose I have scripts to do that, too, do you!).

Plans for Next Time

In response to a request from Bill Kibler, my next subject will be CP/M emulators under DOS and especially the marvelous MYZ80 emulator. All of my real CP/M machines have suffered fatal hardware failures, but MYZ80 allows me to continue running them in emulation. With a powerful DOS machine, the emulation actually runs faster than the real machine! As you have probably noticed, my writing schedule has been spaced out, partly to make room for many new contributors to TCJ and partly because my schedule gives me less time for writing. So you can expect my next installment two or three issues from now.

+-

Listing 1. A first cut at an error handler script for 4DOS Release 5.

```
@echo off
*set badcmd=%&
beep
echo.
echo The following command could not be executed:
echo.
echos '--> '
input /e %%badcmd
echo.
call %badcmd
unset /q badcmd
```

+-

Listing 2. The NOISE.BTM script that runs a single simulation with parameter values specified on the command line.

```
@echo off

:: This script starts a run of NOISE.CIR and
:: invokes PMATE to perform an analysis of
:: the results, which are recorded in the file
:: NOISE.RES. The following variables must
:: be provided  on the command line:
::          1. the rise/fall time in ps (rftime)
::          2. latch RTD area ratio (ratio)
::          3. coupling RTD area ratio (couple)
::          4. clock high level (clkhi)
::          5. clock low level (clklo)
::          6. noise voltage (noise)

iff !%6==! then
color bri yel on bla
text

This script makes a run of NOISE.CIR and,
using the PMATE macro NOISE.MAT, writes
the results of the run to the file NOISE.RES.
The syntax is as follows:

endtext
echo    %0 rftime ltch_ratio cpl_ratio clkhi clklo
noise
text

The value of rftime must be in picoseconds.
endtext
color whi on bla
quit
endiff

:: Generate the parameter file for the circuit
:: and write the parameter values into the result
:: file. A timer is started to record the time for
:: the run.

echo. >>noise.res
echos %_date:'  ' >>noise.res
timer on >>noise.res
echo. >>noise.res

echo .TRAN %@eval[3*%1]ps
%@eval[21*%1]ps >noise.par

echo .PARAM tau    = %1ps >>noise.par
echo  rftime  = %1ps >>noise.res

echo .PARAM ratio   = %2 >>noise.par
echo  ratio   = %2 >>noise.res

echo .PARAM couple  = %3 >>noise.par
echo  couple  = %3 >>noise.res

echo .PARAM Vclk1   = %4 >>noise.par
echo  Vclk1   = %4 >>noise.res
```

```
echo .PARAM Vclk0    = %5 >>noise.par
echo  Vclk0    = %5 >>noise.res

call format.btm noise 0 2
echo .PARAM noise    = %6 >>noise.par
echo  noise    = %6 >>noise.res

echo. >>noise.res

:: ------------------------------------------------
:: Run the simulation and analyze the results.

c:\pspice\pspice1.exe noise.cir noise.out

iff %? gt 0 then
echo   Run did not terminate normally
>>noise.res
echo. >>noise.res
iff exist noise.out then
c:\commands\edit.com $ b9e xi noise.mat $ .9
echo. >>noise.res
endiff
timer off >>noise.res
echo. >>noise.res
echo ----------------------------------------
>>noise.res
beep 440 3 880 3 440 3 880 3
inkey /w20 /k"yn" Abort entire run? %%key
if %key==y cancel
tail -18 noise.res
quit
endiff

:: Have PMATE perform its analysis.

c:\commands\edit.com $ b9e xi noise.mat $ .9
echo. >>noise.res
timer off >>noise.res
echo. >>noise.res
echo ----------------------------------------
>>noise.res
tail -18 noise.res
beep 440 3 880 3 440 3 880 3
echos 'Paused for 5 seconds . . . '
delay 5
```

+-

Listing 3. The script NOISESWP.BTM that uses a binary search algorithm to sweep the noise parameter to determine the range of values over which the circuit operates correctly.

```
@echo off

:: A large block of comments describing how
:: this script works has been omitted here.
:: The required command line parameters are
:: the same as the first five with NOISE.BTM.
:: ------------------------------------------------
:: Check syntax and write header into
:: summary file.

iff !%5==! then
{ A syntax message that appears here has
been omitted. }
quit
endiff

setlocal
set rftime=%1
set ratio=%2
set couple=%3
set clk_hi=%4
set clk_lo=%5

echo. >>noise.sum
echos Run of NOISESWP
started: %_date at %_time' ' >>noise.sum
timer /2 on >>noise.sum
echo. >>noise.sum

:: Format variables and write header to
:: summary file.

call format.btm rftime 4 0
echo Waveform rise/fall
time.....: %rftime ps >>noise.sum
{ similar lines for other parameters omitted }

echo. >>noise.sum
echo    noise   performance >>noise.sum
echo    ----  ---------- >>noise.sum
echo. >>noise.sum

:: Initialize noise margins

set Nneg=0.0
set Npos=0.0

:: Check noise=0 case.

set tstval=0.00
gosub test
if %flag NE good goto finish

:: Binary scan over positive values

set sign=pos
gosub binary

:: Binary scan over negative values

set sign=neg
gosub binary

:: Write out summary of scan.

echo. >>noise.sum
call format.btm Nneg 0 2
call format.btm Npos 0 2
echo The noise voltage margins
are -%Nneg to %Npos >>noise.sum
:finish
echo. >>noise.sum
echo Run of NOISESWP completed.
>>noise.sum
echos %_date at %_time' ' >>noise.sum
timer /2 >>noise.sum
echo. >>noise.sum
echo ----------------------------------------
>>noise.sum
cls
tail -23 noise.sum
quit

:: ------------------------------------------------
::          SUBROUTINE BINARY
:: ------------------------------------------------
:: This subroutine performs a binary search for
:: the maximum value of 'noise' that works.
:binary

*set delta=0.32
*set value=0.00
*set dir=up
*set bin=0

do forever

iff %dir == up then
*set value=%@eval[%value+%delta]
else
*set value=%@eval[%value-%delta]
endiff

iff %sign EQ pos then
*set tstval=%value
else
*set tstval=-%value
endiff

gosub test

if %flag EQ bad *set bin=1
if %bin EQ 0 iterate

*set delta=%@eval[%delta/2]
if %delta LT 0.01 leave

iff %flag EQ bad then
*set dir=down
else
*set dir=up
endiff

enddo
return

:: ------------------------------------------------
::          SUBROUTINE TEST
:: ------------------------------------------------
:: This subroutine carries out one analysis
:: point and writes the results to the logging file
:: NOISE.SUM.  The variable "flag" is set to
:: the value GOOD or BAD (or UNK).  The
:: variable "tstval" carries the current value for
:: the noise  voltage.
:test

if exist noise.gd  del noise.gd
if exist noise.bad del noise.bad
if exist noise.unk del noise.unk

call noise.btm %rftime
%ratio %couple %clk_hi %clk_lo %tstval

iff exist noise.gd then
set flag=GOOD
iff %sign EQ pos then
if %value GT %Npos set Npos=%value
else
if %value GT %Nneg set Nneg=%value
endiff
elseiff exist noise.bad then
set flag=BAD
else
set flag=UNK
endiff

call format.btm tstval 5 2
echo    %tstval      %flag >>noise.sum

return
```

32-Bit Systems

All Readers

TCP/IP & OSI

Real Computing

By Rick Rodman

Tiny-TCP

After much fiddling around, Tiny-TCP is almost working. I have been finding and fixing several minor bugs in the TCP and FTP layers; the TCP layer is working fine now, and I expect the problems with FTP will be fixed by the time you read this. Then comes the fun of porting it to a variety of machines.

Here's a brief description of the software. At the lowest layer, (see side bar on OSI layers) you have the Internet Protocol (IP). This layer defines only a message structure. Message packets are framed, checked and passed to the next layer. Addressing uses the four-byte IP Address scheme, usually depicted as four decimal numbers separated by periods. IP sits atop a low-level driver, which I will get into in a moment.

Above IP is the Transmission Control Protocol (TCP). This layer is session-oriented - you establish a connection, send data, and then hang up. The messages are sequence-checked so that they arrive in the proper order and are passed to and from the application.

The application, in our case, is the File Transfer Protocol (FTP). This package is quite easy to use, transferring using ASCII commands and providing help messages. Other applications can also be coded, talking to the TCP layer using what is sometimes called a "socket library interface".

Besides TCP, there are other protocols which can sit on top of the IP layer: Address Resolution Protocol (ARP), User Datagram Protocol (UDP), etc. Tiny-TCP supports a tiny subset of ARP, but none of the others.

Tiny-TCP itself consists of five C files. The main one is TINYTCP, which contains the IP and TCP layers. TINYFTP is the FTP application. ARP is the ARP routine. The fourth of the original source modules, SED (Simple Ethernet Driver), is a driver for a 3Com Multibus Ethernet board which most of our readers won't have. In its place I have written a SLIP driver called SEDSLIP. SLIP stands for Serial Line IP, and is a very simple, and commonly used, way of throwing IP packets out on a serial link. The last module is called MAIN and supplies the main program and real-time clock interface.

For testing, I have been using an IBM P70 portable PC running PC-DOS, and on the other end of the wire, a Dell PC running OS/2 and IBM's TCP/IP for OS/2. The method behind this madness is to ensure that the final package will be compatible with genuine TCP/IP SLIP. Next to the Dell is a DEC Rainbow which will be the next target for the code.

Porting this code will mean modifying the SEDSLIP routine for your serial port, and modifying the clock driver in MAIN for whatever you have available. If you don't have anything, it would probably work OK to simply add a value each time the clock routine is called. On the PC, it appears to be impossible to receive data on a serial port reliably without using interrupts - even on a 486, and at 1200 baud. So much for progress.

Once everything is working, you will be able to send files to, or get files from, any machine on the network, from any other machine. It will only be necessary to issue commands from one end of the connection. The basic capability we are aiming for here is simple file transfer. Don't start dreaming about running X Window on your Kaypro. (By the way, has anyone tried running Ladder on one of those emulators? Or M.U.L.E., one of the best computer games ever written?)

One really great thing about TCP/IP is the price of the documentation: free. You can get any of the RFCs which specify each feature by requesting it through E-mail. You send an empty mail message with a subject line of "RFC nnn" to *service@nic.ddn.mil*, where nnn is replaced by the RFC number. Some RFC numbers of interest are: 793, for TCP; 791, for IP; 768, for UDP; 959, for FTP; 826, for ARP; 821, for SMTP. Since this is a mail service, everyone can play. Sometimes things that are free are worth every penny, but these are pretty good.

Now, recall that the only communications capability which is common to all of these disparate machines is the RS-232 port. This is the normal way of using SLIP: bidirectional, low-speed serial links, sometimes even over modems. But if all of the connections are point-to-point RS-232 connections, and you have more than two machines, routing will be required, which is an IP-layer function. Routing means receiving a packet from one serial port, examining the destination address, and sending it on another serial port if it's not for the receiving machine.

I've considered two possible approaches to this routing situation. One is throwing together a 2-board S-100 system with a CPU and 4 to 8 serial ports, which will do routing and nothing but routing. The

other is to use the PC-532, with its multitasking capability and many free serial ports. But there is also a hardware alternative approach: the RS-485 network.

The RS-485 network

Tilmann Reh has proposed an alternative where the RS-232 ports would be level-translated onto an RS-485 bus, which would function somewhat like a low-speed Ethernet. The low-level driver would be a modified version of SEDSLIP which would "listen before speaking"; in the unlikely event that two machines spoke at the same instant, their retransmission delay would be slightly different. This would eliminate the need for the routing box. All machines would receive all messages, but would ignore messages for other machines. Other approaches using microcontrollers with 9-bit interfaces have been discussed as well, but the need to keep costs low has eliminated these.

Here are some notes from Tilmann's messages: "Now to the physical interface. Yes, RS-485 is a bus. But to start at the beginning: We have several common serial interfaces out there. I will give you a brief description of them in a senseful order:

RS-232C: unbalanced, single TX, single RX, max. 20k bps.

RS-423A: unbalanced (coax), single TX, max. 10 RX, differential receiver, max. 100k bps.
RS-422A: balanced (STP cable) counterpart to RS-423, single TX, max. 10 RX, max. 10M bps.

RS-485: serial bus interface, max. 32 transceivers, upward compatible to RS-422A, max. 10M bps."

In *TCJ* #49, the problem of "zapping" through RS-232 data leads was extensively discussed. As RS-232 cables connect machines on different power circuits, noise spikes (ground noise) are transmitted through the RS-232 cables from one machine to another. This noise bypasses the power supply filtering, coming right into the most delicate circuitry. This is another reason for thinking about other approaches besides a mesh of point-to-point RS-232 links.

Tilmann writes: "For your information: Ethernet LANs are also insulated. The coupling to the network cable is done with a small transformer. So the cable itself is 'floating' and not connected to any protective (or circuit) ground. This would also apply to our RS-485 net if we insulate all transceivers." Doing so may require one or more isolated DC-DC converters, which adds cost to the circuitry, however.

"I just had an idea about how to avoid that extra line. What if we used a monoflop (one-shot) to enable the transmitter. The trigger could be the transmit data. In order to safely switch on before we start to send, we would have to send a single start bit (data FFh) to trigger the one-shot. The time constant of the one-shot must be chosen according to the baudrate we want to use. This approach of course has some difficulties, but it would allow for real portable hard- and software!" Actually, this would not be much of a problem. The SLIP driver works by sending out IP frames separated by C0 hex bytes. An extraneous FF byte would be ignored.

I'm leaving the hardware design to Tilmann, and concentrating on the software side myself.

Sprite

I haven't tried to load Sprite yet. Its authors characterize it as an experimental distributed operating system; when users started clamoring for all their favorite Unix features, they decided to take it down instead, to avoid the burden of support. It's supplied on CD-ROM with a monumental amount of source code and documentation. One of the interesting ideas they used was a "Log File System." In this file system, there is no directory structure per se. Instead, the serving computer simply keeps a big log of everything that the user does to all the files. Then, if anyone needs to go back and read something, the real work comes: starting at the beginning, it reconstructs the current image from all of the changes. This dramatically improves file-writing speed, which is often a network problem, but one would think that it would provide poor reading speed. Fortunately, most files are simply read and written as whole units, rather than being randomly read and written at disparate times.

As I mentioned, I haven't actually run any of this code, but reading some of the documentation really got my wheels turning. I've often wondered why we put up with the limitations of the way files are on all of our current operating systems. Notice that you can only add data to the end - never to the beginning or middle; and a few operating systems allow you to delete from the end, but never from the beginning or middle. Lately I've been trying to design an efficient disk queuing mechanism which allows for multiple writers, and these limitations make it a real headache. Presumably, the reason is the block nature of disk storage and low-level decisions to forego byte-addressibility - which, really, only pushes the overhead out of the operating system into all of the user programs.

This is a fundamental nature of computing. Overhead doesn't ever go completely away - if you push it out of one place, it reappears everywhere else. For example, the killer feature of Unix, which guarantees it can never succeed, is case-sensitivity. Some will argue that it can be handled in the user programs - but the difficulty, hence the cost, of checking for upper and lower case is tremendously greater in a user program. So, to spare a few dozen lines in the kernel, the computing world has paid, and continues to pay, the price of millions of programmer-hours and megabytes of source code. It would be tragic if it weren't so stupid - and so commonplace.

Macintosh fans like to remark that IBM's decision to use the 8088 in the PC was the costliest mistake in history. It's hard to compare that with other historical gaffes, for example Napoleon's invasion of Russia, but in terms of pure dollars and cents, I think they have a good point. But the whole history of computing is the history of shortsighted mistakes just

like that, and we can expect a whole lot more of them to be made in the future.

The future belongs to you, *TCJ* reader. When your turn comes, take the extra time and do it right. Go ahead and design for the ages. Someday, maybe, someone will appreciate it.

Next time

There's more to cover in the TCP/IP land from the Linux side. Linux's TCP/IP support has been criticized as deficient, but it's actually not bad at all. Then we can move to other networking topics and, from there, back to the basics of Real Computing. Additionally, I may take time out to discuss X Window and MPEG. There's no limit to what you can do with computers - they're the greatest toys - er, *tools* - in the universe.

Where to call or write

Real Computing BBS or Fax: +1 703 330 9049
E-mail: rickr@aib.com
Mail: 8329 Ivy Glen Court, Manassas VA 22110

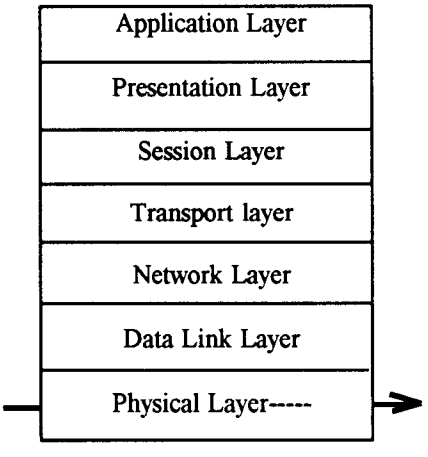
The OSI and Physical Layer

The International Organization for Standardization (ISO) and the CCITT have standardized on a reference model for networking, often referred to as the Open Systems Interconnection (OSI) reference model. This model partitions network functionality into seven layers. These are, of course, arbitrary partitions, and different networking packages will partition themselves into fewer layers, usually, for efficiency.

The bottom-most layer is the Physical layer. This layer concerns itself with how a 1 or a 0 looks on the physical network media. Above that is the Link layer, which defines a packet structure atop the physical media. At this layer, collision detection would take place. The next layer up is the Network layer, at which routing and connections between machines is handled. Atop that is the Transport layer, which concerns itself with moving data across the network; then the Session layer, which establishes and takes down routes across the network.

The Presentation layer is concerned with the standardization of the meaning of the data, things like ASCII and EBCDIC, and the Application layer is concerned with the use of the data, for example, for word processing files. The distinctions between the layers are very subtle and ambiguous; but, as pointed out, it doesn't really matter because actually using this many layers would be grossly inefficient.

In TCP/IP, the IP layer corresponds roughly to the Link and Network layers; the TCP layer corresponds roughly to the Transport and Session layers. FTP, as the end user of the data, takes the rest.



Many physical configurations are possible for our small network. The normal way in which SLIP (Serial Line IP) is used is in a point-to-point arrangement. This would require that at least some of the machines be able to forward packets (fig. 1).

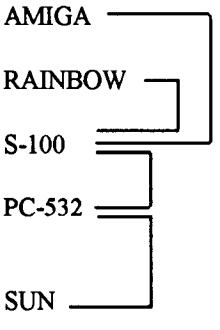


Fig. 1. All links are bidirectional.

In the point-to-point arrangement, some of the machines would have to be on at all times, if they are performing router services for the other machines. In figure 1, the S-100 and PC-532 machines have to be running for the Amiga and Rainbow to be able to connect to the Sun.

Ideally, of course, all machines would be connected directly to each other. But, while S-100 systems can easily be equipped with lots of serial ports - I've seen systems with over 30 ports - newer machines like the Amiga and PCs usually can't have more than one or two. This is called "progress" - I don't think.

A simple scheme which allows interconnection of all the machines with a single port on each is called the "serial token ring". This is shown in fig. 2. The data out from each machine's serial port is connected to the data in on the next machine.

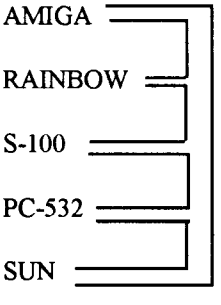


Fig. 2. All links are Unidirectional.

While visually quite elegant, this scheme has a number of drawbacks. All of the machines must resend all received pack-

ets which are not for them; this means that the networking software must be running at all times on all machines. In the worst case, the packet has to be sent and received by three machines before the fourth gets it. This could cause significant delays, at the speeds being proposed, if the entire packet must be received before retransmission.

The RS-485 bus approach is shown in figure 3. In this approach, all machines receive all messages, but ignore any that are not for themselves. When sending, the machine would wait for a certain amount of quiet on the bus before sending. If he received no response, there would be a timeout before retransmission.

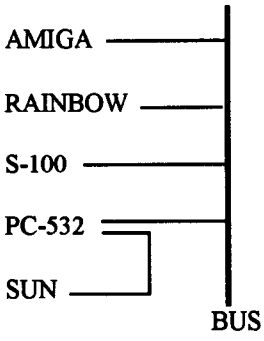


Fig. 3. All links are bidirectional.

The bus approach has many advantages. It doesn't matter whether any machine is on or off, unless you want to talk to that machine. Also, there is no routing, so any packet you send goes directly to the recipient with no forwarding - and no delays. On the other hand, the possibility of collisions could slow things down. In figure 3 I show the Sun being connected with a point-to-point link. This is because I don't think I can modify the Sun's SLIP driver for non-point-to-point configuration.

From the user's standpoint, there is little difference in any of the scenarios shown. This is because most of the implications of the connection scheme are contained within the Physical layer, with small changes at the Network layer.

Regular Feature

Intermediate

Spring Letters

Dr. S-100

By Herb R. Johnson

"Dr. S-100's Spring column" by Herb Johnson (c) Mar 1993

Introduction

Well, my recent contract job ran its course, through the worst winter of 40 years in New Jersey. The bad news about the 14th snowstorm is that it cost me a transmission - it literally blew itself up! The good news about the completing my contract is...more time for my IMSAI, Compupro and my columns! Actually, my writing career was enhanced recently by the publication of my article on astronomical image processing in Observatory Techniques in their Winter 1994 issue. If enough TCJ readers are interested in microprocessors, CCD's, and telescopes, I'll clue you all in on this revolution in astronomy which lets amateurs do professional-quality observing even in the glare and fog of the city!

Meantime, I can now address the enormous letter pile. My regrets to those requesting information or a source for the WonderBoard from Fritz's ComputerWorks, or who have a WonderSystem to sell: a reply is on the way! A few letters of note come to mind:

Eyes on the Skies by CompuPro

More news from Robert Grey of Chicago, who uses a Compupro 8/16 system to operate a radio telescope. I was able to dig up a few Disk 1 (floppy controller) cards for him for parts, and a Interfacer 1 (I/O) card to get him back on the air! He's still interested in upgrading his 8085/8088 processor card to a faster 8085 or Z-80: I'll try to get him a "plug and play" Z-80 card configured in the near future. (If you read my previous TCJ article, I described how Robert has conducted a SETI (Search for Extra Terrestrial Intelligence) program for the last ten year, listening at the radio wavelength of 21 cm.)

One of the nice features of Compupro systems is the wide range of cards they provided over the years, including Intel 80286 and Motorola 68000 processor cards, hard disk and RAM disk cards, and networking cards. I haven't checked lately but I imagine they are still in business, servicing their industrial business and development customers. If anyone has had recent dealings with them, I'd be curious. Robert would like to help a few other S-100 users: "I've got a few used Compupro boards: Interfacer 3, Interfacer 4, and Ram 20 (32K static RAM card), plus some software and docs."

As for Robert, he is now copying his collected data from 8" disks to a PC-compatible 486 system for further analysis. Although he admits that system has more computational power than his 8085-based Compupro programs, it would need an IEEE-488 interface to operate his Hewett-Packard spectrum analyzer, and a number of optically-isolated I/O lines to operate the motors of the dish. In addition, all his graphic and control programs would need to be rewritten, and with new hardware drivers to boot! Robert has wisely chosen to maintain his investment in Compupro equipment, while moving the data. As I am also an amateur astronomer, I hope to work with Robert to make his data available to other amateurs and professionals for further study or just curiosity!

IDE and S-100...coming soon?

Just before my deadline, I read a letter from Claude Palm of Palmtech in Queensland, Australia. "My firm has designed a single-chip IDE [hard disk drive] interface for the S-100 bus. I have named it PT IDE100. It is based on a [programmed logic chip] in a 68-pin PLCC package. As the main author of S-100 related articles in TCJ, I thought you may be interested in this device. [!]"

"I enclose some preliminary information on the PT IDE100 and a suggested application for the chip which I call the HARDBOARD. I have a point-to-point [i.e. hand-wired] soldered prototype complete with FDC [floppy disk controller] working at present with no problems...Incidentally, this letter was written on that system...The PT IDE100 is the first commercial S100 product I have become involved with, and I am testing the waters to see if it is worth pursuing this or any future S100 products."

I immediately FAXed a letter of interest and accepted his offer to evaluate his chip and prototype design. I was surprised that evening when he called me to discuss details! Claude is a designer who uses the S-100 bus as a prototyping environment for interface design, "because the bus is so straight forward". He tells me that it was "easy" for him to design a similar one-chip interface for the Hitachi HD64180 (Z180 compatible) processor, which was also on the same system as the IDE100. I will be evaluating the costs of a PC board layout for his

IDE interface, and the effort required to make his BIOS software "commercial", as it was derived from licensed software.

It's too early to determine costs, but I suspect a S-100 card with IDE, ROM, and floppy might approach $150 due to small-production costs. Claude and I have also discussed making the chip itself available, and the possibilities of a single-board Z180 system he is developing. I can understand Claude's caution in "testing the waters" for S-100 products, considering the typical S-100 system today is acquired for $100 down to zero. However, I think a few boards may yet be sold! I need to hear from my Faithful Readership on this: what do you need on an S-100 IDE interface to make it work for you? And, more to the point, will you pay the price? Write or call quickly while I have some opportunity to work on this with Mr. Palm.

S-100 is not the only bus service in town...

John J Fiorino of Brooklyn NY makes me an offer I can't refuse, and gives us a few lessons in old systems use. "Regret the delay in responding to your note...the snow has made it impossible to get the car out of the garage! I certainly would like [your] SWTPC [SS-50 bus] 6809, 64K 8 inch drives and I/O card with [the] FLEX [operating system]. I have just the same system new, which has never been hooked up and running. I'm still using the SWTPC [Motorola] 6800, but I'd like to upgrade to the 6809. Your machine would be looked at as spare parts...I would like to pick it up myself as it would be easier for both of us. You would not have to lug it down to the post office and I would not have to wait for the delivery truck."

[By the way, this is a very effective argument! I've packed many systems and shipped them cross-country, sometimes in pieces to get the weight down: and sometimes in pieces after they arrive! "Cash and carry" is an important bargaining tool.]

John would like to hear of other SS-50 system users and sources for "old chips and 360K diskettes and drives": "I have a lot of software for the 6809 and a large amount of 8-inch diskettes. I'd like to set up the 6809 now, seeing that I could now have backup." Another bit of wisdom from John: keep two systems running, one for parts and repairs! The easiest way to test a bus card, whether S-100 or SS-50, is to "swap" it into another working system.

Rick Rodman, my TCJ colleague, asks if I have a Compupro System Support 1 card and manual: I do, and I usually can make them available for a modest charge. Any other readers that have manuals available, or whom need manuals, for S-100 cards? Drop me a line or call!

Kaypro's and Cromemco (?)

Richard de Nobel of Silver Springs, MD says "I am the owner of a Kaypro 4 which I acquired as my first computer at age 63. I've been having some fun with it! So, I subscribed to TCJ and a few other small clubs here near Wash. DC. I've got CP/M Version 2.26 up and running after much difficulty: I don't know why, but mostly no documentation. I have a copy of MicroSoft Basic-80 rev 5.21 up and running also. I even learned how to configure it a little, and now have a serial printer LC-50 on the printer port."

"I would like to change a few things, like how to change the default 132 character line for LPRINT, the line editor is not good, etc. I have a programming (Fortran) background and understand quite a bit and I'm not too bashful about trying a few things." Richard would like to find docs for Basic-80 "unassembled": anyone know about this?

"One thing I came across was that Cromenco had at one time an excellent group of programs for [their] System 2 or 3 that included COBOL, FORTRAN IV, and a 32k Structured Basic!!... I gotta ask you about [whether] Cromenco software will run on a Kaypro 2X DSDD diskette or Kaypro 4. I would also like to find a better BASIC and line editor. Is any of this software available? Also, I have a friend with a Northstar Horizon, so if you would care to comment on support in software for that machine too."

Hmmm...Well, Cromenco is still around, but I don't think you want to spend hundreds of dollars for their software. The problem is that they didn't use CP/M as their operating system, but a look-alike called CDOS. While it has many of the same features of CP/M and some of the same "DOS" calls, it is not 100% compatible. And like most "big guy" commercial software, they did not provide sources to end users. I've heard of CP/M emulators that ran on some Cromemco systems, and there are CP/M's configured for Cromenco systems, but I don't know offhand of successful "conversions" of CDOS commercial programs. I'm not a CDOS "guru" by any stretch...any readers care to give me a clue here?

However, your real interest is in a more powerful BASIC. You should get together with a PC-compatible friend and get ready to buy the CP/M CD-ROM: it's bound to have some alternatives to 8K Microsoft Basic! Or, contact the vendors in this magazine or as cataloged in the Z-Letter (see the ads) for their "best" BASIC's and Fortran's.

Call me IMSAI

Walter Rottenkolber of Mariposa CA apparently reads my articles, as he gives me best wishes "in your new house" that I wrote of some time ago. "I read your Dr S-100 articles in TCJ, because I bought an IMSAI a few years ago and am in the process of bringing it alive again. It has a set of CCS [California Computer Systems, a good S-100 vendor] boards -- CPU, static memory (4 - 16K cards) and [floppy] disk controller. The only glitch is that the 8" drives were sold beforehand, and all I was able to get was a 5.25" SS drive. With a friend's help, I transferred the CCS CP/M files over to a Kaypro disk."

"I adapted a multiformatter, a multidisk program, and a sysgen so that I could use my Kaypro to read/write to the A400 (CCS diskette) format. The CCS System

file is now on the system tracks of the IMSAI disk, for the moment as a 20K CP/M." [Translation: he wrote programs to format disks for the CCS system, and copied the CCS CP/M over to them.]

"When I start the IMSAI, the system comes up in the ROM monitor. All the ROM functions work, except BOOT. This locks up and the system seems to go into an endless loop." [Walter should be able to confirm this by single-stepping the CPU, provided he has adapted the IMSAI front panel to fully operate the CPU: this may not be easy. Pins 20 and 70 are often grounded on later S100 cards,but must be at least "floating" for the IMSAI front panel to operate.]

"I load the CP/M manually this way..[with the following ROM commands]:

```
PO 12 0
        ; sets parameters for SSSD,
        ;18 sectors, 128 bytes each.
QO 0 2
        ; sets side 0, track 0, sector 2
R2C00 4600
        ; reads from disk to memory from
        ; 2C00 to 4600
S2C07 --> 07-00
        ; zeros out count byte in
        ;CCP buffer to prevent a autoload
        ;of the extended BIOS. The
        ;standard BIOS will run the A400
        ;drive and the serial port.
G4200
        ; jumps to the CP/M Cold boot
```

The logon message appears and then the A> prompt. Now the 'fun' begins. All keystrokes appear on screen as ^@, i.e. CP/M's character for binary zero [character 0]. When you type in enough characters, a new CRLF and A> prompt is outputted. I dumped the code and it looks OK. Most of the BIOS follows the ROM code. So, I have the strange situation where CONOUT apparently sends chars OK to the terminal but CONIN (these are CP/M DOS calls) is locked on 00H. However, there is no problem with R/W when in the Monitor."

Well, I congratulate you on some ambitious programming!

Why are you zeroing out the CCP each time? Why not put the zero in on the diskette sector once and forget it? For my reader's sake, Walter is disabling the command string in the CCP string buffer by telling it there is a zero-length (no) string. CP/M can 'auto-start' a program or command at boot time by embedding the command in the CCP buffer as part of the CCP code on the boot tracks. Typically, this is a way to load in more BIOS code at bootup, or to execute a .BAT file at bootup.

After you have loaded your current CP/M and BIOS, try loading in or hand-entering a simple console echo program (read keyboard, send character to console), making BIOS calls. Confirm the BIOS works as CP/M expects. Write another console echo program, making BDOS calls instead of BIOS calls. Does it work?

Another tact you can try is to verify that the CP/M is properly relocated. Dump memory and see if the JMP instructions make sense: are they located where they should be as compared to a similarly sized CP/M on your Kaypro? Also, I'd try a bigger CP/M, say one for a 32K system. Meanwhile, if the front panel is working, you should step through the code from the A> prompt after you type in a letter, and see what is going on. It would seem a shame to have an IMSAI front panel if you can't use it to step through code!

Since you have a Kaypro, and since you clearly have the knowledge and information on the CCS system, I'd suggest you write a simple, stand-alone BIOS for the CCS hardware that would make no ROM calls at ALL! You can abstract out the disk read/write code from the ROM by disassembly and include it into your BIOS. Test it by writing a code fragment to read a number from the console from 1 to 18, which will read that sector of the "current track" into a buffer. Do the same for reading sector 1 of track 0 to 9 as entered: you get the idea. As for the BOOT code, you can use your earlier trick to load it in via ROM commands. After doing all this, you might find the solution to resolve your current problem, or simply work around it.

Quick notes

Victor Lypka has a system with some IMSAI I/O and memory cards, a Solid State Music video card, some ROM and I/O, and a Tarbell cassette. Anyone interested?

Thanks to James C Matthews of Montgomery AL, who has offered to help rewrite some Northstar BIOS routine to support their disk controller on my Processor Tech SOL system. Seems that NorthStar started out as a supplier of disk controllers for SOL users before offering their own systems!

References

Robert Grey, 3071 Palmer Square, Chicago IL 60647.
Observatory Techniques,
John J. Fiorino, 518 85th Street, Brooklyn, NY 11209.
James C Matthews, 2028 Merrily Drive, Montgomery AL 36111
Walter J Rottenkolber, p.o. Box 1705, Mariposa CA 95338
Victor Lypka, 9201 Oday Drive, Highland IN 96322. 219-924-1315.

TCJ Center Fold

Mr. Kaypro

By Charles Stafford

Special Feature

Intermediate

Advent Decoder Board

WHEREIN We Undertake The Construction Of An Implant

In The Beginning, there were ANALOG computers, and only HIGHLY QUALIFIED WIZARDS (HCWs) were allowed in the same room with them. These ANALOG computers conversed in varying voltages and currents and all was well. Since they were HARD-WIRED and constructed of potentiometers, coils, meters, and relays, they were RELIABLE if not inscrutable.

Then Came The Binary and Boolean Algebras, and VACUUM TUBES and later SEMI-CONDUCTORS, and MICRO-PROCESSORS, and DIGITAL computers. The HCWs eschewed their ANALOG computers and defected en masse to the new gods of speed. These DIGITAL computers conversed in 0s and 1s and their communications were voluminous, and required the efforts of KEYPUNCH OPERATORS (a lesser form of HCW) and PROGRAMMERS (a special breed unto themselves), and these communications were contained in many, many boxes of cards which were carefully warehoused.

The Quest for Speed continued, and MAGNETIC TAPE was re-discovered, as was ROTARY MOTION and the DISK DRIVE was born and mutated into several varients. And the HCWs were ecstatic, because only they could see the BITS and BYTES on the diskette.

Today the common sizes are 5.25" double-sided double-density (DSDD) (360k), 5.25" high-density (HD) (1.2mb), 3.5" DSDD (720k), and 3.5" HD (1.44mb), with 3.5" very-high-density (VHD) (2.88mb) on the horizon. (I made up the VHD designation). Most KayPro s were delivered with the 5.25 DSDD variety, the exceptions being the K-2s, which had 5.25 SSDD 180k drives, and the Robies and K-4Xs which had 5.25 2.6mb DriveTec drives. The DSDD drives in the KayPros had a capacity of 390k, 10% more than their IBM brethern, but more is better, right? Sometime during the drive evolution, the QUAD density drive was developed, with 96 tracks per inch instead of 48, resulting in 160 tracks total and a capacity of 790kb.

Some of the more enterprising HCWs transplanted these DSQD drives into K-4s and thus were born K-8s. Microcornucopia and Avent Products formalized the modifications and they became very popular since they were within most budgets and HARDDRIVES were extremely expensive.

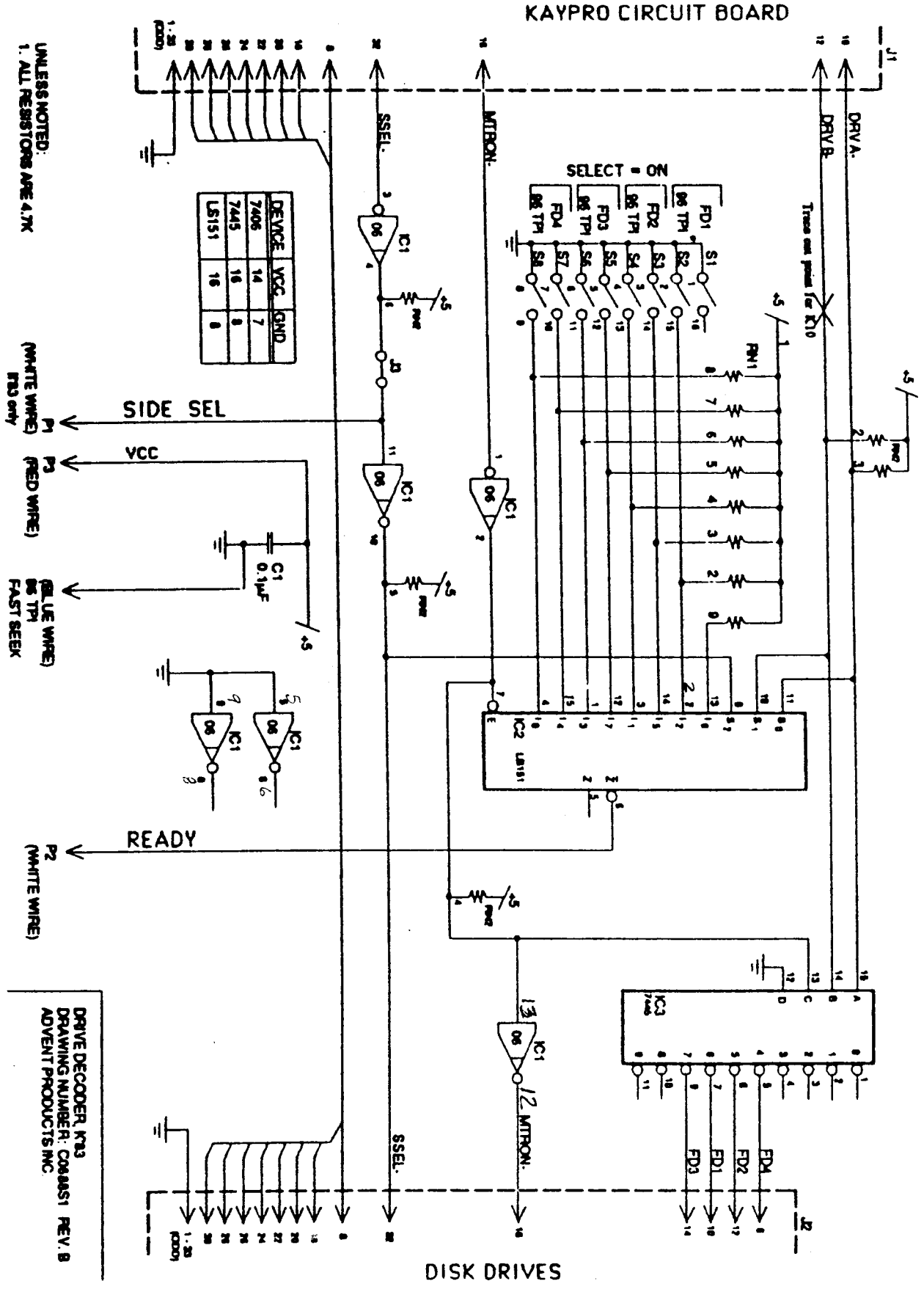
The two conversions used different designs based on their respective monitor roms. The MicroCornucopia (MicroC) monitor rom uses a utility program to configure the drive bios and thus only requires a multiplex decoder on the drive select lines to select the proper drive. The KayPro mother-boards were designed with only two drives in mind, so only drive select A and drive select were implemented. By multiplexing both lines we can come up with four choices, and one of four drives. The drawback to using the MicroC rom is that when you install a harddrive, the Micro-C rom won't boot off the harddisk.

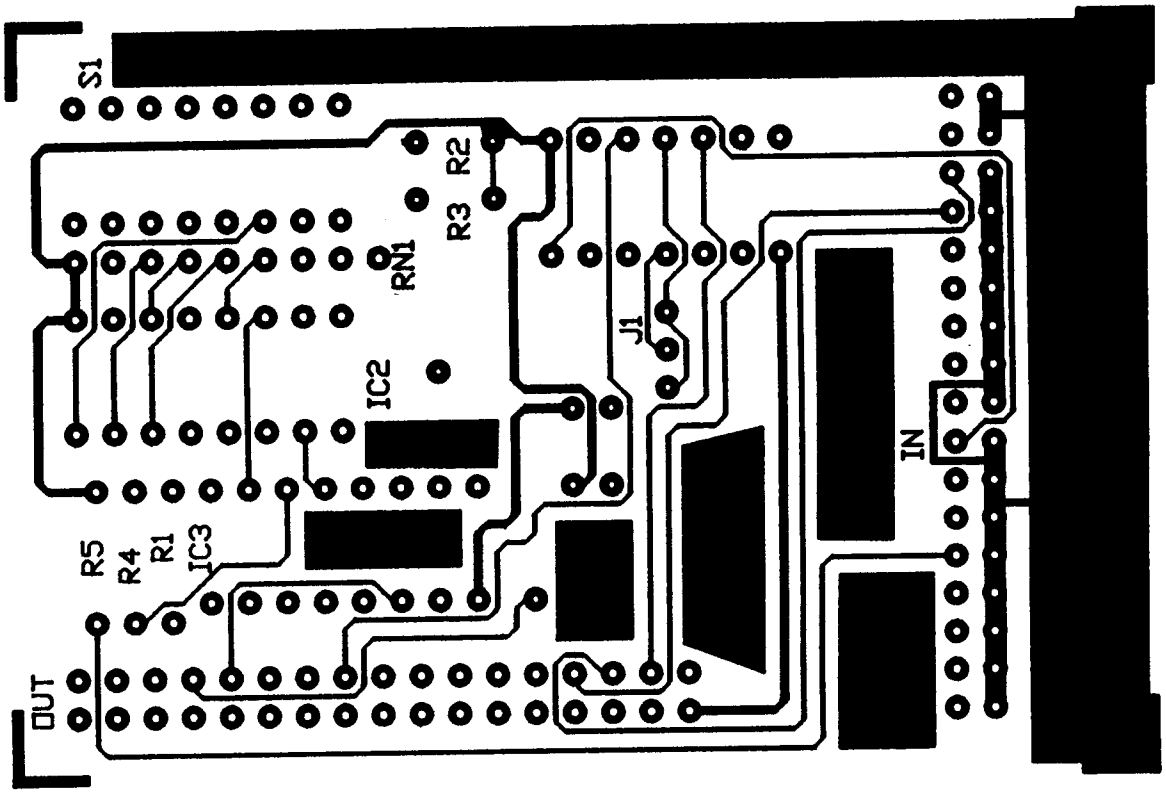
The Advent TurboRom on the other hand will boot off a harddrive, but dynamically configures the drive bios based in an inquiry to the "Personality-Decoder Board". The advantage is that reconfiguring drives only requires only rearranging the hardware and setting the switches on the PDB. The disadvantage is that the PDB is more complex. Unfortunately Advent is out of the CP/M business and the stock (theirs and mine) of PDBs is completely gone. Fortunately, we can build one fairly easily.

We will attack this project on two parallel tracks. One, with a custom printed circuit board that you can make, if you feel up to it, and, the other, with an off the shelf prototype board. The difference is where you put the labor, into point-to-point wiring, or into the manufacture of a printed circuit, which will be considerably smaller.

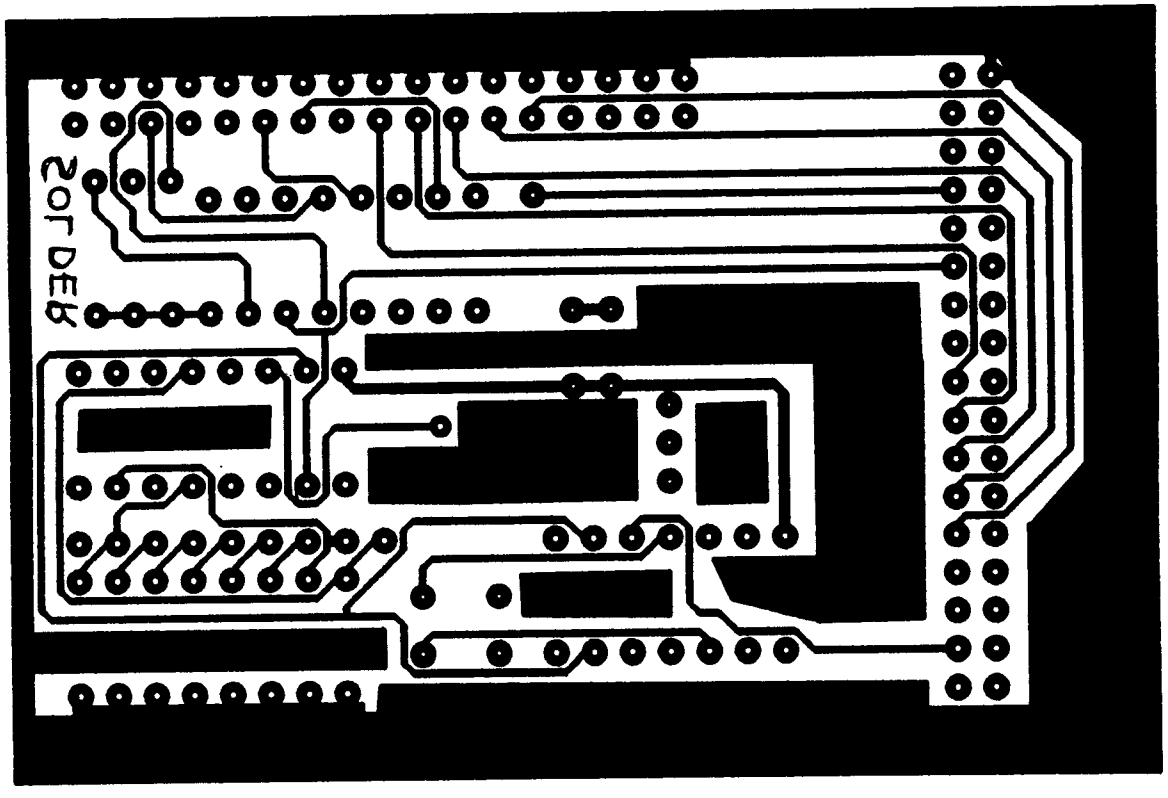
This will also be a two article project. We will start with making the custom circuit board, and do the assembly in the next issue.

Some time ago I mentioned that I knew nothing about making circuit boards and Ed Fanta called me and volunteered to make some for the PDB. The process that seems the best for the novice is the copier method. Here are Ed's notes on the entire process and the "plots" as well. NOTE These plots are twice as big as you will want for the finished product. Reduce them to half size on the copier you are using and they'll be just right. Experiment on plain paper before making the mask material to make sure the size is proper. You can check the size by measuring the distance across eleven pins of an IC. The spaces


KAYPRO CIRCUIT BOARD
UNLESS NOTED:
1. ALL RESISTORS ARE 4.7K
SELECT = ON
SIDE SEL
VCC
READY
DISK DRIVES
SSEL-
MTRON-
DRV A-
DRV B-
P1 (WHITE WIRE) FD only
P3 (RED WIRE)
(BLUE WIRE) 96 TPI FAST SEEK
P2 (WHITE WIRE)
DRIVE DECODER, K'83
DRAWING NUMBER: C08AS1 REV. B
ADVENT PRODUCTS INC.



TOP or component side



BOTTOM or solder side

should be 0.1 inch each, so the distance across ten spaces (eleven pins) should be one inch. The complete directions for use are on the package of material for the masks. Both the mask kits and etchant are available from most electronic hobby stores.

Ed's notes:
Hello Chuck
Things are finally winding down a bit, the light at the end of the tunnel is getting closer. I am getting time to write this and Eric found the time to plot out the etching masks. The following is a few ideas about the PCB process.

Etch Tank

Find a tall, narrow tank. A good start is look in the housewares department. No, no, not the kitchen cabinets, go to a department store and look over the plastic wares. The one I found most suitible was a beverage container for dispensing softdrinks, it is about 10 inches high, 8 inches wide, and about 3 inches deep with a snap on lid. A tray for holding the board to be etched might be made from a smaller such container or, as I have used, a cut down antifreeze jug. The tray should have holes cut in the side and 2 parallel rows of holes cut in the bottom. The holes in the bottom are for our next stop, the aquarium department. The small aeriator pumps work well to keep the etchant stirred up. The holes in the bottom have a section of aeriator tubing laced through them, melt holes in the tubing with a piece of wire heated over a candle or some such, then heat the end of the tube so the plastic just starts to melt and squeeze the end shut with a pair of pliers. Always keep the pump above the liquid level or remove the tray when not in use. Now look a little further down the aisle from the pumps and pick up an aquarium heater, a small one is just fine. Make sure it will fit over the lip of the etch tank and will clamp securely.

Drilling circuit boards

Drilling is the most tedious part of the process, patience and a good eye are required. If you get a good etch the centers of the pads will be etched and will help center the drill bit, if not use a sharp awl to mark the centers. If your registration between sides of the board is not perfect or the bond of the copper to the fiberglass is weak it is likely you will peel the traces off the off side of the board. One way to get around these problems is to drill the board half way through, flip the board over and finish drilling. Twice the fun! A good drill press with a spindle stop comes in very handy, see if you can borrow the neighbors. Carbide drill bits are used in the industry as they maintain a good sharp edge. For hobbiest use go to a hardware store or a machine shop supply and buy a couple of no. 63 or 64 size high speed bits, the edge will not last as long but they will bend long before they break.

Putting on the etch resist

Etch resist can take several forms. Photosensitive, manually applied rub-ons, iron-on, and silk screen. Manually applied is OK for small, simple jobs, but is tedious for anything more than a couple of IC's. Photosensitive has been the way to go in the past as it will give good resolution, with the tradeoffs of hard to register on a double sided board, and getting the proper exposure. Silkscreen is the way to go when making a bunch of boards but has a higher initial cost for the equipment and still takes some work to get a double side board registered. Iron-on is probably the best for the average hobbiest as it has the lowest cost and takes no special equipment. There are several types of iron-on film available, many are advertised in magazines. Another idea is to use transparency film made for use in a photocopier. It works almost as well and is available at most office supply stores. The big problem to using this method is finding the right copier, many (particularly Xerox) fuse the toner onto the film at too high a temperature and it will not transfer to the board properly. Use of the film is simplicity, simply copy your artwork onto the film, lay it on your board, and run over it with a laundry iron. Be very careful with the film after copying as the toner will flake off, do not bend or fold the film. Registration of the two sides can be a problem unless you have X-ray vision, an idea for that is to add a couple of extra pads in opposite corners of the board and drill them after ironing on the first side, then simply sight through the film to line up the holes. A small prick punch or a fine awl should be used to mark the holes for the drill.

Etching the board

Good etchant, good agitation, and the proper temperature make for a well etched board. A properly set up tank will etch a board in 5 to 8 minutes when the etchant is fresh, if the etch time rises above 12 to 15 minutes it is time to replace the etchant. Try to keep the temperature around 110 degrees.

Soldering the components

Most hobbiest boards will not have the luxury of having a solder mask screened on so we fake it! If you have a good steady hand and a fine brush you can use paints or laquers to make a fine line between pads to keep the solder from jumping across and making a solder bridge. If you have an unsteady hand and slop some on the pads simply use a small tool to scrape it off the pad. Tinning is a good way to make it easier to make the solder take hold. There are several tinning solutions available but another way is to apply a very thin coat of paste flux, a little solder and then chase it around the board with a soldering iron.

Be careful not to let the layer get too thick, you want just enough to coat the pads. When soldering a double sided board without plated through holes there is the problem of soldering the component side. One way is to cut short pieces of fine wire and place them in the hole with the socket lead. Another way is to forgo the sockets and solder the IC leads directly to the traces, just make sure you have good components. A third way

Continued on page 33

CONNECTING IDE DRIVES

by Tilmann Reh

Special Feature

Intermediate Users

Part 4: IDE Commands

In part II we covered the basics of the IDE interface in terms of history, concept, hardware, and register structure. In part III I started describing the various commands and parameters of IDE drives. This time I will finish that command description and offer some sample driver routines.

I must apologize!

Sorry for the badly formatted Pascal listing printed with part III in the previous issue of TCJ. Bill had to delete all the empty lines in order to compress it to a single page. Now I know that this doesn't make a program more readable or easier to understand, even if it's written in Pascal. We will try to do this better in the future.

Commands Continued...

We already covered most of the manufacturer-independent commands in the previous part. However, there are three commands not explained yet. Let's get started with the command which was already used in the sample program printed with the previous part -- so you'll now know what you really did there (in case you ran that program).

Identify Drive (ECh):

This command reads some detailed parameter information from the IDE drive. Again, it's invalid for the older (external) controllers. It is started by writing the command code into the command register, and then it executes like a Read Sectors command. The DRQ Flag will be set, declaring that data can be read. After having read a complete "sector" (256 words, 512 bytes) of data, the DRQ flag will be reset and the drive will be ready again. The data consists of the following fields:

Word Adr.	Byte Adr.	Type	Content
0	0	word	Configuration/ID word
1	2	word	Number of fixed cylinders
2	4	word	No. of removable cylinders
3	6	word	No. of heads
4	8	word	No. of unformatted bytes per physical track
5	10	word	No. of unformatted bytes per sector
6	12	word	No. of physical sectors per Track
7	14	word	No. of bytes in the inter-sector gaps
8	16	word	No. of bytes in the sync fields
9	18	word	0
10-19	20-39	20 char	Serial number
20	40	word	Controller type
21	42	word	Controller buffer size (in sectors)
22	44	word	No. of ECC bytes on "long" commands
23-26	46-53	8 char	Controller firmware revision
27-46	54-93	40 char	Model number
47	94	word	No. of sectors/interrupt (0 = no support)
48	96	word	Double word transfer flag (1 = capable)
49	98	word	Write protected
50-255	100-511	-	reserved (read as zero values)

Some of these fields have special meanings. The configuration/ID word consists of 16 single-bit flags. However, I don't know for sure if their meaning is really manufacturer-independent. The "controller type" word is encoded as a number representing a particular type.

Configuration/ID word bit flags:

15	Non-magnetic drive
14	Format speed tolerance gap required
13	Track offset option available
12	Data strobe offset option available
11	Rotational frequency tolerance > 0.5%
10	Data transfer rate > 10 MB/s
9	Data transfer rate > 5 MB/s, <= 10 MB/s
8	Data transfer rate <= 5 MB/s
7	Removable disk
6	Non-removable disk
5	Spindle motor can be switched off
4	Head switching time > 15 us
3	Not MFM encoded
2	Soft sectored
1	Hard sectored
0	reserved

Controller type word values:

0	Not specified
1	Single ported, single sector buffer
2	Dual ported, multiple sector buffer
3	= 2, with look-ahead read capabilities

The string-type data fields (character arrays) contain plain text information about the serial number, controller model, and firmware revision of the drive. Each word holds two characters,

which must be displayed with the high-byte character first in order to get readable results.

As far as I know, most IDE drives follow the data field description above. However, there still are many things which are manufacturer-dependent. Fortunately, these details are not critical. To give you some examples: The controller model field of Conner drives contains plain text with the complete drive description like

"Conner Peripherals 40 MB - CP3044".

Seagate's IDE drives offer only a short cryptic ID string, which sometimes doesn't even contain the drive type.

A very interesting difference, even between drives of the same manufacturer, shows up with the "Number of cylinders/heads/sectors" fields. Some drives show their physical values there, independent of the active emulation mode (for example, my CP-3044 does so). Other drives always show the parameters of the active emulation, or those of the default emulation mode. Surprising especially with my drive is that the physical parameters can't be used for drive operation! As a result, the data delivered by this command must be considered carefully. However, it's normally possible to extract useful information by reading the drive's ID information for several different active emulation modes.

Read/Write Sector Buffer (E4h/E8h):

These are the last two common IDE commands. With these commands it's possible to read or write the drive's sector buffer directly. I haven't found any use for these yet, but probably there is (at least was) one. In my opinion, these commands are useless for normal operation.

Block Mode Commands (Read/Write/Set Multiple, C4..C6h):

By the use of these commands, one can access disk data in larger blocks than the physical sector size. Several sectors are grouped together and handled as a block of data. However, many drives don't support this mode. I don't have detailed information regarding the parameters. If a particular drive supports the block mode, the details will surely be printed in its user manual.

Power Commands (E0..E6h, except E4h):

The power commands are not supported by every IDE drive. However, if they are, they are normally compatible. The power commands are commonly used within portable computers (laptops, notebooks, handhelds, or whatever the names are). They allow for automatic or manual changing between normally four operation modes:

Read/Write Mode	(4.2 W)	complete drive circuitry operating
Idle Mode	(2.0 W)	motor running, r/w circuitry turned off while no command is active
Standby Mode	(0.5 W)	motor stopped, r/w circuitry turned off, interface active
Sleep Mode	(n/a)	everything stopped, exit only with reset

The power requirements mentioned in this table are those of my Conner 42-MB drive. While no r/w operation is in progress, the drive normally is in idle mode (also when being reset). Read/write mode is always automatically entered when a r/w command is issued; after completion of that command, the drive enters idle mode again.

When the drive is put into standby mode (manually or automatically, see below), the drive (motor, r/w circuit) is shut down while the host interface remains active. So when a command is issued which requires motor or r/w operation, the appropriate circuitry is automatically switched on again.

Once the sleep mode is entered, there is no way out except for reset by means of hardware or software. This is because even the drive's local processor and interface controller are stopped, so there is no way to communicate with the drive. (However, the task file can still be read.)

As mentioned above, there are six power commands:

Set Standby Mode (E0h), Set Idle Mode (E1h):

The drive will enter the desired mode immediately. There are no parameters required. If the drive already is in that mode, the command will have no effect.

Set Standby (E2h) or Idle (E3h) Mode with Auto-Power-Down:

These commands take a parameter in the sector count register. If that parameter is non-zero, the Auto-Power-Down (APD) feature is enabled (with a zero value, APD is disabled). When one of these commands is issued, the drive immediately enters the desired mode. If APD is enabled, the drive will automatically enter standby mode after being in idle mode without activities for a given period of time. This delay can be specified by means of the parameter for these two commands: the SC register must contain the delay time in counts of 5 seconds. The minimum delay of 60 seconds will be set if the SC register contents is smaller than 12. With a maximum value of 220, the maximum delay is about 18 minutes. These limits again apply to my particular drive; other drives may have other specifications.

Read Power Mode (E5h):

This command reads the actual mode. If the motor is spinning (meaning that the drive is in idle mode), the value FFh will be returned in the SC register. Else (when in standby mode or just spinning up) a zero value will be placed in the SC register.

Set Sleep Mode (E6h):

This command puts the drive into sleep mode immediately.

Every internal activity is terminated and all circuitry switched off.

There are some more power-related commands, having the command codes F8..FDh (except FCh). Their general meaning is similar to the power commands described above (E0..E5h), except that the time delays are specified more exactly (in counts of 0.1 seconds). However, I have not yet seen a drive which supported these commands, and I don't have detailed information about them.

Cache On/Off (EFh):

This is the last command which I will explain here. It is used for enabling or disabling the automatic read-ahead feature (read cache) of the drive. The write precompensation register (WP) is (mis-)used as a parameter register for this command (today, this is the only use of the WP register). If the WP register contains AAh, the feature is enabled; with 55h, it is disabled. Every other value will result in an aborted command error. After reset, the drive defaults to read-ahead feature enabled.

Whew -- this was a lot of stuff! (I hope it was not too hard.) However, now you should know about IDE commands in detail (if you didn't fall asleep while reading). Before we start practical work, here, for the programmers, are the short-form tables that I promised.

Table 1: Task File Registers (as printed in part II)

/CS0	/CS1	A2	A1	A0	Addr.	Read Function	Write Function
0	1	0	0	0	1F0	Data Register	Data Register
0	1	0	0	1	1F1	Error Register	(Write Precomp Reg.)
0	1	0	1	0	1F2	Sector Count	Sector Count
0	1	0	1	1	1F3	Sector Number	Sector Number
0	1	1	0	0	1F4	Cylinder Low	Cylinder Low
0	1	1	0	1	1F5	Cylinder High	Cylinder High
0	1	1	1	0	1F6	SDH Register	SDH Register
0	1	1	1	1	1F7	Status Register	Command Register
1	0	1	1	0	3F6	Alternate Status	Digital Output
1	0	1	1	1	3F7	Drive Address	Not Used

Table 2: Error Register

Bit	Flag	Meaning
7	BBK	Bad block mark detected
6	UNC	Uncorrectable data error
5	-	-
4	IDNF	Sector ID not found
3	-	-
2	ABRT	Command aborted (status error or invalid command)
1	TK0	Track 0 not found during recalibration
0	-	-

Table 3: SDH Register

Bit	Flag	Meaning
7	EXT	Extension Bit. Always 1.
6-5	SIZE	Sector Size. Always 01 (512 byte sectors).
4	DRV	Drive bit. Master/single drive = 0, slave = 1.
3-0	HEAD	Head field. Binary head number 0..15.

Table 4: Status Register, Alternate Status Register

Bit	Flag	Meaning
7	BSY	Drive busy. Task file cannot be accessed.
6	DRDY	Drive ready (up to speed and ready for command).
5	DWF	Drive write fault.
4	DSC	Drive seek complete (actuator on track).
3	DRQ	Data request (ready for data transfer).
2	CORR	Corrected data (bit is set when data has been recovered by use of ECC).
1	IDX	Index. Active once per disk revolution.
0	ERR	Error. See other bits and error register.

Table 5: Digital Output Register

Bit	Flag	Meaning
2	SRST	Software reset (active when set to 1).
1	/IEN	Interrupt enable (active when set to 0).

Table 6: Drive Address Register

Bit	Flag	Meaning
7	-	not driven (for PC floppy compatibility)
6	/WTG	Write gate (active when 0)
5-2	/HSx	Head select 3..0, one's complement of active head
1	/DS1	Drive 1 selected (active when 0)
0	/DS0	Drive 0 selected (active when 0)

Table 7: Commonly needed Commands with Parameters

Code	Command	Parameters
1x	Recalibrate	D
20	Read Sectors with retry	SC,SN,C,D,H
30	Write Sectors with retry	SC,SN,C,D,H
40	Verify Sectors with retry	SC,SN,C,D,H
50	Format Track	C,D,H
7x	Seek	C,D
90	Exec Diagnostics	D
91	Set Drive Parameters	SC,(C),D,H
Ex	Power Commands, see below	
E4	Read Sector Buffer	D
E8	Write Sector Buffer	D
EC	Identify Drive	D
EF	Cache On/Off	D,WP

Power Commands:		
E0	Standby Mode	-
E1	Idle Mode	-
E2	Standby Mode with APD	SC
E3	Idle Mode with APD	SC
E5	Read Power Mode	(SC)
E6	Sleep Mode	-

Table 8: Error Conditions

When an error occurs, the error flag in the status register (ERR) is always set. For the different groups of commands, the following status/error flags are valid then:

Recalibrate	ABRT,TK0,DRDY,DWF,DSC
Read, Verify	BBK,UNC,IDNF,ABRT,DRDY, DWF,DSC,CORR
Read Long, Write, Write Long	BBK,IDNF,ABRT,DRDY,DWF,DSC
Format, Seek	IDNF,ABRT,DRDY,DWF,DSC
Diagnostics, Initialize, R/W Buffer,Identify, Set Cache	ABRT
Invalid command	ABRT

Table 9: Interrupt Conditions

The drive generates an interrupt (if enabled) under the following conditions:

Recalibrate	after successfully reaching track 0
Read	each time DRQ is set
Write	when DRQ is set, from second sector on (only when multiple sectors are written)
Verify	after completion for all sectors
Format Track	after completion
Seek, Initialize, Power Commands (except Sleep)	after command is issued/initiated
Set Sleep Mode	when drive is in sleep mode
Read Buffer, Identify	when data is ready for reading

Now let's come to the example routines for accessing an IDE drive. These examples are given as Turbo-Pascal (3.0) source (based on my IDE test program). They apply to the use of my IDE interface board (described in TCJ #56), so there always are 512 data *bytes* transferred instead of 256 data *words*.

In all examples, named constants are used for accessing the IDE registers at their particular I/O addresses. These named constants must be declared elsewhere. Their names are derived from the related IDE register names and IDE commands.

The examples are programmed in a very modular fashion so that they are easy to understand. For implementation in a system BIOS, for example, most of the subroutines will contain so little code that the complete read/write routines will normally be coded inline. In addition, a real implementation, unlike these examples, will have time-out functions in most loops. If someone is interested in the IDE driver of my CPU280 system BIOS, please contact me (however note, it's Z280 assembly language and commented in German).

1. General access: Wait for drive ready / wait for data request

In Pascal, two small procedures serve this purpose. In assembly language, I use two macros instead, because the subroutine calling overhead would be too much.

```
procedure Wait_Ready;
begin
repeat until port[IDE_CmdStat]<=128;
end;

procedure Wait_DRQ;
begin
repeat until port[IDE_CmdStat] and 8<>0;
end;
```

2. General access: Command issue

```
procedure IDE_Command(Cmd:byte);
begin
Wait_Ready;
port[IDE_CmdStat]:=Cmd;
end;
```

3. General access: Reading/Writing the sector buffer

In the Pascal implementation, the two functions return a Boolean value which is true if there were no errors during r/w of the buffer.

Both routines require the drive to be ready for data transfer!

```
function Read_SecBuf(var Buf:BufType):boolean;
var i : integer;
begin
Wait_DRQ;
i:=port[IDE_Data];  (* specific to my IDE interface board *)
for i:=0 to 511 do Buf[i]:=port[IDE_Data];
Read_SecBuf:=port[IDE_CmdStat] and $89=0;
end;

function Write_SecBuf(var Buf:BufType):boolean;
var i : integer;
begin
Wait_DRQ;
for i:=0 to 511 do port[IDE_Data]:=Buf[i];
Wait_Ready;
Write_SecBuf:=port[IDE_CmdStat] and $89=0;
end;
```

4. General access: First access, initialization

```
procedure HD_Init(Cyls,Heads,Secs:integer);
begin
port[Dig_Out]:=6;
delay(10);          (* Drive Software Reset *)
port[Dig_Out]:=2;
Wait_Ready;
port[IDE_SecCnt]:=Secs;
port[IDE_CylLow]:=lo(Cyls);
port[IDE_CylHigh]:=hi(Cyls);
port[IDE_SDH]:=pred(Heads)+$A0;
IDE_Command(Cmd_Initialize);
end;
```

5. Data access: Single sector read

```
function HD_ReadSector(Cyl,Head,Sec:integer; var
Buf:BufType):boolean;
begin
Wait_Ready;
port[IDE_SecCnt]:=1;
port[IDE_SecNum]:=Sec;
port[IDE_CylLow]:=lo(Cyl);
port[IDE_CylHigh]:=hi(Cyl);
port[IDE_SDH]:=$A0+Head;
IDE_Command(Cmd_ReadSector);
HD_ReadSector:=Read_SecBuf(Buf);
end;
```

7. Data access: Single sector write

```
function HD_WriteSector(Cyl,Head,Sec:integer; var
Buf:BufType);
begin
Wait_Ready;
port[IDE_SecCnt]:=1;
port[IDE_SecNum]:=Sec;
port[IDE_CylLow]:=lo(Cyl);
port[IDE_CylHigh]:=hi(Cyl);
port[IDE_SDH]:=$A0+Head;
IDE_Command(Cmd_WriteSector);
HD_WriteSector:=Write_SecBuf(Buf);
end;
```

Now we have reached the end of the "behind IDE" article series. In another column I will describe my revised IDE interface board for the 8-bit ECB bus in somewhat more detail than in TCJ #56. This will include a TTL equivalent of the GAL contents, for those who are inexperienced in reading a Boolean equation design, or who want to build it up using discrete logic.

For a list of abbreviations, see parts II and III of this article.

Mr. Kaypro. Continued from page28

is to use wire wrap sockets and leave them a little ways above the board so you can reach under with the solder using the iron on the end of the lead under the board and letting the heat travel up the lead. A special thank you to Eric Craig of C&C Machine for doing the plotting and letting me do some of the layout work. Thanks Eric!

Chuck
I hope this looks good enough to be used in one of your articles, another set of plots will be coming soon as I was not happy with the set you received earlier. Ed.

Rest assured Ed, and the new plots will be published in the next issue. The list of parts that will be needed to complete the project is presented last.

The circuit board can either be one you make using the plots printed, or one of the prototype boards listed above. Both 218905 and 207906 are 3/4 hole per pad double sided, with through plated holes, while 462905 is single sided. 462905 will work just fine, but will require more thinking, when it comes to connector layout.

For those of You who elect to manufacture your own printed circuit board, good luck. For those of you who elect not to, hang loose and we'll finish construction and installation in the next issue.

P.S. I owe some of you replies to your letters, keep the faith, and I'll get to it "real soon now". CBS

Foundation	1ea . custom printed circuit board OR SYNTAX # 218905 or 207906 or 462905
ICs	1ea. 7406 1ea. 7445 1ea. 74ls151
Sockets	1ea. 14 pin dip 2 ea. 16 pin dip
Resistors	1ea. 8 section sipp 4.7k per section 5ea. 4.7k .0125 watt
Miscellaneous	1ea. 8 position dip switch 1ea. 34 pin IDH pcb connector 1ea. 34 conductor double in-line header 1ea. 0.01 microfarad disk ceramic capacitor

Regular Feature

68xx/68xxx Support

C & 6800/09 Coding

Small System Support

By Ronald W. Anderson

A "while" has gone by since my first column appeared, and I have received three letters in response. That's not a whole lot to go on, but here goes.

I thought I'd start out with something I found interesting and with which I disagree heartily. The subject is the C programming language, or rather someone's interpretation of how to use it properly.

I have a book, "The Waite Group's C++ Programming" by John Berry, published by Howard Sams & Company, 1988. The subject is the use of pointers rather than array notation. Let me quote:

"When most programmers design a function that uses one or more character string parameters, they usually define them as pointers to a character. For example:

void example(char *s1, char *s2)

represents the header line of such a function. The only other way to declare character string parameters is to declare the parameters as character arrays. However this is inefficient -- even in traditional C -- because the compiler merely converts them into memory address references or, in other words, a pointer. Thus the only real way to handle a character string parameter is by a call by reference. This is equally true for other arrays; however, character strings represent the most common case."

If I am reading what I think I am, they are saying that because the compiler converts array notation to the same code as pointer notation, I should use pointer notation. As a long time user of languages that use arrays, I find:

void example(char string1[], char string2[])

to be less cryptic. If it doesn't matter which I use, and the compiler is going to convert either to the same code, I'll use the clearer notation any day thank you. In these days of super whiz 66 MHz computers, who cares if the compiler has to work just a tiny bit harder?

To be sure, I am exaggerating a little here. I understand how pointer notation relates to array notation, but I prefer to use array notation.

Obviously I'm not a true C purist. I'm sure somewhere must hang a sign that says "Real C Programmers use Pointers", or more probably "Real C Programmers Don't Use Arrays". My point is that the reason given here NOT to use array notation is in my mind a better reason TO use them! The following test program in C contains several ways to print to a terminal, a character string terminated by a null (ASCII code 0), as are all strings in C:

```
// test of string printing.

char string[] = {"This is a test\n" };     // define a string to print
                                           // C adds the NULL automatically
void main()
        {
        char *s;
        int n;

        n=0;
        while (string[n]) putchar(string[n++]);

        n=0;
        while(putchar(string[n++]));

        s = &string[0];                    // s = &string ought to work but gives warning
        while (*s) putchar(*s++);

        s = &string[0];
        while (putchar(*s++));
        }
```

The first and third versions are the array and the pointer versions respectively. The second and fourth are shorter and more cryptic because putchar() returns the character that it has just "put". That is, you can invoke the function, increment the index, and test for a null all in the same statement. (Ain't C wunderful?) The only drawback to this arrangement is that you have putchar(0) or null by the time you detect that you are at the end of the string. Turbo C interprets null as a space which it puts at the beginning of the next line on the monitor, since the null follows the newline (\n).

I'll have to admit there's not much difference with such a simple example. The pointer notation avoids an index variable, but rather "sneakily". The notation *s++ doesn't increment the location pointed at by s, but the value of s itself.

(If I remember correctly, the notation (*s)++ increments the value pointed at by s).

The above is a working program that compiles and runs under Turbo C version 3.0. (I'm using the ANSI C compiler, not the C++ here). As you can see you have to define the text string and then point s at the string. There's not much reason here to prefer pointers over arrays. Now some C whiz please write me and tell me I've been clumsy with my example and that pointers are much better to use "because"...

6800 / 6809 Subjects

I received a plea for help recently from someone who has a couple of old SWTPc 6800 / 6809 systems and is looking for software for them. I'm trying to help him out, though I stopped using 6800 versions of software in about 1979 or 1980. I really don't have any 6800 version software around anymore. Maybe it was shortsighted of me to lose it all, but back then who was thinking of what computers would become now and the possibility that some of these would become collector's items and museum pieces? Can someone out there help me locate any 6800 software information?

I first suggested that my correspondent contact Peter Stark for a copy of 6809 SK*DOS which runs exactly like FLEX. I am going to send him a copy of my PAT text editor along with a hardcopy printout of the manual I had back in the 6809 days. Somehow I've lost the files on disk. I have later manuals for the 68K version and for the PC version, but I couldn't find the 6809 version.

One problem my correspondent had was that companies that are out of business generally move out of their offices. He had sent out over 50 letters to people who had advertised in '68' Micro Journal, and all but three or four were returned as undeliverable. TSC responded that they no longer support FLEX and software that runs under it. I doubt that to be a specific enough statement to be considered a release to public domain. Maybe if someone wrote a letter explaining that there's still a lot of interest out there, and that our "library" would charge and pass along a royalty to be set by TSC? Perhaps they would release much or all of their old FLEX software!

Perhaps the message here is that if you buy an old computer, try to buy all the software with it. It is legal for the registered owner of software to sell (or give away) his copy provided he doesn't also keep a copy for himself. In the case of selling the hardware, it hardly makes sense to keep the software. That probably means you shouldn't buy an old computer from someone who never used it very much, since he won't have much software for it. Aside from that, a well used computer is more likely to be in working condition. Storage in a cold garage or attic for a few years can do horrors for electrolytic capacitors in the power supply not to mention those tin plated connectors that SWTPc used for so long. Those connectors were the eventual demise of my old original computer. I couldn't keep it running long enough to edit a file. It would lock up and I would go and "wiggle the cards" to get it working again for a while. Moisture can ruin a power transformer such that it will run for several hours and then suddenly go up in smoke due to insulation breakdown.

By the way, I suppose you have all heard that integrated circuits run on smoke? Someone deduced that from the fact that if you let the smoke out they don't work anymore!

In the process of poking around my old disks I ran across my assembler working disk with some goodies on it. I'll be presenting some of them here. I indicated earlier that I switched to the 6809 in about 1980. My switch was some of the cause of the intermittent operation of my old SWTPc computer. Apparently (I heard later) those tin plated connectors are good for half a dozen insertion - removal cycles. When I first switched over, I set up my computer to run either the 6800 processor board or the 6809. I added a switch to the I/O address decoding changes I made in the motherboard. Swapping cards soon wore all the tin plating off of the molex pins and I had trouble forever after.

My advice regarding these connectors is that if your computer works reliably don't remove the cards to clean the inside of the case. Leave it alone until it fails. The first try at curing an intermittent board is to plug it onto a different slot in the motherboard. Sometimes it is the alignment of the pins on the motherboard that causes problems. If someone built the computer from a kit, the pins might not align precisely. It was easy to solder with the pins not quite inserted all the way. The holes in the circuit board were oversized and the connector could tilt while soldering it in. To make matters worse, the connectors for the "SS-50" motherboard buss were supplied in ten or twelve pin lengths so they were inserted in the mother board and soldered in place in pieces. Anyway, try a different location first.

The tin plated pins on the motherboard can be cleaned by folding a piece of kraft paper (i.e. from a brown paper grocery bag) in half over the pins and rubbing it back and forth along the row of pins while squeezing slightly. Be careful not to remove too much skin from your knuckles via the adjacent rows of pins in the process. You might argue that this only cleans the connector pins on two sides. The receptacles on the cards only contact the pin on one of those two sides, the side closest to the plug-in board.

I've cleaned the card edge connectors by running a toothpick into each socket position and rubbing a bit. If you do this you can use some solvent. A little Naptha works fine (but don't smoke while using Naptha, it is the main ingredient in lighter fluid). Be careful not to bend the spring contacts in the card edge connectors too far. That is, use thin toothpicks of the flat type and don't overdo the cleaning. All such efforts on my part seemed to help only for a while. I considered soldering the boards permanently to the motherboard, but soon decided that would make it pretty hard to do any future repairs.

While SWTPc used tin plated connectors for all of their computers, GIMIX

used gold plated ones. If you remember those times, there was a real premium on gold for plating connectors, and they were quite a bit more expensive than the tin ones. I remember that a local company who make computers for industrial use had decided that the tin plated connectors (those same Molex connectors used by SWTPc) were just as reliable as the gold ones. I think they later switched to gold as well, though they were rather quiet about it. Actually the gold plated connectors are rather fragile too. They will wear out before very many removals and insertions. They won't however, tarnish or corrode and become poor contacts if they are left alone. If the system works, don't try to improve it by cleaning. Anything you do will result in wear on the contact surfaces. The old saying applies "If it ain't broke, don't fix it".

Enough advice for this time. Are there any of you out there using FLEX on a 6809 system? If so, I have a bunch of utilities I wrote over a long period of time, that I would be interested in sharing with you. There's a KILL utility, same as DELETE except it doesn't ask you twice if you are SURE. It does require a full filename including extension, however.

Then there's a RENUM utility that reads a BASIC program file (.BAS extension) and renumbers it. The utility works for TSC Extended BASIC. I used it with the older BASIC before Extended BASIC existed. XBAS as I called it, has a built-in RENUMBER function that works without leaving the BASIC interpreter, but it only renumbers by tens starting with line ten. My file renumber version is run on a file while out of the BASIC environment. You can specify a starting line number and interval. The old file is renamed to extension .BAK for backup and the new one retains the original file name. I updated it to work with the new features of BASIC, but I don't quite remember whether I ever got the ON ERROR feature working with it. It has one nice feature, that you can scramble lines with a text editor. That is, you can move a subroutine, for example from the end of the program to the beginning (they run faster that way) with no regard for the line numbers being out of sequence. RENUM will renumber them back into ascending sequence.

I vaguely remember some condition where it didn't properly renumber a line. At any rate, if any of you are interested in a few new utilities, send me a note and a 5.25" disk, and tell me whether you want single or double sided / density. I'll include assembler source code for all of the utilities I can find in my archives. If I need more than one disk, I'll supply a second.

I received a letter very recently from a new subscriber to TCJ asking me if my PAT editor would run on a KAYPRO. Unfortunately, 6809 software doesn't run on an 8080, as most of us know. The letter points up the need for some beginner material. We all remember, no matter how far we are along in the computing area, how it was to start. It was confusing to read more advanced material and we were all overwhelmed by the new terms that were thrown at us from every side.

Beginner's Notes

Since this is a little short at this point, I'll begin a section for beginners. If Bill wants to cut it for the month he can do that.

I remember my first computer. As I mentioned a few times ago it was a KIM-1 single board computer. It had about 256 bytes of memory if I remember correctly. The Saturday after I brought it home and fired it up, I had poked around with the monitor a little, and I decided to write a simple program to do something or other, but at the end I couldn't figure out what to do. Should I send the processor a HALT instruction? If I did that, I wouldn't be able to get it to run again. It had not yet occurred to me that the computer must always be running SOME program or it would be "dead". The answer came to me over a Pizza break supper. When the computer is turned on, it comes up running it's ROM monitor program which looks at the keyboard and waits for the user to do something. My program, at it's end simply had to jump back to the monitor program. I soon found the "how" in the excellent manuals that came with the KIM-1 and I was able to load my program by means of Hexadecimal codes, run it, and return to the monitor.

Of course these days we have another level of "standby", the disk operating system. In FLEX or SK*DOS09 all it takes is a simple jump absolute to the address $CD03 known in FLEX as "WARMS" the warm start entry point. Having done that, FLEX is ready to load and run another program from a disk.

Let's talk for a moment about the different levels of programming that are or have been used on computers. Way back in the early days of computers (and repeated in the early days of the microprocessors in the era of that KIM-1) there was machine language programming. That means that the programmer actually calculated the machine codes necessary to make the program run. A program might look like a string of hexadecimal numbers, since that is exactly what it was. The KIM had provision for entering numbers in hex, and so was a bit easier to program than if it had required binary entry. At any rate, the programmer had to calculate the length of a branch for a branch instruction (i.e. the number of machine instructions that had to be skipped to arrive at the desired place in the program) and insert it as such. Negative branches involved counting backwards from Hexadecimal FF etc.

Early programmers got tired of trying to remember that $C6 meant "load accumulator A" with the value immediately following it, for example, and devised a Mnemonic system. LDA # would do it, and was much easier to remember than the actual Hexadecimal code. Incidentally, $C6 isn't necessarily the correct code. Though I programmed a 6809 for a long time, I presently don't remember any of the hex codes for instructions except that $20 for a BRA instruction (branch unconditionally or always) sticks in my mind.

At any rate, someone wrote an Assembler program. With it, you still program at machine level, but you do it with Mnemonic codes that are easier for people

to read. The assembler program reads a "source file" and creates the "object file" machine code.

Later, still more convenient "languages" came along. These were called High Level Languages, which were one more step removed from machine code. One could write a compiler for a language such as Fortran, which would generate code for a 6809, and another for an 8080. Though the 6809 couldn't run the 8080 object code and vice versa, a 6809 could run the code generated from Fortran source code and the 8080 could run the code generated by it's specific Fortran compiler, from the same source code with a few limitations.

High level languages may be written in two different ways. There are interpreters and compilers. Any given language may be written as either. BASIC is usually written as an interpreter. The BASIC interpreter takes your source code just as you have written it, and interprets it line by line as the program runs. That is, it analyzes what you have told it to do and jumps to the appropriate machine routine in the interpreter to execute the action you have defined. Interpreters are slow because, for example, if you have a loop that executes 1000 times, it has to interpret the lines within the loop 1000 times. On the other hand you don't have to go through a compile step before you run your program, so program development is rapid.

A compiler on the other hand reads the source code and generates machine code in the form of an object code file. The machine code generated by the compiler runs much faster than the interpreter. I've seen speed differences of more than ten times.

There is another form that is sort of between the two. A simple compiler of some sort generates a pseudo code (called a P code), which is a simplified intermediate code that is suitable for an interpreter when the program runs. Essentially it does a partial compile and then interprets the code produced by the compiler. These are generally intermediate in speed between an interpreter and a compiler. These days P-code compilers are rather rare.

C has gotten to be one of the most "portable" languages. It's definition if you include the "standard library" includes the syntax for file handling and a lot of other useful things left dangling in Pascal for example.

Pascal is a fine language, (in fact my first preference) but file handling was left to the implementors, so opening a file in one version of a Pascal compiler might be quite different than doing the same thing in another. On the other hand Pascal is a simpler language than C, and it may be compiled VERY fast. If you've ever run Turbo Pascal and Turbo C, you know that the Pascal compiler is like lightning compared to the C compiler. The reason is simply that the C compiler has to do a great deal more to get from the source code to the object.

For you beginners, though Gertrude Stein said "a rose is a rose is a rose", unfortunately a processor is not a processor is not a processor. That is, a 6809 won't run 8080 code, or even 68000 code for that matter. They can all be programmed using a compiler such as a C language one, and the C "source code" may well be identical for all of them, but the compiler can't be identical. Each has it's own arrangement of hardware and it requires different machine codes to run that hardware. Each has a different compiler to produce the different machine code from the same source code instructions.

Just to make matters a bit more complicated, there are things called cross compilers that can run on one machine but generate code that runs on another. Early along, there was a cross assembler that ran on a 6809 under FLEX and generated code for a 68000.

Well, perhaps this is enough for this time. I hope these feeble efforts might have given you a glimmer of understanding. We'll continue little sections specifically for you in future columns.

Special Feature

Intermediate Users

Part 3: Corrections

MULTIPROCESSING FOR THE IMPOVERISHED

by Brad Rodriguez

Part 3: Mid-Course Corrections

"[I]nnovation really happens by blundering through to success on the back of one's mistakes."

- Gifford Pinchot III, *Intrapreneuring*

BRAD'S FIRST MISTAKE

TCJ reader Andrew Houghton was quick to spot a potentially disastrous goof in my 6809 Request/Grant logic. The 6809 is a *dynamic* processor, which means its internal registers are like dynamic RAM -- they need to be constantly refreshed. This refresh occurs whenever the internal clock is running. But, unlike the Z80, the 6809 *stops* its internal clock when the MRDY (a.k.a. WAIT\) line is held low. The result: hold MRDY low for more than 16 usec, and the CPU registers become garbage! (This interesting fact is contained in a teeny tiny footnote in the 6809 data sheet.)

You can see that if eight CPUs are contending randomly for the bus, the odds of one being "shut out" for 16 usec or more are high. The solution has two parts: a) limit the amount of time any one CPU can grab the bus, and b) force the processors to use the bus in strict rotation. For example, if there are eight CPUs, and each holds the bus for no longer than 2 usec, no CPU will ever have to wait more than 16 usec for its turn at the bus.

Part (b) is easy -- it affects only the design of the bus arbiter. Part (a) is more tricky. Recall from the last article that the Read-Modify-Write logic works by stretching every bus request for two extra E cycles. Thus the bus is "held" until the Write portion of the RMW begins. A side-effect is that the bus is held for two E cycles *after* the write...a total of 5 E cycles for the complete RMW operation. If the next instruction also uses shared memory -- or worse, is executing from shared memory -- this stretched bus request may be continued by the next instruction, and the bus may be held indefinitely.

What we *really* want is a circuit that stretches the bus request only until the end of the Write. Since we don't know which memory references are RMW cycles, we'll settle for a circuit that stretches a bus request for *only 3 E cycles*, even if a second access (the Write) occurs. In other words, we don't want to "retrigger" the delay if another access occurs during the stretch period. I couldn't think of such a circuit offhand, so I dusted off my old college text and went through the full ordeal of formal state machine design (see sidebar). Suffice it to say that It Can Be Done, and the new delay circuit requires only replacing the D flip-flops with JK flip-flops, and adding one inverter (see schematic).

On a 1 MHz 6809, 3 E cycles require 3 usec, so up to five CPUs can share the bus in rotation without violating the 16 usec limit for any CPU. Using a 1.5 MHz 68A09 or a 2 MHz 68B09 allows eight CPUs.

BRAD'S SECOND MISTAKE

My second mistake was Thinking Small. I had decided that the simplest possible CPU board would be best for TCJ. But everyone who sends me mail seems to want the improved ScroungeMaster II! So, at the risk of abusing the patience of TCJ's readers (and editor), I'm going to complete this series of articles with the "SM II" design. If you've already bought parts for the 6809 uniprocessor, you'll be happy to know that only the 2681 DUART, 75176 transceiver, and a 74LS139 are discarded. I think you'll find the improvements worth the cost.

To keep our Esteemed Editor happy, I'll publish and describe the six pages of schematics in three installments. In this article, I will cover the (slightly revised) CPU, some new memory mapping logic, and address decoding. Following that will be memory, parallel I/O, an interface to IBM PC bus peripheral cards, and the bus arbiter. Third will be the serial I/O and some other frills.

THE REVISED CPU AND RMW LOGIC

Figure 1 shows the revised CPU circuit. U1 is the 6809, unchanged from the uniprocessor design.

OFFBD\ comes from the address decoding logic, and is pulled low during an access to the external bus. When this happens, U7A and U14 generate a "stretch" signal for two clock cycles (see sidebar). When either OFFBD\ is low or STRETCH\ is low, the REQ ("request") signal is output high by U6A. This signal is sent to the bus arbiter, to indicate that this CPU desires to obtain (or hold) the bus. With JP3 in the "3CLK" position (as shown), the bus will be grabbed for three E cycles, and Read-Modify-Write accesses will be indivisible. If JP3 is

moved to the "1CLK" position, the stretch circuit is disabled, and the bus will only be grabbed for one E cycle at a time. (For experimentation with other mutual exclusion schemes.)

If this CPU does not "own" the bus, GRANT\ will be high. The combination of GRANT\ high and REQ high will output a low from U6B, clearing all the flip-flops of U15. This will cause all the Q\ outputs to be high, one of which is jumpered (via JP6) to U27A. This forces U27A to output a low, pulling MRDY low and halting the CPU. This function can be disabled (for experimentation) by jumpering JP6 to the bottom (ground) position.

GRANT\ is pulled low when the CPU acquires the bus. This releases the CLR input of U15, causing its outputs Q1\ through Q4\ to go low in one, two, three, and four clock cycles, respectively. JP6 selects how many clock cycles will be added before the CPU completes the memory access.

When both GRANT\ and OFFBD\ are low (and thus OFFBD is high), U7C and U6C pull the DRIVENBL\ signal low. This enables the three-state drivers on the external bus. Note that neither GRANT\ nor OFFBD alone is sufficient: if GRANT\ enabled the drivers, and the bus is still granted to this CPU when we read the on-board EPROM (for example), both the bus transceiver and the EPROM would attempt to drive the CPU's data lines simultaneously, causing a "conflict." If OFFBD\ enabled the drivers, this CPU could attempt an off-board access and activate its bus drivers while the bus was granted to another CPU, causing a conflict on the external bus.

The new external bus accommodates IBM PC peripheral cards, which may, for their own purposes, pull IORDY low to stretch the memory access. Obviously, this signal should be observed only by the CPU which is on the bus, so U27B gates IORDY with DRIVENBL\: *both* must be low to assert XWAIT high ("external wait"). XWAIT is logically ORed with the "internal wait" signal by U27A: when *either* wait is requested, the CPU's MRDY input is pulled low.

IBM PC peripheral cards may also assert an active-high interrupt. One of the five interrupt signals on the PC bus is selected by a jumper (not shown here) as the signal XIRQ. XIRQ is inverted to active-low by U27C. It may then be jumper-routed to either the CPU's IRQ\ input, or IRQ4\, an auxiliary interrupt input of an I/O chip. U27 is *not* open-collector, so *if the external interrupt is connected to IRQ\, no other interrupt source should also be connected to IRQ\.* JP4 can be removed entirely if external interrupts aren't used.

JP5 can connect the REQ signal to a programmed output pin, to allow experimentation with software request/grant schemes rather than hardware. For standalone use (not plugged into a bus), U6, U14, U15, and U27 may be removed. (MRDY is then pulled up to +5 by R9.) Terminal block J1 is a power connector for standalone use.

MEMORY MAPPING

The ScroungeMaster II expands the 6809 address space to 1 MB, and allows both memory and I/O access to IBM PC peripherals. Figure 2 shows the address generation and decoding logic. The 6809's 64K memory space is divided into nine regions:

address

0000-0FFF	"page 0", 4K	)
1000-1FFF	"page 1", 4K	)
2000-2FFF	"page 2", 4K	) each of these 4K "pages"
3000-3FFF	"page 3", 4K	) can be "mapped"
4000-4FFF	"page 4", 4K	) to any 4K region in a
5000-5FFF	"page 5", 4K	) 1 MB address space
6000-6FFF	"page 6", 4K	)
7000-7FFF	"page 7", 4K	)
8000-FFFF	fixed EPROM region, 32K	

The addresses from 8000-FFFF are "unmapped" -- the raw physical address lines from the 6809 CPU are connected to the EPROM, so that whenever the program reads 8000-FFFF, it *always* gets the EPROM. The addresses from 0000-7FFF are "mapped" -- they are passed through a circuit which converts the 16-bit address from the CPU (actually 15-bit, since we know A15 is zero) to a 20-bit address.

This is done by U2 and U3, a pair of 74S189 16 word x 4-bit high speed RAMs. Assume for the moment that the TASK line is pulled high (jumper JP2 removed). When the CPU outputs an address, bitsA12-A14 select one of eight locations in the fast RAM. Eight bits are output from that location (four from U2, and four from U3). These bits are the high eight bits of the "mapped address," MA12-MA19. Every chip except the EPROM uses these as if they were the "real" address bits output by the CPU. (Note that the low twelve address bits A0-A11 are unchanged, and are used by all chips.)

Suppose that these eight locations contain hex 12, 34, 56, 78, 9A, BC, DE, and F0, respectively. Then, when the program

reads	0000-0FFF,	it actually gets	12000-12FFF;
"	1000-1FFF,	" " "	34000-34FFF;
"	2000-2FFF,	" " "	56000-56FFF;
"	3000-3FFF,	" " "	78000-78FFF;
"	4000-4FFF,	" " "	9A000-9AFFF;
"	5000-5FFF,	" " "	BC000-BCFFF;
"	6000-6FFF,	" " "	DE000-DEFFF;
"	7000-7FFF,	" " "	F0000-F0FFF.

The 6809 can only access 32K of "mapped" memory at any one time (in eight 4K chunks), but by rewriting the mapping RAM (U2 and U3), all of a 1 MB address space can eventually

be reached. (Note that U2 and U3 completely ignore A15, so they remain active even during EPROM accesses.)

The mapping RAM is written by a Write to an address in the EPROM space (8000-FFFF, A15 high). U7D and U8B detect the combination of A15 high, R/W\ low (write), and E high (data strobe), and produce the WRMAP\ signal. *The location which is written depends upon A12-A14.* Thus, to write the first mapping RAM location, write a byte to any address in 8000-8FFF.

to program the map for					0000-0FFF,	write location	8xxx.
"	"	"	"	"	1000-1FFF,	" "	9xxx.
"	"	"	"	"	2000-2FFF,	" "	Axxx.
"	"	"	"	"	3000-3FFF,	" "	Bxxx.
"	"	"	"	"	4000-4FFF,	" "	Cxxx.
"	"	"	"	"	5000-5FFF,	" "	Dxxx.
"	"	"	"	"	6000-6FFF,	" "	Exxx.
"	"	"	"	"	7000-7FFF,	" "	Fxxx.

Since the 74S189 has logically inverted outputs, you actually have to write the *complement* of the desired data to the mapping RAM. So, for the example above, you would write hex ED, CB, A9, 87, 65, 43, 21, and 0F, respectively.

Finally, the TASK input is connected to a programmable output bit on one of the I/O chips. When a '1' is output, the *last* eight registers in the 74S189s will be used for mapping. When a '0' is output, the *first* eight registers are used. This means that two independent maps can be stored in the mapping RAM, and switched by changing one bit. This could be used to support two tasks, or perhaps "main task" and "interrupt" memory maps. Note that you have to select map 0 before writing map 0, and likewise for map 1.

ADDRESS DECODING

U8C generates the read signal for the EPROM when A15 is high, R/W\ is high (read), and E is high (data strobe). All other chip selects and data strobes are generated from the mapped address. The 1 MB mapped address space is divided as follows:

00000-DFBFF:	external bus, memory references (895 KB)
DFC00-DFFFF:	external bus, I/O references (1 KB)
E0000-FFBFF:	on-board RAM (127 KB)
FFC00-FFFFF:	on-board I/O (1 KB)

The 8086 used in the IBM PC distinguishes between I/O and memory references. Since the 6809 has no such provision, some segment of its memory address space must be assigned to simulate the I/O signals. I have chosen to generate the I/O Read and Write signals when any location in the last 1K of the external address space is referenced (the IBM PC I/O space is 1K long). Memory Read and Write signals are generated for the other 895K.

This complex address map requires a two-level decoding scheme. Normally I prefer to minimize the number of levels of decoding logic, since this is in a critical timing path. (I particularly despise cascades of 74LS138s.) In this case I'm willing to incur the timing penalty of an extra level of NAND gates.

U4 generates a signal, IOZONE\, which is low when either the external I/O space *or* the on-board I/O space is accessed. This is easier to see if you look at the binary addresses for these two regions:

external I/O:	1101 1111 11xx xxxx xxxx
on-board I/O:	1111 1111 11xx xxxx xxxx

If MA17 is ignored, the logical AND of the remaining high address bits will correctly identify both regions. U8A generates a signal, ONBOARD\, which is low when either the on-board RAM or on-board I/O is accessed -- that is, whenever the top three mapped address bits are '1'.

The four possible combinations of IOZONE\ and ONBOARD\ identify the four memory regions: LCLIO (local I/O), RAM (local RAM), MEM (PC bus memory), and IO (PC bus I/O). These are combined in U5 with the R/W\ selection signal and the data strobe E, to produce four Read strobes and four Write strobes. You may wish to verify that all of the combinations are correctly decoded. Note that, by the nature of the 74LS138, only one strobe can be asserted at any time. Since the mapped address is always generated, even during EPROM accesses, these strobes must be blocked when A15 is high (EPROM space). This is done by feeding A15 into one of U5's active-low enable inputs.

The bus request logic needs a simpler signal. OFFBD\ is asserted (low) whenever an address in the external bus space is detected. We can't just use the logical inverse of ONBOARD\, since ONBOARD\ is unpredictable when accessing the EPROM. OFFBD\ is asserted, by U7E and U6D, when ONBOARD\ is high *and* A15 is low.

When IOZONE\ is low *and* MA17 is high, the address is in the on-board I/O space. When this occurs -- and, again, when A15 is low -- decoder U24 is enabled. This generates eight on-board I/O chip select signals, depending on the values of A7-A9. In effect, this divides the 1 KB on-board I/O space into eight 128-byte regions. Each of these regions will typically be occupied by one I/O chip. (You might think that the inclusion of MA17 is redundant, since LCLIOWR\ and LCLIORD\ are only gen-

erated when MA17 is high. But some I/O chips, such as the 6522, use R/W\ and E instead of the RD and WR signals, so the chip selects *must* be qualified by MA17 too.)

It's my habit to put all of the bypass capacitors (C1-C30) in one place on the schematic. This was the page where I had some extra room.

PC BOARDS

I'm pleased to announce that -- thanks to the interest of many TCJ readers -- I am proceeding with the layout and production of a PC board for the ScroungeMaster II. Estimated cost is US$20 each; the final cost won't be known until the layout is complete. If you would like boards from the first (and maybe only) production run, please contact me at b.rodriguez2@genie.geis.com on Internet, B.RODRIGUEZ2 on GEnie, or at Box 77, McMaster University, 1280 Main Street West, Hamilton, Ontario, L8S 1C0, Canada.

REFERENCES

[HIL74] Hill, Frederick J. and Peterson, Gerald R., *Introduction to Switching Theory and Logical Design*, Second Edition, John Wiley & Sons, New York (1974), ISBN 0-471-39882-9. A classic, but probably superseded by many newer textbooks.

STATE MACHINES

A digital circuit that contains flip-flops (or some other kind of storage) is usually called a *sequential circuit*. At any particular time, there will be some pattern of 1s and 0s in its flip-flops (or other storage); this is called the *state* of the circuit. A *state machine* is a sequential circuit that can change from one state to another, depending upon its inputs. (State machines can also be simulated in software, but that's not important here.)

State machines are designed by first identifying all the states which are required, and all of the possible transitions between the states. The decision of which pattern of 1s and 0s corresponds to which state is frequently left for last. This is because the logic can often be simplified, if a "non-obvious" pattern of 1s and 0s is used.

For example, here are the states of the new RMW stretch circuit, in tabular form. There are only three states, numbered 1-3, and one input (OFFBD\). The rightmost columns indicate the "next state", that is, the state to which the circuit will change on the next clock pulse, for any given input.

#	state	Next state if OFFBD\ low	Next state if OFFBD\ high
1	Idle	2	1
2	1st delay cycle	3	3
3	2nd delay cycle	1	1

The circuit will remain in the Idle state until OFFBD\ is asserted (low). Then the circuit will move through the two "delaying" states, *regardless of the level of the OFFBD\ input*. This is exactly the behavior we want -- the Write cycle does *not* retrigger the delay.

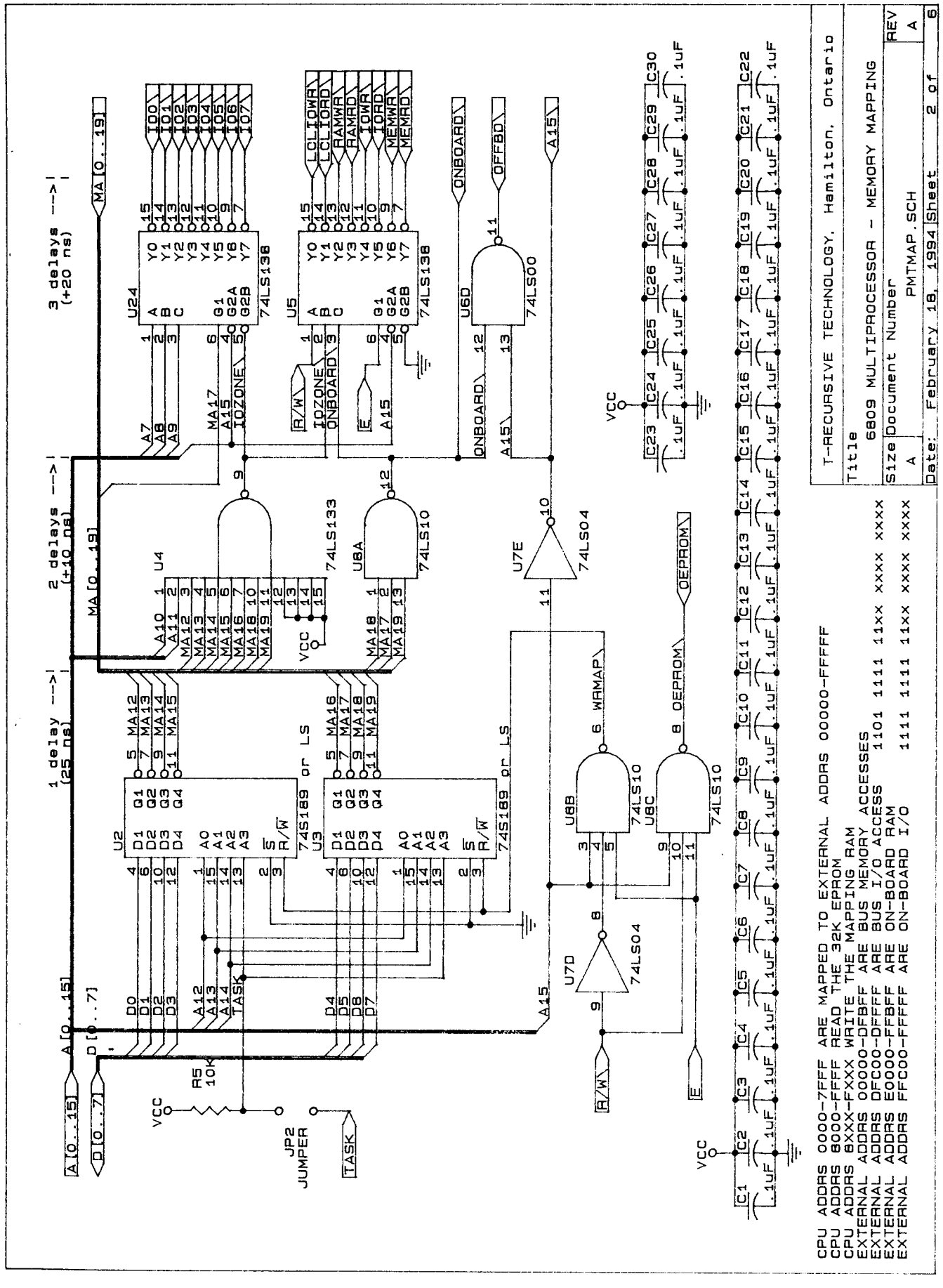
At least two bits (flip-flops) will be needed to represent three states. In this case, JK flip-flops require the least extra logic. The operation of a JK flip-flop when the clock pulse occurs is as follows:

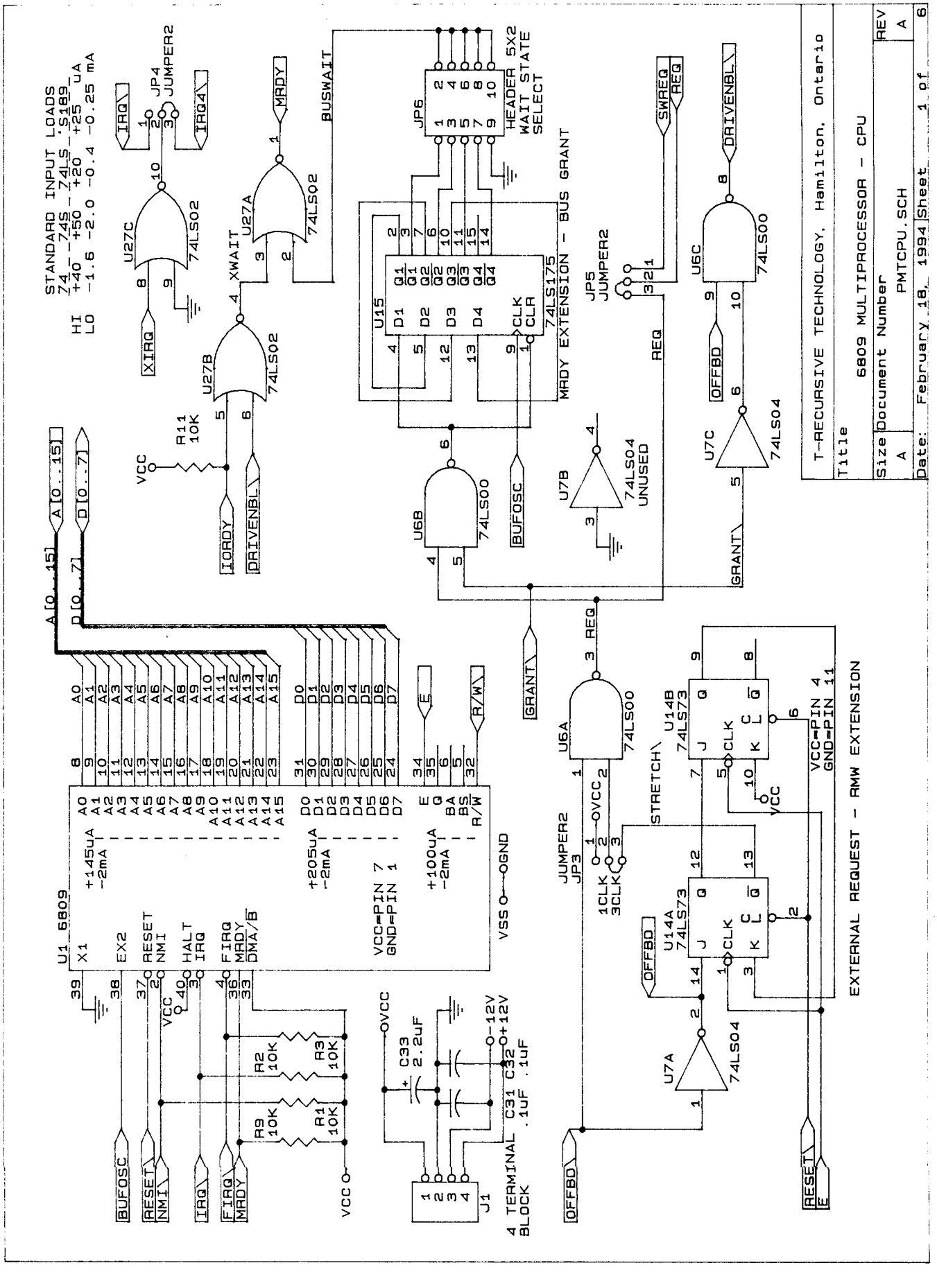
inputs		outputs	
J	K	Q	Q\
0	0	hold previous outputs	
0	1	0	1
1	0	1	0
1	1	toggle previous outputs	

In this case, I was able to have one of the flip-flop outputs directly represent the "stretch" signal (i.e., active during states 2 and 3), and also to represent Idle as 00, so that RESET\ could set the circuit to the Idle state. I encourage you to take the schematic and the JK table given above, and to verify that the circuit does step through all the states and produce the correct "stretch" output for any input combination. Start at state 00 (Idle), and note that the 74LS73 transition occurs at the falling edge of E. Observe also that the circuit cannot remain "stuck" in the unused fourth state, if it ever accidentally gets there.

Space prohibits a full description of the design techniques I used. This circuit is small enough that the simplest manual techniques were adequate: Karnaugh maps, flip-flop transition lists, and an exhaustive trial of state assignments [HIL74]. A much more powerful tool is the Quine-McCluskey method, which can be done manually or by computer. Newer and better methods have doubtless been devised; consult any good textbook on digital logic design.

Curmudgeonly observations: Alas, state machine minimization is going the way of long division: before long, no one will be taught how it's done. They'll just punch "divide" on their calculators, and click "minimize" on their PLD design programs. I call it "Engineering without Understanding." Harrumph.


CPU ADDRS 0000-7FFF ARE MAPPED TO EXTERNAL ADDRS 00000-FFFFF
CPU ADDRS 8000-FFFF READ THE 32K EPROM
CPU ADDRS 8XXX-FXXX WRITE THE MAPPING RAM
EXTERNAL ADDRS 00000-DFBFF ARE BUS MEMORY ACCESSES
EXTERNAL ADDRS DFC00-DFFFF ARE BUS I/O ACCESS 1101 1111 11xx xxxx xxxx
EXTERNAL ADDRS E0000-FFBFF ARE ON-BOARD RAM
EXTERNAL ADDRS FFC00-FFFFF ARE ON-BOARD I/O 1111 1111 11xx xxxx xxxx
T-RECURSIVE TECHNOLOGY, Hamilton, Ontario
Title
6809 MULTIPROCESSOR - MEMORY MAPPING
Size A
Document Number PMTMAP.SCH
REV A
Date: February 18, 1994
Sheet 2 of 6
1 delay --> (25 ns)
2 delays --> (+10 ns)
3 delays --> (+20 ns)
A[0..15]
D[0..7]
MA[0..19]
74S189 or LS
74LS138
74LS133
74LS10
74LS04
74LS00
R5 10K
JP2 JUMPER
TASK
R/W
IOZONE
ONBOARD
OFFBD
OEPROM
WRMAP
RAMWR
RAMRD
IOWR
IORD
MEMWR
MEMRD
LCLIOWR
LCLIORD
VCC


STANDARD INPUT LOADS
74 74S 74LS 'S189
HI +40 +50 +20 +25 uA
LO -1.6 -2.0 -0.4 -0.25 mA
T-RECURSIVE TECHNOLOGY, Hamilton, Ontario
Title 6809 MULTIPROCESSOR - CPU
Size A Document Number PMTCPU.SCH REV A
Date: February 18, 1994 Sheet 1 of 6
EXTERNAL REQUEST - RMW EXTENSION
MRDY EXTENSION - BUS GRANT
HEADER 5X2 WAIT STATE SELECT
4 TERMINAL BLOCK
U1 6809
74LS02
74LS00
74LS04
74LS175
74LS73
JUMPER2
R11 10K
R1 10K
R2 10K
R3 10K
R9 10K
C31 .1uF
C32 .1uF
C33 2.2uF
+12V
-12V
VCC
VSS
GND
BUFOSC
RESET
IRQ
FIRQ
MRDY
DMA/B
HALT
XIRQ
IRQ4
IORDY
DRIVENBL
BUSWAIT
XWAIT
GRANT
REQ
SWREQ
OFFBD
STRETCH
1CLK
3CLK
UNUSED

Special Feature

Intermediate Users

Battery Backup

Little Circuits

by Dave Baldwin

Now there is a way to send your comments and suggestions to me about Little Circuits. DIBs BBS went online on Feb. 2, 1994 running Wildcat software operating from 1200 to 14.4Kb. There is a *TCJ* conference where you can leave messages. I've created a special logon that allows you to get directly to the *TCJ* conference and file area. Call (916) 722-5799 and use the following logon:

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

The *TCJ* download area has a ProComm script for logging on. There also some phone number lists that might be interesting. If you also want access to other areas, log on with your own name and password.

BATTERY BACKUP

When you turn your computer off (or the local power company turns it off for you), you want your real-time clock to keep ticking and sometimes you need to keep data in memory for the next time the computer comes on. All the real-time clock chips I know of require a battery to keep them running when the power is off. For non-volatile memory, you can use EEproms, Flash eproms, or battery backed static rams. EEproms and Flash eproms require special code to write to them and can't be used as normal read/write ram whereas battery backed static rams operate as regular rams that happen to retain their data when the power goes away.

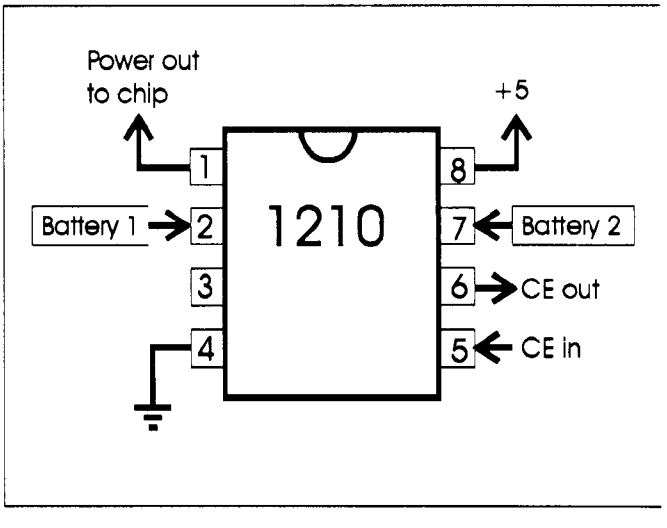
I'm covering battery backup circuits this time because they have a lot in common with the reset circuits I discussed in the last Little Circuits article. Just like the reset circuits, battery backup circuits have to operate through the power up and down cycles and brown-out conditions without losing data and/or timekeeping. Most of the battery backup circuits include the same supply voltage sensing circuitry that reset circuits use.

For new designs, you don't have to design anything. Real-time clocks and static rams are now available with lithium batteries built into the package so you can just plug them in to a dip socket. Chips are available that just plug into existing sockets for MC146818 real-time clocks and 64kb and 256kb static rams. If you are designing a new system, you should note the batteries are not replaceable and are rated to last 10 years. However, they are very convenient. For a real-time clock, I prefer the Harris (Intersil) ICM7170 because it has circuits for for an external lithium battery built in.

Older systems, especially those with the National Semiconductor MM58167A real-time clock, often have flaky battery backup circuits. The MM58167A is probably the most misused device as far as battery backup is concerned. I've seen a number of circuits that almost work. All the information needed is in the design guide for the chip. I spent a lot of time trying to get one to work on a Z80 single board before a friend gave me a copy of the design guide.

CIRCUIT CONSIDERATIONS

There are three considerations for battery backup circuits. First is the Vcc/Battery switch circuit. Second is glitch prevention during power supply changes, especially on the 'Write' input. The third item is power-fail detection. Devices like the 58167A have a Power down input that needs to be switched at the proper time.



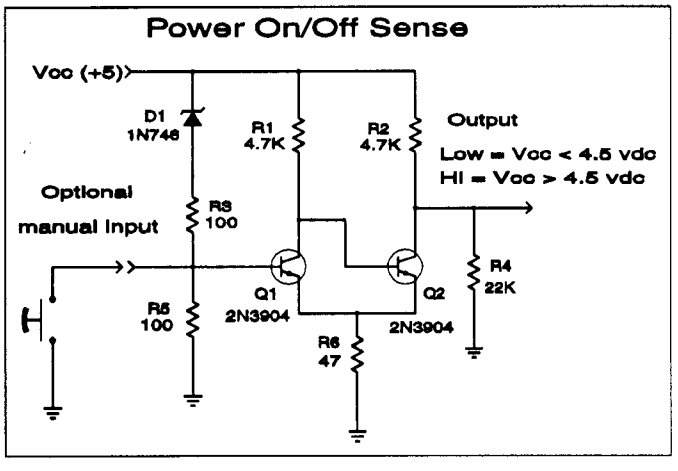
There are several IC's (non-volatile controllers) designed to provide the battery switch and the glitch prevention. One is the DS1210 from Dallas Semiconductor, also available as the

MXD1210 from Maxim. The 1210 contains battery switches for two lithium batteries and a chip-enable gating circuit. The chip connects the power output pin to the power input pin that has the highest voltage, whether it's the regular 5 volt supply or one of the two batteries. The 1210 also has a voltage level detector built in that disables the Chip-enable output when the supply voltage drops too low. The level detector circuit is just like the reset circuits in the last Little Circuits article. This time it is used to gate a chip-enable signal instead of providing a reset signal.

There is an interesting note in the data sheet for the ICM7170. It says that you need to put a 2K resistor in series with the lithium battery for UL approval. This is to prevent the battery from being shorted if the IC fails.

Dallas has other non-volatile controllers with more chip-enable outputs. Maxim includes the battery switch circuit in several of their power-supply monitor IC's. The MAX693 reset controller mentioned in the last article also includes a battery switch circuit.

The 1210 is all you need to convert a low-power cmos static to battery backup and it will provide two of the three required functions for the 58167A. The two transistor circuit in the next figure will provide the power-down signal. It can also be used for a power up/down reset circuit.



Q1 and Q2 are connected as a Schmitt trigger. D1, R3, and R5 set the voltage at which the output changes. In my test circuit, it was about 4.5V. Low voltage Zener diodes like the 1N746 have a very rounded knee voltage curve. The 1N746 is supposed to be a 3.3V diode, but the one I used in my test circuit actually measured 3.0V. You may have to specially select the diode or adjust the values of R3 and R5 if you use this circuit. You could also use two red LED's or 1 green LED and adjust R3 and R5. The 22K resistor on the output is to make sure the output signal can't float up when the power is off.

If you're not concerned about glitch protection, the MAX703/4 contain the battery switch circuit and reset and power fail outputs you could connect to the 58167A power-down input.

The most complete chip I've found is the Dallas DS1234. It gates both the chip-enable and write signals, provides a power fail output, and does not turn on the battery until you program it to. You can use this chip in a unit that has to sit on the shelf for a while before it is used and not use up the lithium battery.

RECHARGEABLE BATTERIES

All of the previous circuits were for lithium batteries which are low current devices. Sometimes you would prefer to use rechargeable, re-usable batteries that provide higher load currents. For printed circuit board mounting, this usually means nicads and a charging circuit.

Nicad capacity (C) is rated in milliampere-hours, a somewhat misleading term. Five hours appears to be the standard. A 'AAA' cell is typically rated at 500mAh (which is 'C') which means that you can discharge it at 100 mA for five hours before it's considered dead. Then the charts in the battery specs show a curve that ends at six hours. Maybe it's just conservative rating. Anyway, the recommended charging currents are based on the rated capacity.

I've heard all kinds of mystical theories on ways to extend the life of nicads. I can't recommend any of them. Some of them, such as completely discharging the battery by shorting it, are dangerous. Batteries can explode because excessive high discharge currents heat up the internal resistance of the battery and cause internal gasses to expand. Besides that, even rechargeable batteries wear out after a while.

Nicad charging circuits range from simple half-wave rectifiers with a series resistor to circuits that sense both temperature and battery voltage and adjust the charging current automatically. The standard charging rate is 0.1C (C = capacity) or 50mA for a 500mAh battery. After the battery is fully charged, trickle-charging at .02C to .05C is recommended to keep the battery fully charged. Nicads will self-discharge and go dead without the trickle-charge.

Nicads are also temperature sensitive. They have 100% of their rated capacity about 20°C or just below room temperature. Below and above this temperature, the capacity and ability to be charged drops. Self-discharge increases with temperature also, so nicads will stay charged longer at cooler temperatures.

I don't yet have a charging circuit I can show you that works from normal computer power supply voltages (+5 and +12). The one I have been using requires about 7 volts. Maybe I can slip one in next time.

REFERENCES

Dallas Semiconductor 1992-1993 Product Data Book.
Maxim New Releases Data Book, Volume III, 1994.
Harris Data Aquisition Data Book, 1991.
PowerSonic Nicad battery data sheet.

Regular Feature

Classic Support

Group Reviews

SUPPORT GROUPS FOR THE CLASSICS

By JW Weaver

GROUP REVIEW

We received a portion of the Windsor Bulletin Board User's Group newsletter, courteys of Emmanuel Roche. It seems Emmanuel contributed in October of last year a complete set of the now defunct Piconet Library. The contents were mainly older programs, some of which I know were later updated and released on the SIGM disks.

The newsletter also indicated that Shane Badham had contributed his complete PD library. This library contains programs by Digitial Research at about the time PCDOS was hitting the market. Since these may be some missing and otherwise not avaialble programs, the group will be checking them out and hopefully have them available for release some time this year. The list of programs available are in volume 121 and can be obtained from Rodney Hannis, at £3.00 each.

Volumes 117 to 120 contain the Z-System programs and have been produced by Mark Minting. The newsletters comments are: "These are designed for Z-S itself but many will run satisfactorily if you are equipped with at least ZCPR33 and might even run on vanilla CP/M. In the MSDOS world of 1993 where most programs appear to be sloppily written and to occupy 100K+ it is very pleasing to find so many very fast comprehensive programs with lengths of only a few K. If you have still resisted the temptation to experiment with Z-System may I tell you that prices in the states have fallen and Mark can now let you have NZCOM or Z3PLUS complete with several hundred K of add-ons and a printed manual for just £24.00."

Also listed are Volumes for MSDOS support, and Vol. 34 has MYZ80 latest release. To properly use MYZ80 with Z-System you will need NZOM or Z3PLUS. MYZ80 does run CP/M as shipped. There was also a request for letters to Novel the new owners of DRI to get GSX-80 into public domain like they did with GSX-86 (CP/M80 and CP/M86 respectively).

TCJ Staff Contacts

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

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

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

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

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

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

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

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

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

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

Tilmann Reh, Germany, E-mail: tilmann.reh@hrz.uni-siegen.d400.de. Has complete MS-DOS disk emulation program for CP/M+, contact Jay Sage.

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

USER GROUPS

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

Sacramento Microcomputer Users Group, PO Box 161513, Sacramento, CA 95816-1513, BBS: (916)372-3646. Publishes newsletter, $15.00 membership, normal meeting is first Thursday at SMUD 6201 S st., Sacramento CA.

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

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

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

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

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

Adam International Media, Adam's House, Route 2, Box 2756, 1829-1 County Rd. 130, Pearland TX 77581-9503, (713)482-5040. Contact Terry R. Fowler for information.

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

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

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

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

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

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

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

OTHER PUBLICATIONS

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

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

Z-100 LifeLine, Steven W. Vagts, 2215 American Drive, Roseville, CA 95747, (916) 773-4822. Publication for Z-100 (a S-100 machine).

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

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

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

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

Historically Brewed, A publication of the Historical Computer Society. Bimonthly at $18 a year. HCS, 10928 Ted Williams PL., El Paso, TX 79934. Editor David Greelish. Computer History and more.

Other Support Businesses

Sydex, PO Box 5700, Eugene OR 97405, (503)683-6033. Sells several CP/M programs for use with PC Clones ('22Disk' format/copies CP/M disks using PC files system).

Elliam Associates, PO Box 2664, Atascadero CA 93423, (805)466-8440. Sells CP/M user group disks and Amstrad PCW products. See ad inside back cover.

Davidge Corp. 94 Commerce Dr. PO Box 1869, Buellton CA 93427, (805)688-9598. Z80 support of Davidge and Ampro Z80 Little Board.

Star Technology, 900 Road 170, Carbondale CO, 81623. Epson QX-10 support and repairs. New units also avialble.

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

Peripheral Technology, 1480 Terrell Mill Rd. #870, Marietta, GA 30067, (404)973-2156. 6809/68000 single board system. 68K ISA bus compatible system. See inside front cover.

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

AAA Chicago Computers, Jerry Koppel, (708)202-0150. SS-50 6809 boards and systems. Very limited quanity, call for information.

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

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Volume Number 1:

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

Volume Number 2:

- **Issues 10 to 19**
- **Forth tutorial and Write Your Own.**
- **68000 CPU for S-100.**
- **RPM vs CP/M, BIOS Enhancements.**
- **Poor Man's Distributed Processing.**
- **Controlling Apple Stepper Motors.**
- **Facsimile Pictures on a Micro.**
- **Memory Mapped I/O on a ZX81.**

Volume Number 3:

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

Volume Number 4:

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

Issue Number 32:

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

Issue Number 33:

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

Issue Number 34:

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

Issue Number 35:

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

Issue Number 36:

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

Issue Number 37:

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

Issue Number 38:

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

Issue Number 39:

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

Issue Number 40:

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

Issue Number 41:

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

Issue Number 42:

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

Issue Number 43:

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

Issue Number 44:

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

Issue Number 45:

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

The Computer Journal Back Issues

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

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

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

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

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

Issue Number 51:
- Introducing the YASBEC
- Floppy Disk Alignment w/RTXEB, Pt 3
- High Speed Modems on Eight Bit Systems
- A Z8 Talker and Host
- Local Area Networks--Ethernet
- UNIX Connectivity on the Cheap
- PC Hard Disk Partition Table
- A Short Introduction to Forth
- Stepped Inference as a Technique for Intelligent Real-Time Embedded Control
- Real Computing, the 32CG160, Swordfish, DOS Command Processor
- PMATE/ZMATE Macros
- Z-System Corner, The Trenton Festival
- Z-Best Software, the Z3HELP System
- The Computer Corner

Issue Number 52:
- YASBEC, The Hardware
- An Arbitrary Waveform Generator, Pt. 1
- B.Y.O. Assembler...in Forth
- Getting Started in Assembly Language, Pt. 3
- The NZCOM IOP
- Servos and the F68HC11
- Z-System Corner, Programming for Compatibility
- Z-Best Software
- Real Computing, X10 Revisited
- PMATE/ZMATE Macros
- Controlling Home Heating & Lighting, Pt. 3
- The CPU280, A High Performance Single-Board Computer
- The Computer Corner

Issue Number 53:
- The CPU280
- Local Area Networks
- Am Arbitrary Waveform Generator
- Real Computing
- Zed Fest '91
- Z-System Corner
- Getting Started in Assembly Language
- The NZCOM IOP
- Z-BEST Software
- The Computer Corner

Issue Number 54:
- Z-System Corner
- B.Y.O. Assembler
- Local Area Networks
- Advanced CP/M
- ZCPR on a 16-Bit Intel Platform
- Real Computing
- Interrupts and the Z80
- 8 MHZ on a Ampro
- Hardware Heavenn
- What Zilog never told you about the Super8
- An Arbitary Waveform Generator
- The Development of TDOS
- The Computer Corner

Issue Number 55:
- Fuzzilogy 101
- The Cyclic Redundancy Check in Forth
- The Internetwork Protocol (IP)
- Z-System Corner
- Hardware Heaven
- Real Computing
- Remapping Disk Drives through the Virtual BIOS
- The Bumbling Mathmatician
- YASMEM
- Z-BEST Software
- The Computer Corner

Issue Number 56:
- TCJ - The Next Ten Years
- Input Expansion for 8031
- Connecting IDE Drives to 8-Bit Systems
- Real Computing
- 8 Queens in Forth
- Z-System Corner
- Kaypro-84 Direct File Transfers
- Analog Signal Generation
- The Computer Corner

Issue Number 57:
- Home Automation with X10
- File Transfer Protocols
- MDISK at 8 MHZ
- Real Computing
- Shell Sort in Forth
- Z-System Corner
- Introduction to Forth
- DR. S-100
- Z AT Last!
- The Computer Corner

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

Issue Number 59:
- Moving Forth
- Center Fold IMSAI MPU-A
- Developing Forth Applications
- Real Computing
- Z-System Corner
- Mr. Kaypro Review
- DR. S-100
- The Computer Corner

Issue Number 60:
- Moving Forth Part II
- Center Fold IMSAI CPA
- Four for Forth
- Real Computing
- Debugging Forth
- Support Groups for Classics
- Z-System Corner
- Mr. Kaypro Review
- DR. S-100
- The Computer Corner

Issue Number 61:
- Multiprocessing 6809 part I
- Center Fold XEROX 820
- Quality Control
- Real Computing
- Support Groups for Classics
- Z-System Corner
- Operating Systems - CP/M
- Mr. Kaypro 5MHZ
- The Computer Corner

Issue Number 62:
- SCSI EPROM Programmer
- Center Fold XEROX 820
- DR S-100
- Real Computing
- Moving Forth part III
- Z-System Corner
- Programming the 6526 CIA
- Reminiscing and Musings
- Modem Scripts

Issue Number 63:
- SCSI EPROM Programmer part II
- Center Fold XEROX 820
- DR S-100
- Real Computing
- Multiprocessing Part II
- Z-System Corner
- 6809 Operating Systems
- Reminiscing and Musings
- IDE Drives Part II

Issue Number 64:
- Small-C?
- Center Fold last XEROX 820
- DR S-100
- Real Computing
- Moving Forth Part IV
- Z-System Corner
- Small Systems
- Mr. Kaypro
- IDE Drives Part III

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

SPECIAL DISCOUNT

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

10% on cost of Back Issues when buying 10 or more issues.

	U.S.	Canada/Mexico (Surface)	Canada/Mexico (Air)	Europe/Other (Surface)	Europe/Other (Air)
Subscriptions (CA not taxable)					
1year (6 issues)	$24.00	$32.00	$34.00	$34.00	$44.00
2 years (12 issues)	$44.00	$60.00	$64.00	$64.00	$84.00
Back Issues (CA tax)	add these shipping costs for each issue ordered				
Bound Volumes $20.00 ea	+$3.00	+$3.50	+$6.50	+$4.00	+$17.00
#20 thru #43 are $3.00 ea.	+$1.00	+$1.00	+$1.25	+$1.50	+$2.50
#44 and up are $4.00ea.	+$1.25	+$1.25	+$1.75	+$2.00	+$3.50
Software Disks (CA tax)	add these shipping costs for each 3 disks ordered				
MicroC Disks are $6.00ea	+$1.00	+$1.00	+$1.25	+$1.50	+$2.50

Items: _____

_____

Back Issues Total _____

MicroC Disks Total _____

California state Residents add 7.25% Sales TAX _____

Subscription Total _____

Total Enclosed _____

Name: _____

Address: _____

_____

_____

_____

Credit Card # _____-_____-_____-_____ exp ___/___

Payment is accepted by check, money order, or Credit Card (M/C, VISA, CarteBlanche, Diners Club). Checks must be in US funds, drawn on a US bank. Credit Card orders can call 1(800) 424-8825.

TCJ *The Computer Journal*

P.O. Box 535, Lincoln, CA 95648-0535

Phone (916) 645-1670

TCJ CLASSIFIED

Needed: Manuals for IMS Corp. 518-W12-10H60, has 10 meg HD. S-100 with 720K 5 1/4in drives running IMS CP/M 2.24J. Robert Edgecombe, 9546 Los Palos Rd. Atascadero, CA 93422.

Needed: Information on MEGATEL QUARK single board computer. This is a Canadian Z80 based CP/M Plus system. Peter W. Borders, 1350 Sunset Dr. Norfolk, VA 23503. P.Borders on GENIE or 71170,77 CompuServe.

Kaypro CPM hardware and software. Plenty of it. Cheap. Send #10 SASE for list. Mike Arman, Box 785, Ormond Beach FL 32175 (904) 673-5576, FAX (904) 673-6560.

OPTICAL PHOTOTYPESETTER! AM-Varityper Model 3510, complete, working. Only $500, was $20,000 new. Four 8 inch hard sectored disk drives, dozens of fonts, sets 4 pt. to 72 pt. type, monitor, processor, dryer, spare parts, complete schematics, all manuals, full lenses, motors, much more, a real treasure trove. This things weighs almost 500 pounds. Use it to set type, or tear it apart for the time of your life. (904) 673-5576, Fax (904) 673-6560.

Avaialble: *MicroC* Kaypro Disks, *MicroC* Back Issues and legal copies of CP/M, and more. Many bootable CP/M disk formats avaialble. Disk copying, most formats inculding Apple CP/M. Manuals and more! Lambda Software Publishing, 149 West Hilliard Lane, Eugene, OR 97404-3057, (503) 688-3563.

NEW CLASSIFIED RATES!

NOW $5.00 PER LISTING!

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

Commercial Advertising Rates:

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

Send your items to:

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

Reader to Reader continued;

I need a way to copy thie data on these diks to ANY CPM or DOS format, from which I will then be able to import the files into the system I'm using now. These 8 in disks use a proprietary format of 24 sectors of 12K keystrokes per sector. I believe the capacity of the disks is 188K, but I'm not sure. The operating system is pre-CP/M, but uses an 8 bit bus.

I have several of the correct 8 inch drives available, and can provide the required power to them (5,12,24, and 117 Volts!), but the drive controller is on two large cards with several dozen chips each, and plugs into another strange and wonderful bus interface.

I also have available several Kaypro 2, 4, and 10s, and all kinds of DOS stuff. Is there anty way I can trick the Kaypro controller into thinking that it really does want to talk to an 8 inch hard sectored drive? Unfortunately, the typesetter has no serial port, so that avenue is closed.

There are two or three places offering disk conversion, but they all act like they are doing me a great big favor by charging me lots of money. Any help would be appreciated.

Mike Arman, Ormond, FL.

OK Mike, sounds like a good problem. I doubt that the Kaypro could help without some major ROM changes. It is possible to do some full track reads that would give you the data plus the disk information. Later a large amount of editing would be needed to pull out your data.

Have you talked to Lambda, David does disk copying at reasonable rates and can do hard sector I think. The getting the correct format sounds like the real problem, as you really just want your data files and not everything else. That means whoever copies the disks must know or have a similar system, a real problem if they were pre-CP/M.

If the data is real valuable then whatever the cost someone who can do it charges will be cheaper than not getting it at all. Let us know what happened. Thanks. Bill Kibler.

Send your letters to:

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

The Computer Corner

By Bill Kibler

Regular Feature

Editorial Comment

PLC?

What is A PLC?

Since my column of last month stated I now work on PLC's, several people have asked what that means. Since we are here to teach and enlighten here is an explanation of what a PLC is.

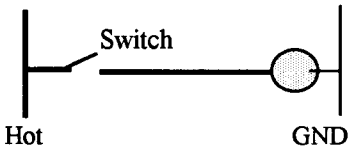
A Computer

PLC's are simply computers used for industrial control. What does PLC stand for? Programmable Logic Controller. The idea is to be able to change the flow of operations with simple keystrokes and not physically "re-wire" the control circuit.

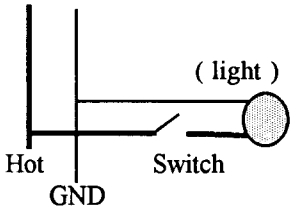
Since even that explanation might be a bit vague for many, lets start from the top. Industrial electric circuits are controlled by a ladder logic of wiring. Your house wiring is in the form of a ladder. Each circuit has a separate rung. Industrial controls are just the same.

A simple Ladder Circuit

Let us take your house and look at a bedroom. The bedroom has a light switch, that is used to control the lights. Now most people who have any idea of the wiring would not call this a ladder circuit, but it is. If we were to draw it in an industrial or theoretical manner, it would look like this:



Now the wires, and this is where most people get confused, are usually inside one outer jacket. So when the electrician wires the circuits both the Ground wire (GND) and the Hot wire or 120V are side by side in one cable. Like this:



Since the ground is the same potential it is convenient to draw it as one single path. The same is true for the hot side. The important item here is the need for the appliance or light to be connected between the two sides. Electricity flows from the hot side to ground given a completed path (hopefully not your body). Our simple diagram shows a light switch, that when closed connects the hot side to one side of the light bulb. Since the other is grounded, your light will be on.

We can expand this idea in an industrial environment, by making the light, a motor, a pump, or a whole series of devices. The switch can be sensors that detect a low condition in a water storage tank, and thus would activate the pump to fill it back up. The average washing machine has a similar ladder diagram where the timer switch will activate various operations in a given sequence. There will also be safety features, such as not allowing you to spin your cloths if the lid is open.

For many years these operations have been done by using relays, switches, and other mechanical devices. I worked on many ladder logic layouts, that were composed of relays and massive bundles of wires going and coming from the different devices. To make a change in the process would require a physical change in the wiring and maybe even a different type of relay or switch. I need not really comment on the problems associated with mechanical devices as I am sure everyone has had some switch or control fail on them more than once.

The industrial wizards that be, figured that a computer could then replace all these physical devices and one would only need inputs and outputs to and from the computer. Since most industrial technicians were use to and understood ladder logic designs, the program developed to control these computers was programmed in ladder Logic as well. Thus the name programmable logic controllers.

The program is entered by actually placing input, usually a reference number (actually a memory bit) on one side of the ladder and placing the output number (again a memory bit, only this one associated with an output device) on the other. The display device would connect a line between the devices as shown below:

```
|                    |
|-(501)----<801>-|
|                    |
```

To have a delay from the time the switch is closed and power is actually applied the circuit would look like this:

```
|                    |
|-(501)--[Tim01]-|
|        [#0010]  |
|                    |
|-(T01)----<801>-|
|                    |
```

The delay is indicated by the #0010, or 1 second of delay with a resolution of tenth of seconds. Before PLC's you would need to changed the relay from a regular normally open to a delay on close relay. Generally the time is fixed, or an adjust-

able screw on the top varied the amount of time, and seldom with .1 second accuracy.

Since working on the PLC, I have had a chance to think about what is actually going on. The computer that runs the PLC has several loops which it goes through. The primary loop goes something like this; Check input ports/devices and make memory location be same state (0=off, 1=On); Step through ladder program; Make output ports/devices same state as memory bit values.

The internal layout of a PLC is something like this; A memory or I/O addressed input and output array of devices (these are the signals in and out from the PLC); A program memory area to contain the user program; A memory map representing the actual I/O devices (maybe arranged differently than actual devices i.e. 8 bit versus 16 bit); A memory area containing the supervisory PLC program.

When running the PLC program, it checks inputs, then does the program you have entered, changes outputs and sends the signals back out. The actual testing of ladder rungs is very simple and somewhat Forth like. The machine I use is very much like most machine currently in the industry. Ladders have become very complex as they often end up talking to regular computers.

To allow for more complex operations, extended instruction are provided that perform normal AND, OR and Data Move type operations. To enter those you use a mnemonic mode or a very word (Forth like) instruction. "LOAD" means to test or get the status of a given input or bit, as in "LOAD 501". I imagine this gets the bit and leaves it on the stack or at least the status of it (zero or non-zero). You then have a "OUT 801" instruction which most likely sees if the stack is zero or not and if not zero makes the location "801" active or ON.

This part is so simple and straight forward I feel I could take almost any Forth and turn it into a PLC in about half a day. Now of course there are some more complex instructions and operations than those I listed and they might give me a bit of trouble. But what I wanted to cover was the basic concept and operation and leave implementation to PLC makers.

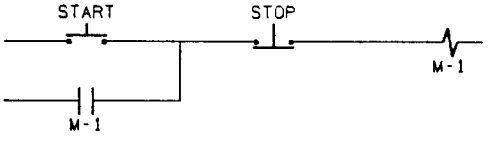
How these PLC's actually do what they do is very much a secret. We are interfacing serial (RS 485) devices to the PLC using an ASCII/BASIC module. The main idea of the BASIC interface module is to be able to output text strings to monitors and display devices. This replaces the "now what does that blinking light mean" with a real display that says "PARTS FALLING OFF ASSEMBLY LINE!!!!" In our case we want to do much more than that and there in comes the problem.

The PLC is very good, but it is made in Japan and as such that is where all the inside information about the interface is kept. We would like to do some real machine level interfacing, but the interface between the BASIC module and the PLC is very convoluted and you can only use their supplied instructions. Needless to say their instructions will not do it the way we need to do it.

What this reminds me of is how I got into Forth. Many years ago I was in a similar situation, needing to do things that BASIC just wouldn't let you do. I did eventually figure a way around BASIC, but it was so much a mess, I figured there had to be a better way. I tried many options and found only Forth really had the power and tools I wanted. That then was how and why I started using Forth. I just wish PLC companies would openly use Forth instead of the BASIC they often provide.

Next Time

Hopefully I have filled in a little about PLCs and their operation. If your tasks are simple and straight forward, PLC's are very easy to deal with and many times simpler than learning a computer language. Till next time, keep hacking that hardware.



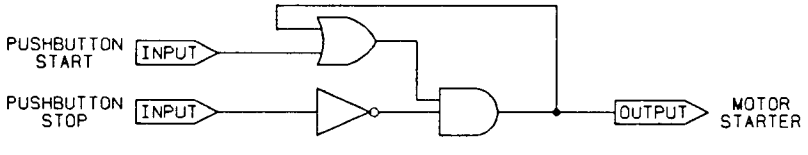
RELAY LADDER SCHEMATIC

IF THE STOP BUTTON IS DEPRESSED, TURN THE STARTER OFF. OTHERWISE,
IF THE STARTER BUTTON IS DEPRESSED, TURN THE STARTER ON. OTHERWISE,
LEAVE THE STARTER IN ITS PRESENT STATE.

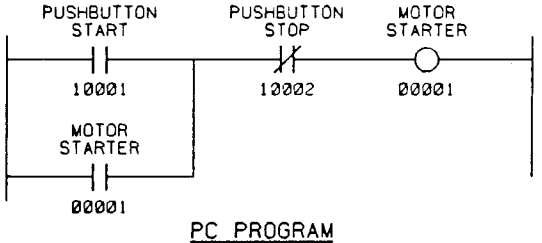
ENGLISH DESCRIPTION

```
10 INPUT A 'PUSHBUTTON START'
20 INPUT B 'PUSHBUTTON STOP'
30 IF A=1 AND B=1 THEN C=1
40 IF B=0 THEN C=0
50 OUTPUT C
60 GO TO 10
```

COMPUTER PROGRAM



SOLID STATE LOGIC



PC PROGRAM

Examples of controlling a motor, from PROGRAMMABLE CONTROLLER HANDBOOK, by Robert E. Wilhelm, Jr., from Hayden Books, ISBN 0-8104-6311-3.

TCJ *The Computer Journal*
Post Office Box 535
Lincoln, CA 95648-0535
United States

BULK RATE
US POSTAGE
PAID
Lincoln, CA
PERMIT NO. 91

ADDRESS CORRECTION REQUESTED
FORWARDING AND RETURN POSTAGE
GUARANTEED

Telephone: (916) 645-1670