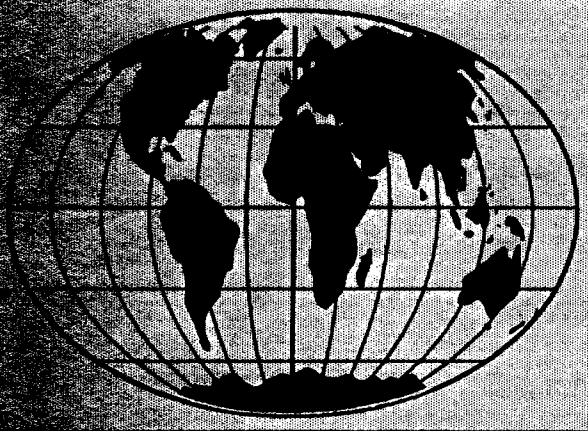
Small Scale Computing Since 1983

Supporting the *Trailing Edge of Technology*



The Computer Journal

Issue Number 77 | January/February 1996 | US $4.00

Reader to Reader ..... GIDE News and more

Real Computing ..... 32-bit OS Reviews

Mr.Kaypro ..... Kaypro External Video

The European Beat ..... GIDE on the KC84/85

Hands on with PLD's ..... Mem Decoder/Clock Gen

Dr. S-100 ..... S-100 Memory Management

Center Fold CPU280

The First TRS-80 ..... Model 1 and BASIC

Small System Support ..... Prime Numbers in 'C'

Program This! ..... Z80 SIO

Morrow MD-3P ..... Repair

Computer Corner ..... Forth Day 1995

ISSN # 0748-9331

Hands-on Hardware and Software

The Computer Journal

Founder
Art Carlson

Previous Publishers
Bill D. Kibler
Chris McEwen

Editor/Publisher
Dave Baldwin

Technical Consultant
Bill D. Kibler

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

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

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

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

Registered Trademarks

TCJ The Computer Journal

Issue Number 77, January/February 1996

Editor's Column 2
And in this issue...

Reader to Reader 3
GIDE news.

Real Computing 7
OS reviews, INTERLNK, and the Pocket Programmer
By Rick Rodman.

Mr. Kaypro 9
The last of the external video mods.
By Charles B. Stafford

The European Beat 12
GIDE on the KC-85.
By Helmut Jungkunz.

Hands on with PLD's 14
Clock generator and Memory Decoder
By Robert Brown

Dr. S-100 20
S-100 Memory Management
By Herb Johnson.

Center Fold 25
Tilmann Reh's CPU280

The First TRS-80 29
The Model 1 and BASIC.
By Gary Ratliff.

Small System Support 34
Prime Numbers in C.
By Ronald W. Anderson.

Program This! 39
The Z80 SIO
By Dave Baldwin

Morrow MD-3P Repair 43
MD-3P repair details
By Jay Huddleston

Support Groups for the Classics 46

Back Issues 48

The Computer Corner 50
Forth Day 1995.
By Bill Kibler.

Editor's Column

And in this issue...

On my first issue, I'm little bit late. I decided that it was better to try to get a good issue out rather than a quick one. I'm sure you'll let me know if I did or didn't. Add that to some project deadlines that occured at the same time and here we are.

And since I'm rushing to get this to the printer tomorrow, I can't find my editor's soapbox. So all I have this time is some news.

The one thing this issue is missing is a Forth article. I hope our Forth authors will send me at least one for the next issue.

I was hoping to have an article by David McGlone of the Z-Letter in this issue. The story of what it took to make some of the CP/M software available should be interesting. Unfortunately, he was just too busy.

Herb Johnson may be taking a breather after #78 for the same reason, too much work to do. We'll see articles from him when he has time.

Also because Herb has become real busy, it looks like *TCJ* will become the US distributor for Tilmann Reh's **GIDE** board. There will be more about this in #78 because the details aren't finalized yet.

Dave Baldwin
TCJ Editor

Events

Trenton Computer Festival is scheduled for the weekend of April 20-21. It has a CP/M and Z-System conference.

1996 Rochester Forth Conference will be held June 19-22 in Toronto at Ryerson Polytechnic University. Conference info available on the Web at "http://maccs.dcss.mcmaster.ca/~ns/96roch.html" or from Elliott Chapin at 416-921-9560.

TCJ Phones and Internet Addresses

The Computer Journal
(800) 424-8825 or (916) 722-4970
Email: tcj@psyber.com
BBS: (916) 722-5799
FAX: (916) 722-7480
Dave Baldwin: dibald@netcom.com
WWW Home page: "http://www.psyber.com/~tcj"

This issue includes the first of the **TCJ Reference Cards**. This time it's the Z80/180 instruction set. It's put inside the Center Fold with a single staple and intended to pulled out and used. Each issue will include a TCJ Reference Card related to one or more of the articles in that issue.

There's been a lot of activity with **GIDE** since the last issue. Reader to Reader is filled GIDE news and Helmut has some more in in **The European Beat**.

The **Center Fold** this is Tilmann Reh's CPU280. I wanted this to be an example of a modern CP/M system. However, now it's 'obselete' because Zilog decided to stop making the Z280 CPU. We finally publish the schematic for it along with the code for the PLD's.

Speaking of PLD's, Robert Brown (of Alta Engineering) wrote **Hands on with PLD's** for this issue. He gives examples of a clock generator and a memory decoder for a Z80. He shows how easy it is to decode odd size memory regions with a PLD. He has also uploaded files for building the Alta Engineering PLD programmer, a digital storage scope, and a Logic Analyzer with the PLD source files to the TCJ/DIBs BBS.

A new feature in this issue is **Program This!** This column will focus on a specific chip or device and provide enough info for you make it work. This time, I'm writing about the **Z80 SIO.** The article provides code samples for both polled and interrupt mode operation and complete source code for two different programs will be available on the TCJ Web and TCJ/DIBs BBS.

Gary Ratliff has an article about **The First TRS-80**, the Model 1 and tells about programming it in two different versions of Basic.

Chuck Stafford, **Mr. Kaypro**, is back with the last of the external video mods for the Kaypros and schematics for three different versions.

Our other regulars are here too. Rick Rodman has OS and programmer reviews in **Real Computing**. Ronald Anderson shows methods in 'C' for finding prime numbers in **Small System Support**. And Herb Johnson, **Dr. S-100**, talks about S-100 memory management.

In The Next Issue:

Number 78 will include articles about a homebuilt TTL processor, a DIY 6502 circuit, and the 8031 along with all the regulars.

READER to READER

Letters and News

All Readers

MINI Articles

The letters this time are more like the GIDE News column. Read Helmut's article for even more. We also have a message about a new programmer's organization for CP/M software and a couple of help requests.

- -

Hello Dave,

After having read the current TCJ issue #76, I have (as usual) some comments. But before I start commenting, there is another actual theme to talk about. Thanks very much for your message indicating that the Z280 is going out of production. When I previously asked my distributor, they always told me they'd inform me when any product I've ever bought there is cancelled. However, they didn't tell me, and still they can't explain how this could happen. But, fortunately, they have a current order which they will get in spring, and I can have some of those "last Z280 chips". Now I already ordered a few, to have spares for the CPU280 systems which are already in use. But I also would like to note that this is about the last possibility for getting a CPU280 system! If there is anybody out there who wants to take this chance, please leave me a note. If there are some orders, I will try to get more Z280's - however, they're too expensive to put many of them on stock just in case anyone wants a CPU280 later.

Now on to the comments on TCJ #76. In his column about XT's, Frank Sergeant also covers modern hard drives. Because there still are some questions, let me shortly explain the "difference" between IDE and the so-called EIDE. The truth is, there is *no* difference! If that sounds surprising, please think of the following details. As described earlier in my IDE article series, IDE uses an 8-bit sector number register and a 16-bit cylinder register, while the head is contained in the lower four bits of the SDH register. This gives a maximum capacity of 65536 cylinders by 16 heads by 256 sectors by 512 bytes, aka 128 Gigabytes!

Now let's look at how PC's do hard disk accesses. Their ROM-BIOS masks the cylinder number to 10 bits and the sector number to 6 bits, since the very first hard disk controller was limited to those values. The resulting accessable capacity then is 1024 cylinders by 16 heads by 64 sectors by 512 bytes, which is 512 Megabytes. Now does this value sound familiar to you? Right, it's the so-called "IDE capacity limit", but it is not caused by the interface or the drive, it's just because the PCs ROM-BIOSs never stopped masking off those higher bits!

When it became obvious that 512 MB were not enough, the manufacturers simply blamed the drives and their interfaces, and created a "new" standard which would remove this restriction: EIDE. In fact, physically and logically EIDE is identical to IDE. The only thing that was changed was the bit-masking in the ROM-BIOS: so-called EIDE controllers (or capable PCs) simply don't mask off any bits from cylinder and sector registers. This is confirmed by the fact that you can mix-up either type of drive and "controller": it will always work perfectly. Just the old PC with its "IDE-controller" will be unable to access anything beyond the 1024x16x64 limit. It's only a question of software. For those "older" PCs there are drivers which replace the ROM-BIOS driver and don't mask any bits off, and with such a driver it's perfectly possible to completely use so-called EIDE drives on an old so-called IDE controller. (BTW, there are many older IDE drives which have more than 1024 cylinders, and the drives itself were never limited to this value.)

This is just another case of willing disinformation in the PC business! (There are some more, like diskette capacities which are far less than would be (legally) possible, just because of IBM's thumbness!)

Another detail mentioned by Frank is LBA. LBA stands for "Logical Block Address(ing). Its meaning is rather simple: the sector number, cylinder, and head registers are logically combined to form a 28-bit absolute sector number. With LBA, you don't have to care about any drive's geometry, just access the sectors by their absolute number. Drives which are capable of LBA, indicate this by a status bit in the "Identify Drive" result. For each command, the desired mode (LBA or CHS, for Cylinder Head Sector) must be explicitly set.

I think LBA is an attempt to eliminate all those geometry emulations which were caused by some unflexible BIOSs (see? again a PC software problem), and it also provides similar access as with SCSI drives (the main competitor of IDE). But for compatibility reasons, all PCs will surely continue to support CHS addressing for many more years...

I have some more comments on TCJ #76: In his column, Herb Johnson partly confused the history of the GIDE development. My first IDE interface was a board for the ECB bus, accompanying the CPU280. This board was described in TCJ #56, more than three years ago. My circuit was used by Wayne Sung as a base for his development, a smaller interface for the Epson QX-10, which used an EPROM state machine instead of my GAL approach. About the same time, Claude Palm seemingly started developing the single-chip IDE controller PLD for use

in his Z180 SBC. However, I don't know if he was also inspired by my article in TCJ, or if we just had the same idea at the same time. In further discussion of his PLD chip, Herb Johnson, Bill Kibler, and I came to the conclusion that there was a need for a small and cheap interface which would connect to as many different computers as possible, so that was the start of the GIDE development.

The story of Claude Palm fighting his spurious FDC error is very interesting. Because of troubles like this, I always decode /IORQ with one of /RD or /WR to get valid select signals. This way, /M1 is not needed for the decoder at all. A very good design rule to prevent such problems is: Always use signals which are *active* to generate select signals, and never use *inactive* signals for this! In this case, never use "/M1 inactive" as a qualifier for I/O selection, but use "/RD or /WR active" for this purpose. This is a general rule which is a great help in many cases.

When Claude writes about his "18 MHz Z180", I assume this is not the real working frequency of the Z180. More probably, it is the frequency of the crystal which is devided by two to form the CPU clock. To my knowledge, the fastest Z180 chip available is the 12.5 MHz version. We should always use the real CPU clock, not the crystal frequency, when talking about clock frequencies anyway.

Regarding Bill Kibler's comments on CD-ROM access for CP/M: We are currently checking the possibilities for connecting a CD-ROM drive to GIDE. In my opinion, the best method will be a file transfer shell which allows you to navigate through the hierarchical file system of the CD-ROM and transfer selected files to the CP/M host. I don't think it makes much sense trying to establish a CD-ROM as virtual CP/M drive - the driver would be too complex, and would need additional hooks for accessing the different directory levels of the CD-ROM. That transfer shell could be about the same style as my "MSDOS Disk Emulator" for CP/M-Plus, which performs the same task for MS-DOS diskettes on any CP/M-Plus computer. However, those Linux CD-ROM driver sources surely are a good source of information!

Wow, this is a rather long letter - more like a mini article. BTW, hopefully I will find the time to explain and describe GIDE in detail in a future issue of TCJ.

Greetings, Tilmann Reh

Ed: The Zilog databooks list the Z8S180 as operating to 20MHz. See also Jonathan Taylor's message about GIDE and CD-ROM.

- - - - - - - - - - - - - - - - - - -

From: wayne sung
<wsung@jessica.Stanford.EDU>
Subject: Re: Z80 SIO

It's been a while back, at a different job, that I did the SIO circuit, so this is from memory. What I was doing was trying to generate a stream of pulses to run a number of time clocks. These used a data format completely unlike rs232. I had a single board computer with a Z80 and SIO, so I took the incoming data from a radio clock into one of the SIO ports, changed the format, and was going to use the other port to generate the pulse stream to the clocks.

The clocks needed something like a pulse-width modulated signal at about a 2 kHz bit rate. This was where I found that using break didn't work. The clocks would never see the data. Using a scope I saw that the pulse widths were completely unpredictable. Changing to a control lead solved the problem. I seem to remember a statement in the data sheets that said something like break can be toggled at any time. Apparently it was not meant to say break will actually show up when I asked it to.

One interesting part of this project was that there is no ram at all. Since I didn't do any subroutine calls, and I didn't need many variables, everything was done in the registers (don't remember if I used the alternate set or not) and a 2 kbyte rom.

Another interesting part had to do with two different sources of timing information. Originally the clocks were driven from a generator which was "synced" to WWV. The reason the quotes are there is that it took a human to do the sync, i.e. someone listened to the WWV radio and pushed a set button to lock the clock.

After we obtained a WWVB clock to do NTP with, someone asked me if I could use it to run the building clocks as well. Being the gadget freak I am, of course I tried it. Now the original generator put out a continuous signal, even though the digits were latched. So I decided I would put out only two updates a second (the time is..., I repeat the time is...)

Somewhere in the building there is a bit of crosstalk between the clock drive line and an intercom system. So by listening to the intercom one can tell which generator was driving the clocks. The original generator had a continuous 2 kHz tone, mine had two clicks close together every second. So I told them one was an analog clock and one was a digital clock.

Ah, the IDE project. Please forgive the long-winded introduction here, but the whole thing hinged on TCJ so hopefully you'll not be too bored.

I first "met" John Baker on the net. Still have never spoken to him personally, though there has been much email. What happened was somehow we got talking about something (don't even remember what now) and wound up fixing his Epson QX-10 completely via email.

I would ask him to try this or that, send measurements etc (boy it's hard to simulate a scope on email). Eventually we zeroed in and by faking one pin the machine came up. Turns out the caps in the power supply had dried out causing a lot of ripple on the 5v line so the power supply was keeping reset active.

John wrote a letter to TCJ about himself and included this particular episode as part of it.

At the same time, Tilmann Reh had also written up his IDE design in TCJ. It turns out one Mr Roche in France wanted some kind of HD adapter for his QX-10, and approached Reh about converting his design. Reh said he did not have a QX-10, hence couldn't do it, but that John Baker had one and to check there.

Of course John sends the request to me. I had a large bag full of registered high-speed eproms that were pulled from some comm equipment. I wanted to try using these for lookup table logic design, and this proved to be a workable project.

I was using mostly Reh's writeup for IDE description, but there was no feel for whether one polarity or the other of several pulses that might

work better. Rather than put in extra inverters, I put all the sequencing stuff (except one flip-flop) in the eprom. It also decoded whichever range of bus addresses I wanted.

This way I could just change the eprom bits to change polarities. Indeed two pulses wanted to be opposite what I assumed.

The final design had one LS08, one LS107, two LS646 and the eprom. Of course a pal design is just as possible, but I personally have had bad results with them. I think some other people I have corresponded with might have made pal designs.

One of the few times I beat Murphy to the finish line, but just barely. My QX-10 was already limping, having lost one floppy drive. Almost the very moment I got the first bios that really worked the second floppy died also, and I couldn't even reboot. I wrote up what I did, and John polished the bios. What I had done was change the sizes of the drives from 2 x 5 MB, which is what the bios had, to 2 X 8 MB. I did not think to expand the allocation vector, and eventually there would have been some funnies. John caught this and finished the changes.

Roche did not have email, a lot of this was done USPS, so it took maybe a year end-to-end. In the meantime I changed coasts and now don't even have any space to do 8-bit stuff any more.

John's link has been producing a steady stream of interest. The funniest request I got was from someone who wanted directions to build an IDE for a pc.

- - - - - - - - - - - - - - - - - -

Ed: I received this next message from John very early one Saturday morning.

From: "John D. Baker" <jdb8042@blkbox.COM>
Subject: GIDE BIOS!

Just ten minutes or so ago, I brought up the first iteration of a CP/M 2.2 BIOS extension to support the GIDE on my Davidge DSB 4000 single-board computer!

I've just barely started testing. I'm going to bring up ZCPR-D&J as a minimum and then move to NZCOM to make the testing process nicer. Naturally, my BIOS extension includes a ZDDOS clock driver...

The Davidge CP/M system has to be relocated for a 62K system and the GIDE extension is loaded at the top of memory and patched into the standard floppy-disk BIOS (DSB 4000 v1.3). I'm able to save quite a bit of space by using the data areas of the floppy-disk BIOS and only have to duplicate the disk-handling code (but it's greatly simplified).

As I've implemented it now, the floppy disks remain A: through D: and the hard disk is E: through I:. Naturally, I have to boot from the floppy disk at cold-start.

More as things develop, or as I develop things.

John D. Baker
http://www.blkbox.com/~jdb8042/

- - - - - - - - - - - - - - - - - -

And this one is about a CD-ROM interface with the GIDE.

From: johntayl@spuddy.mew.co.uk (Johnathan Taylor)
Subject: Re: CP/M with alternate BDOS support

(This message is in response to a message posted by Herb Johnson about the GIDE.)

Here's confirmation that Tilmanns G-IDE interface that Herb' speaks of not only handles big AT-IDE hard-drives flawlessly but it also works fine on ATAPI-IDE CD-ROM drives at the same time! I'm currently working on porting a POSIX CD-ROM reader util to z80-CP/M so that G-IDE users can access the CP/M and C-UG CD-ROMs directly! NO bios support will be required as it'll contain it's own driver. I've yet to decide how to implement the user definable I/O location, there's three choices:-

1/ End user edits sources and recompiles.
2/ Incorporate a patch table ie a cut down ZMP15 style jobby
3/ use the Hi-Tech C 3.09 "ENVIRON" file to store the custom settings in plain text form for the really non-technical!

I think 3 would be the most user friendly but 2 would be the most rugged across search search paths, etc., and 1 means that the end user must have Hi-Tech C and enough TPA/ floppy storage to perform the compile. For now all I need is a UK source for the CP/M CD-ROM itself as the CD-ROM's I've got were donated, bought-in-error, games-console CD's so don't have much of an iso-filesytem to test the utility ie. they only have 4 files and it's easier just to read the dir-block and manually load the 4 files!

Regards, Johnathan.

Ed: I received this note from Johnathan also.

I have obtained a copy of the CP/M CD-ROM and have manually retrieved a couple of files using raw block reads to decipher the CD-ROM directory structures and simply spooling consecutive blocks to a file to retrieve the actual file. Next comes the task of teaching the computer to search the CD-ROM itself.

- - - - - - - - - - - - - - - - - -

And GIDE on the TRS-80.

From: Pete Cervasio <cervasio@airmail.net>
Subject: Got the GIDE (mostly) working

Hi, Dave.

As the subject says, I managed to get the TRSDOS hard disk driver modified to work with the GIDE. It needs a lot of work before I would call it finished, but I've got data on a 340 meg drive as I type. I haven't fixed it to allow a logical drive to start at any physical cylinder, and there's still the limit of 202 tracks (with 2 physical = 1 logical), 8 heads and 32 sectors per track. The cylinder offset will fix the track limit, while the others will take fundamental changes to the way the drive control table is defined. I think there are enough bits that aren't used with hard drives that I can fix it.

I just started to write a bunch of stuff about the inner workings of TRSDOS, explaining what I'm running into. It'll be better in the article, which I've started on. How many words should I shoot for as a maximum before I split it into two or more parts? If you want to print all the code.. the driver winds up at 30 pages, and the "formatter" is about the same. Most of the driver listing is asking the user questions, and the same is true of the formatter. I could chop those parts

out for the article and just say what they need to accomplish.

How about a couple of photos of the completed machine, with the extra power supply and the GIDE and IDE drive installed? I had to apply a hacksaw to the RF shielding to make it fit over the GIDE. There's not a lot of clearance there. I had an old 63 watt IBM-PC power supply that I liberated from it's case (the circuit board, I mean). That's mounted in the bottom of the computer and powers the hard disk and the two floppy drives. The supply that powers the rest of the machine seems happier now that the floppies have "disappeared". :-) The PC supply just squeaks in under the monitor, with maybe 1/2" clearance between a big resistor and the tube. The hard drive (and a 3.5" 720k floppy) are taking the place of one of the 5.25 floppy drives. Personally, I think it looks great.

Well, I better get to bed. It's way too late and I'm getting "punchy". I just had to tell someone about this first. :-)

Talk to you later., Pete C.

- - - - - - - - - - - - - - - - - - -

From: Steven Young
<syoung@nucleus.com>
To: tcj@psyber.com
Subject: Greetings!

Greetings!

A few CP/M users and I have recently founded a fledgling organization - the CP/M Programmer's Organization, or CPO. The goal of this organization is to breathe some fresh life into the CP/M scene. After all, most software is at the least six years old, and has, on the whole, become just a wee bit stagnant.

The two major goals of the CPO are to:

1) Bring current public domain software up to speed with current developments. Obviously, nothing too flashy, since CP/M does only allow sixty-four K ram, but, for example, grafting OOP onto a C compiler, or even creating a C++ compiler, might be nice.

2) To create new software to fill the void of new software which has occcurred. For example, porting vi to CP/M is one of the projects we're looking into.

Mind you, it's still, as I say, fledgling, and as of yet, we're still in the process of getting organized. But nevertheless, initial interest has been amazing, and I seriously think the CPO will prove to be, at the very least, an educational exercise. (and a bit of nostalgia for the greybeards, of course. ;))

Ed: I don't know who he could be talking about. We haven't published any photos of any greybeards.

- - - - - - - - - - - - - - - - - - -

From: Ron Anderson
<RWilAnders@aol.com>
To: tcj@psyber.com
Subject: Antique Laptop

Someone gave me a non-working Toshiba Laptop model T1200. I have it working but I would be happy to find any documentation for it. It has a 10 Mhz 8086, one floppy drive 3.5" 720K, an LCD screen, a NiCad battery pack and a charger/power supply. Having the 8086 ought to qualify it for antique status. I'm thinking of offering $20 for it. I can use some of my "antique" software and use it as a word processor. I have used that same antique editor to write columns. If nobody there knows anything about it maybe you could put in a "letter" asking for help.

Thanks, Ron Anderson

Ed: Here you are, Ron.

- - - - - - - - - - - - - - - - - - -

From: Stephen Stone
<stephen@silcom.com>
Subject: Re: WTB: CP/M Formfilling Software

I was one of the original members of the KayPro Users Group of Santa Barbara in 1983. In 1987, our membership combined with that of the Osborne Santa Barbara Users Group and became the Santa Barbara Classic Computer Society. Our meetings still take place in the Goleta Library at 7 p.m. on the second Tuesday of each month. Attendance ranges from 8 to 14 persons. We had an anniversary celebration in 1993 with 25 people attending. Most discussion in meetings is mix of 8 bit nostalgia and discussion of what we consider to be ideal use of computer technology and technology in general. We actively help newcomers and oldtimers seeking information and help with older computers. We charge dues only when we find our treasury running low. Our monthly newletter survives because CP/M programming genius Al Paarman continues to write interesting and irreverent articles for it.

My garage has more than 10 and less than 20 old 8 bit machines stored in nooks and crannies at any time. One of our members must have close to 200 old computers!

I still use my KayPros for word processing and keep my small business accounting with Checks And Balances. In the last year, I introduced one of our department managers in my hospital to one of my KayPros because our non-profit budget will not allow the purchase of computers for our department and her correspondence load was overwhelming our unit secretrary. The manager endured all kinds of gaff from visitors to the department for having this strange box on her desk — she actually put the machine away because she was embarrassed to be seen with it! Then the burden of increasing work convinced her to try WordStar on the KayPro again. Now she loves it and won't part with the loaned 4-84 .... one less machine in my garage, I guess.

I've never gotten into ZCPR — but have used all of Plu*Perfect Systems' enhancements. While I am not a computer professional, my background in CP/M machines has taught me to think about computers in an meaningful and evolutionary way: seeing computer science as more representative of what people can do with extensions of their knowledge about the need to work and communicate, than as a means to create appliances to be bought for the short-lived thrill of gee wiz features.

My latest desire for some of my 8 bit machines is to have them help me and my friends at work complete the mass of charting on preprinted forms which is now required in any health care setting. If I can find the form filling software for the KayPro which will allow us to type rather than handwrite our medical record entries, it will allow so much more time for care to patients.

Please read Stephen's ad in the Classified's too.

Real Computing

By Rick Rodman

32-Bit Systems

All Readers

Free BSD, NextStep

FreeBSD 2.0

It took some doing, but I managed to collect together enough components from various computers to build one that will run FreeBSD 2.0. FreeBSD only supports a couple of SCSI host adapters - Adaptec and Buslogic.

As with most experimental operating systems, the installation process is nightmarish. Nothing is automated at all. There is a Readme file which explains the mysterious "Disklabel" procedure - but you can only see it from MS-DOS. Once you begin the installation, you can only see a very limited "tutorial" - which is a cruel misnomer, since it gives neither step-by-step instructions nor examples.

Why is it so hard to write installation programs? There are two well-known, commonly accepted principles: First, show all choices to the user at each decision point, with automatic defaults which clearly show the most likely user choice. Second, the user should make all the decisions up front. The user should not have to make a choice, wait five minutes, make another choice, wait five minutes, and so on. The FreeBSD installation procedure flunks both of these principles. Is this because Unix gurus are elitist snobs who feel that neophytes must be forced to struggle? Or because their brains are so warped by repeated mystic incantations that they truly believe that small letters and uppercase letters are totally different and unrelated? Or are they unable to write an installation procedure because their computers already have umpteen copies of FreeBSD already on them? I think it's the latter. Actually, the FreeBSD folks really did try - they made an install script that automates some of the work - but they have a much longer way to go.

The installation has four parts: Fdisk, Disklabel, rebooting, and installing "distributions".

The Fdisk program is pretty standard, with one little gotcha. There is a command for writing a boot record and another for writing an "MBR". To me, MBR stands for Master Boot Record and so these ought to be the same, but they aren't. The MBR refers to a multiple-operating-system boot routine which doesn't appear to work, so just use the regular "w".

Here's the story behind the Disklabel part: FreeBSD calls disk partitions "slices" and subdivides these with a program called "disklabel". There are eight subdivisions which are named "a" through "h", but on a fresh disk they initially contain garbage. You need to delete "e" through "h" first, then do a (W)rite. Next, you need to (E)dit "a" for about 18 megabytes, "b" for about 16 megabytes, and "e" for what's left. You can't change "c" or "d". Do another (W)rite. Next, you need to (A)ssign "a" to mountpoint "/", (A)ssign "b" but you don't enter a mountpoint, and (A)ssign "c" for "/usr". Then do a (W)rite and a (Q)uit.

When you say (P)roceed, the machine has to reboot. This is a surprisingly iffy step. Actually, most of the 32-bit OSs have a reboot halfway through, including OS/2 and NT, and it's a sort of checkpoint as to whether things are going to continue to work.

Installing the distributions is a very long process, requiring you to make a decision, wait anywhere from a few minutes to half an hour or more, make another decision, wait another long time, and so on.

The installation crashed on me while installing Xfree86. Once this happened, I couldn't find any way to resume it. I didn't want to start completely over because I'd already spent over 4 hours on the process. I logged in as root and poked around. The mystic incantation to mount the CD-ROM is:

```
mount -t cd9660 /dev/cd0a /mnt
```

Which is similar, but of course completely different in all the details, to mount commands used under Linux or SunOS. This is one of the pitfalls of Unix: Because so much is installation-dependent, you can be a Unix expert and yet not be able to find or do anything in a different version.

The file INSTALL, which can be read from the CD, mentions briefly that the installation can be resumed with /sbin/sysinstall, or that a "package" (same as a distribution?) can be added with pkg_add (which turns out to be in /usr/sbin). Both of these statements appear to be wrong; there's no apparent way to resume an installation. So, I couldn't test X Window. The text-mode OS works fine, but I didn't see anything new or different. In fact, much of what you get with FreeBSD is exactly identical to what you get with Linux, with the same source code.

In summary, FreeBSD has little to recommend it over Linux. Linux costs less, includes more, and is much easier to install. I hate to denigrate somebody's labor of love, but it needs less love and more labor.

NextStep

NextStep is another 32-bit OS which doesn't include source code. Since I went through the hassle of attempting

to install, I thought I'd warn the unwary who might be suckered in by the OOH (Object-Oriented Hype). The first thing to know is that NextStep supports only two SCSI host adapters: the Adaptec 154x, and a DPT EISA caching host adapter. You must have one or the other. And get this: You're only allowed to have a single 1.44 megabyte floppy attached. You're not permitted to have a second floppy!

I was going to install version 3.2, but its installation procedure crashed every time at the point of trying to talk to the Adaptec board. Version 3.1 hung every time, usually at the beginning, sometimes a little further on. The manual says: "You may need to rerun the installation several times before it works." Can you believe that they print such an admission of incompetence right in the manual?

Instead of requiring you to do a mysterious Disklabel, NextStep always asks you questions along the line of "Do you want to install NextStep? Press 1 for yes, 2 for no." You get asked this question several times. The manuals, even the installation manuals, while full of beautiful screen shots, are almost devoid of any technical information. Next has a fax-back system for anyone who needs to know anything technical. They have an email system too, but you have to have NextStep installed to use it. Like most software companies these days, they picture information exchange as only flowing in the outward direction.

I never did get NextStep to work. Evidently its marketplace obscurity is well-deserved.

After this installment, I'm not going to review any more operating systems, for a while at least. I recommend only three operating systems for PCs: Linux (Slackware or Yggdrasil), Windows NT, and OS/2 Warp. I'll probably recommend Minix 1.7 when the release comes out.

Peer-to-peer networking under DOS

In version 6 of MS-DOS, Microsoft quietly slipped in a two-station LAN. It consists of two EXE files, which are also device drivers, INTERLNK.EXE and INTERSVR.EXE. Connections can be made via serial or parallel ports. You put DEVICE=INTERLNK.EXE in the CONFIG.SYS files of both machines, then run the command INTERSVR on the server and INTERLNK E:=C: (for example) on the client. Presto, the client can access the server's drive.

This got me to thinking about the small-machine network. File transfer is not really very convenient. Yet drive mapping is complicated and very OS-specific, involving interception of file-level OS calls. But what if we trapped BIOS calls instead? There's only a couple of calls that would need to be intercepted, and performance would actually not be much worse. The problems would be related to file sharing, directory entries, and block allocation (allocation vector bits). Yet, in the CP/M case at least, these involve data which is accessible to the BIOS. Furthermore, in a small-machine network, we can restrict who has write access, giving all but one user read-only access if desired.

I was thinking about this in the context of the wonderful CP/M CD-ROM from Walnut Creek. Right now, getting files from it to a CP/M system is cumbersome. There ought to be a way that I could map a group of files, let's say a subdirectory, to a virtual CP/M drive (which is limited to only 8 megabytes), and access it through a serial port as though it were a hard drive.

A small LAN or OS simply can't afford the space to do it the way the big guys do. We've got to cut corners and get away from the layers which, while making big systems more reliable, reduce efficiency and increase code size.

The Intronics Pocket Programmer

Just a brief mention of this product, which works really well. This is an EPROM programmer which attaches to a PC's parallel port - no board required - and costs only $130. It has a ribbon cable and a YAWT (Yet Another Wall Transformer), and a diskette containing the software. It'll program any single-supply EPROM; you can get adapters for MCUs, too, but they cost almost as much as the programmer itself. Anyway, this is a great product. I was afraid it'd be slow, going through the parallel port, but it's just as fast as my old programmer, and a lot more convenient. Now I can easily take the programmer and my laptop to the equipment and copy or reburn PROMs right there. If you're in the market for a PROM programmer, think hard about it.

Next time

Next time we'll move from the Mall to the small. I'll show you how to use Small-C to generate a firmware program to run from PROM. Plus, we'll have some words about the joys of computer telephony.

For more information

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

Walnut Creek CD-ROM
(Manufacturer - FreeBSD 2.0 and CP/M CD-ROM)
1547 Palos Verdes Mall Suite 260,
Walnut Creek CA 94596
+1 510 676 0783

Next Computer, Inc.
(Manufacturer - NextStep)
900 Chesapeake Drive, Redwood City CA 94063

Intronics, Inc.
(Manufacturer - Pocket Programmer)
P.O. Box 13723
Edwardsville KS 66113
+1 913 422 2094

Mr. Kaypro

By Charles B. Stafford

Regular Feature

Kaypro Support

Composite Video

BARRY COLE ROMs

Since we last conversed, in issue 74, (mea cullpa, mea culpa) there has been a great deal of excitement. I actually got to meet and talk with Barry Cole. For those of you to whom this doesn't mean anything, don't feel bad! Up until about a year ago I didn't know anything about Barry either.

Back in 1983, the dark ages, when the '84 series Kaypros were but a gleam in Andy Kay's eye. Barry was repairing about 20 K-IIs and K-4s a week and modifying quite a few of them. When the '84s came out he decided that since they had a larger monitor rom and the appropriate signals were close by, that booting from disk was for the birds and other computers. Reality was just a few short days (weeks) away with the birth of the Barry Cole Rom. He used a 27128 (16k), twice the size of the stock 2764 (8k), 4 times the size of the K-4s 2732 and 8 times the size of the 2716 in the original design (Kaycomps and K-IIs). The code included not only the bios (basic input/output system) and bdos (basic disk operating system) but also a CCP (console command processor) based on ZCPR 2. This meant that he was completely free of Digital Research's copyrights and patents, and the term "boot disk" became meaningless. In fact, according to Barry's promotional literature, a machine with his rom will boot with a piece of cardboard in the drive.

Barry has included a version of ZCPR, as well as bdos changes that will accomodate two 10meg hard drives or one 20meg hard drive. You'll still need to make the hardware changes, of course, but the support is there. The command set understood by the CCP has not only the original 6 embedded ones, (DIR, ERA, SAVE, USER, TYPE & REN) but also:

LIST (sends the file to the printer)
GO (re-executes a previously loaded program)
GET (loads a program at a specified address)
JUMP (executes a program at a specified address other than 0100h)
SAFE (parks a hard drive if installed)
DISK (a built-in copy, era etc utility)
D (a super directory clone)
BOOT (boots from a floppy in drive A)
NORM (initiates "protected" mode disabling most commands, for bbs or similar use) and
PASS (+ a password, undoes the effect of NORM).

The modifications also specifically report disk malfunctions and eliminate most resets (such as aborting a faulty disk write).

The bad news is that I haven't got the ROMs yet. The good news is that I expect to get them in the 1st quarter of 1996. Watch this space for notification.

ON TO MORE TANGIBLE STUFF

Herewith, the promised follow-up to the external composite video project we did in TCJ #74.

Our objective is to understand the project sufficiently to be able to modify it to fit all CP/M Kaypros, and use both composite and TTL monitors (DB-9 connections).

THEORY

The circuit actually consists of five stages; the input signals, the buffers, the mixer, a driver and the output signal(s). See figure 1.

In the last project we took the input signals as well as ground and 5V from the pins of U1, a 7406 that we removed and out in a wirewrap socket. This works very well for the '83 K-10s and all the '84 models, but not so well for the '83 K-IIs and K-IVs. These machines, however, have a row of test points, labeled E1-E6, just toward the front of the computer from the Centronics connector for the parallel port. They have all the signals we need with the exception that the vertical sync is inverted. If the solder in these test points were removed, (easily done with a small soldering iron and a "solder sucker" or wick), we could mount a six pin header, and use it to plug on a "daughter" board. To take care of the vertical sync the connections to pins 1 and 4 must be switched with the vertical sync signal from E4 going to pin 4 and pin 1 being tied "high". See Figure 2.

Since the '83 K-IIs & K-IVs use the same vertical and horizontal sync frequencies as standard composite (& close to TTL Monochrome, by the way) monitors, no tweaking or "monkeying around" is needed. We can now use this circuit in any CP/M Kaypro.

Our next trick is to adapt this circuit to TTL monitors using a DB-9 output connector so we can use "standard" external monitors. The advantage of this is that an EGA monitor will "sync"

right up to an '84 K-anything or an '83 K-10 without any tweaking, and a CGA monitor will "sync" right up to an '83 K-II or K-IV without tweaking. A monochrome monitor, though, may need a "Sync" adjustment, but the stock K-II or K-IV frequency should be within the monitor's range.

The 74ls00 that was used in the last circuit, has 4 inverting "nand" gates. Buffering a signal, and getting it right side up, requires the use of two gates. Since there are three signals involved (Horizontal sync, Vertical sync and video) six gates would be necessary, two more than are available. If, however, a 74ls86 is used, our problem is solved. A 74ls86 is a four section exclusive "or" gate, which, with one input tied "low" (to ground), becomes a non-inverting buffer. This means that signals can be picked up from either the 7406 (in the case of an '84 machine or an '83 K-10) or E1-E6 (in the case of the "83 K-IIs or K-IVs), run through one section each of the 74ls86 and then to the appropriate pin of a female DB-9 connector. Use of a jumper on the vertical buffer to select inverting or non-inverting provides for those monitors that expect inverted vertical sync. See Figure 3.

Which signal goes on which pin depends on the monitor the circuit is intended for.

Pin	Mono	CGA	EGA
1	gnd	gnd	gnd
2	gnd	gnd	open
3	-	open	open
4	-	open	open
5	-	open	open
6	open	video	open
7	video	open	video
8	Hsync	Hsync	Hsync
9	-Vsync	-Vsync	Vsync

Mixing and Matching choices of the input signal methods, buffering methods and output signal schema will result in an easily constructed circuit customized just for your machine, and will probably be the only one like in existance for miles around.

EXECUTION (or CONSTRUCTION, if you prefer)

After you have made the appropriate choices, and drawn or copied the a schematic diagram, a parts list can be compiled. Almost all the resulting parts lists will include a couple of 100 ohm resistors, an IC (74ls00 or 74ls86), at least one socket perhaps two, an output connector, some ribbon cable or coax, a six pin header and female connector or wire wrap socket, and a prototype printed circuit board to build it on. The same sort of prototype board that was used in the first project will work here as well.

Connector mounting can be a challenge. On the first project, a panel mount RCA socket was used. It was the type that is supposed to go into a hole from the front of the panel and have a nut put on from the rear. Flats were filed on two sides of the connector, so that it just fit in one of the ventilation slots on the back of the case. The slot could have been enlarged, but that would have involved cutting on the case and this solution seemed cleaner. Mounting a DB-9 female connector is a little more involved. If a slot is filed just a little wider, the DB-9 will fit just fine. A hole can be drilled just below the slot to accomodate one of the mounting screws, and a small piece of scrap metal can be fashioned and drilled to accomodate the other. A simpler solution is to find a standard "IBM" type slot cover which has been punched for a DB-9 connector, cut the ends off 3/8" outside the mounting holes, drill two more 3/16"holes (one at each end) and mount this adapter over a ventilation slot using screws and nuts with washers to bridge the slot on the inside.

CUTTING THE CASE

If you decide to file, drill or cut the case, the safest way is to remove all the electronics, modify the case, clean it of all chips and filings and then reinstall the electronics.

There is, however, an easier alternative which, if done carefully, is equally as effective. Using Duct Tape (two inch or three inch width), fasten two or more strips edge to edge about six inches long with a 1/4" overlap at the edges, sticky side up, on the workbench. This should result in a single piece about six inches by four or six inches. Stick one end on the inside of the case, parallel to the bottom of the case about an inch above the area to be cut, filed or drilled and the other end, also parallel to the bottom of the case about one inch below the area to be cut, filed or drilled. Now pinch the sides together, forming a tent over the work site, on the inside of the case. Cut, file or drill to your hearts content, and when you are satisfied with the result, smash the tent down on the inside of the case to pick up all the stray metal chips inside, and remove it carefully. Then inspect the area to insure that you didn't let any chips escape.

This procedure sounds complicated, and takes a lot longer to describe than do, but read it first and you'll see how simple it really is.

NEXT PROJECT

An internal modem (you pick the speed) for ALL the CP/M Kaypros.

A LITTLE COMMERCIAL

Recently I have been fortunate enough to find a limited supply of Kaypro monitors which include both the CRT and the Video board, as well as, mother boards (both '83 and '84 varieties) and a few WD1002-05 hard drive controllers. Monitors are $50.00 but the mother boards vary depending upon condition.

Figure 1.

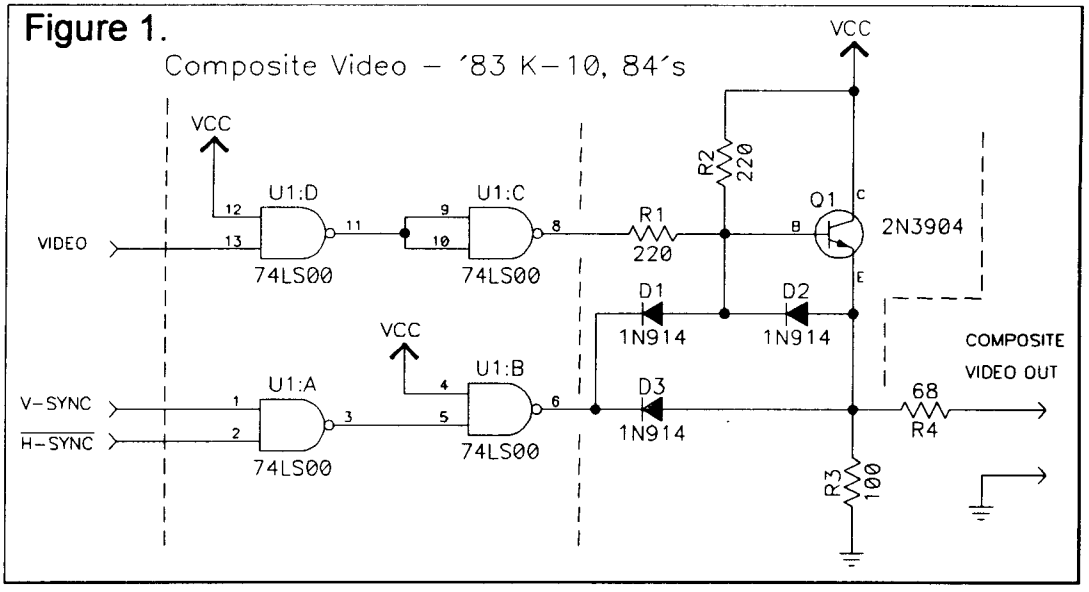


Figure 2.

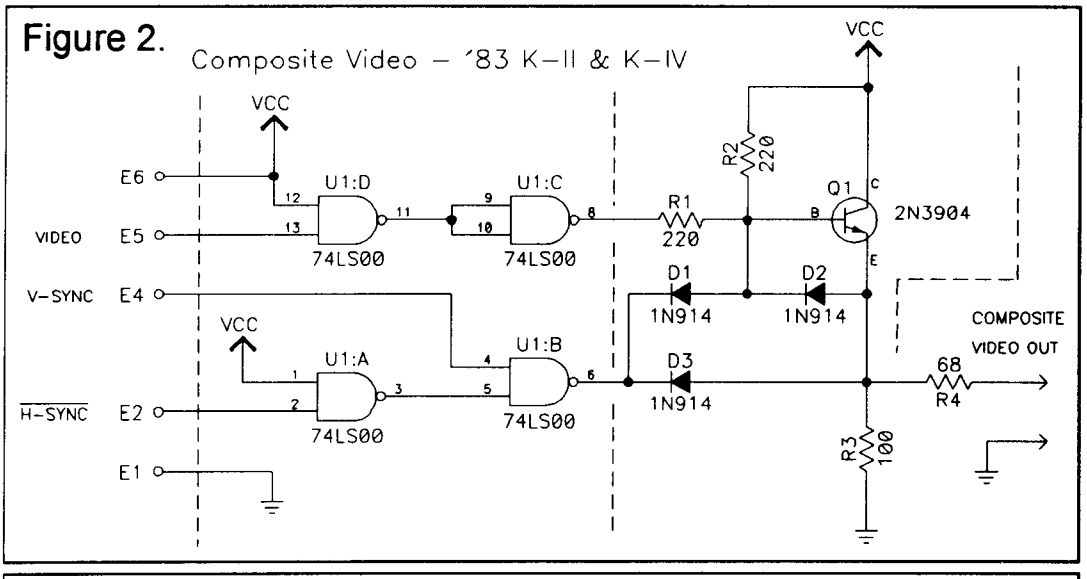
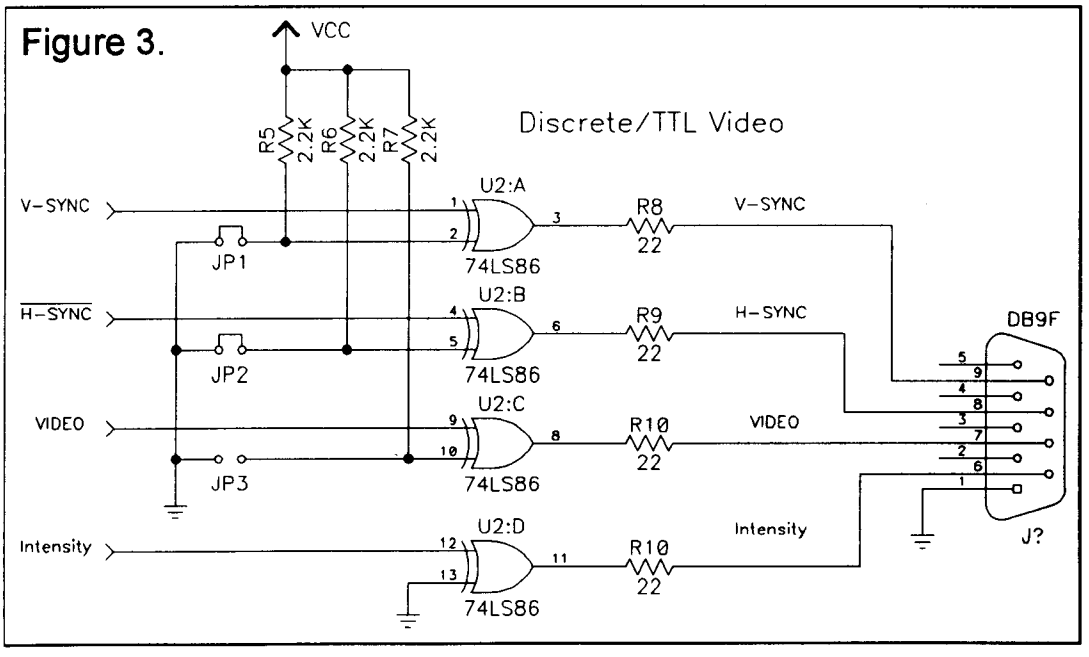


Figure 3.



The European Beat

by Helmut Jungkunz

Regular Feature

All Users

East German Z80 + GIDE

Finally - Hi-Tech for the looneys !

Well, well, if history doesn't go strange ways! I used to dream with my friends of the wonderful things happening in the US and we used to feel pretty lousy about our impossible situation: no RS232C on our computers (by default), very little or poor local programming, and very expensive hardware, by far higher priced than the equivalent in "Amerika".

Many of us were also suffering from being unable to understand enough English and none of the documentation and source code were available in German, French, Spanish at all.

Then, after all these years, we see the American standards drop next to zero almost overnight! What a shock. Here is, where our typical European "being behind" finally works out: the continuing industrial use of Z80 based equipment has brought up some new products, like Tilmann Reh's CPU280.

A small board that (besides it's outstanding, innovative design) is based on a very common industrial bus system, the "ECB-Bus". This allowed people to develop peripheral devices that would also be usable in other machines. This concept was probably the reason why Tilmann developed not only a nifty Hercules terminal and an IDE interface to make AT-BUS (IDE) hard disks available to CP/M users, but he also decided to take up the concept of a Generic IDE interface (GIDE) from an idea of a couple people. So Tilmann sat down and did the work!

The results are overwhelming:
A small board plugs into the (Z80) CPU socket of any Z80 CP/M computer, where the Z80 itself is then sitting on top of the GIDE, in the extension socket present there. At the Trenton Computer Festival '95 I was able to make a (poor, but successful) demonstration of how to mount the GIDE in a KAYPRO 4/10, a computer I, so far, only knew from books. The failing point was a defective hard drive I had purchased at the Trenton flea market. Still, the ease of the procedure could be shown.

Despite the bad condition of the drive, I could make all connections and use the test program once(!). It showed the drive identification (an IDE standard) and did some efforts to read randomly selected sectors. Then the drive died.

I was very upset at that day, having traveled all the way from Germany only to be ripped by some cat in the flea-market, since it was impossible for me to bring along an IDE drive, and neither I nor Jay were able to arrange for one for the demo.

I got back to Tilmann after my return and told him, that his interface was basically a great success. I got people from our AMSTRAD user group (SCUG) interested and also the people from the JOYCE USER AG, a German PCW group. The thing is, besides the easy hardware side, implementing a hard drive into the BIOS is a different story. We still don't have an implementation for the CPC. What a shame! But, at the same time, I want to report that the implementation by the JOYCE USER AG people was already successful.

I planned to show the PCW GIDE at our club's 10th anniversary. Unfortunately, the drive kit arrived late. The only good thing was, that a few days afterwards, our regular club meeting took place. I then arranged to have the important people there and I brought my PCW along.

After a test run of the PCW showed that it was working, we went to disassemble the machine. It turned out, that some of the screws were much longer than expected. Then we carefully marked all the wires and took off the sheet metal screen that is used as a high-frequency shielding. Then we slid out the Main board (right next to the drive(s)) and removed the Z80 CPU. The kit we had received from Reiner Seitz contained all of the cables and connectors necessary. The only thing one has to supply is the power. No problem, if you only have one drive, simply use the second drives power supply.

But be warned !!!!! The PCW drive supply connector looks identical to the standard 3.5 inch connectors - but are reversed between 5 and 12 Volts!!!!

Anyway, once the Z80 was plugged into the GIDE, we immediately fired up the PCW (still without the hard disk connected), to see if it was all working. It came up faultlessly, so we became more audacious and hooked up the IDE drive. Then we used the special startup disk supplied by Reiner Seitz. It uses a more up-to-date BIOS version (1.8) that allows the use of special loader files, a similar technique like the IMP overlays or such. The big advantage with the PCW's CP/M Plus here is that the banked system can be hidden and overlaid if enough memory is available. Thus, the PCW's TPA stays practically unaltered. This is fantastic, since all the programs run in no time at all and behave well. One has to know, there is lots of rubbish about that uses absolute addresses in the PCW system memory.

Naturally, we used the HDFORM tool from Reiner to see what it did. It performed a flashy clean-up of the hard disks directory to allow proper CP/M Plus data entry. Then I wanted to see more and copied an existing installation of Z3PLUS onto the hard disk. What shall I say, it ran "out of the box" in no time at all. The drive in the PCW is implemented in a superfast environment. It's performance may well be compared to 486 speed. Amazing!

I was so impressed, I was smashed to bits. (8)

Then, to top this, a week later, Joerg Linder calls me and tells me, the KC user group had decided to throw their "MicroDOS", a cheap and dirty CP/M clone with added features, overboard, in order to achieve true CP/M capabilities. We discussed different alternatives and I had sent him a copy of Hal Bower's CHEAPLAN TALK, that I had filmed on location at Trenton in April '95. So he agreed, that ZSDOS might be a good choice. I arranged for a package deal, and it seems, the KC85/4 (remember TCJ back issue #75) will be the first European Computer to use ZSDOS by standard! Imagine, over 50 people will be using ZSDOS! Then, I also recommended that they implement ZCPR34 and got another package agreement with Jay Sage for the lot.

They had been fiddling about with the garbled operating system of the KC85/4 in order to not only resource it, but to finally separate the mixed codes for BIOS, BDOS and part of the CCP, that by then couldn't be overwritten! When they were about to give up, Mario Leubner, the chief assembler programmer of the KC User Group, found a few clues that allowed him to rewrite large parts and rethread, what was going on.

So, to my amazement (again), I got a message from Joerg, announcing not only the successful implementation of ZSDOS, but also a new CCP that could be overwritten and - a fully implemented GIDE drive! They also rewrote the RAM access and the CPU sharing, which makes the new KC85/4 practically a new computer. I am pleased to report all these happy events, that took place in the same period when Bill Gates tried to sell Windows 95 to the Germans. Do I have to say, his success was poor here?

With all these happy thoughts in mind, I want to wish you all a successful and happy 1996! (no, it is never too late, I just proved it above!)

Regards,
Helmut Jungkunz
ZNODE 51

P.S: I filmed the KC85/4 presentation at Z-Fest 95, some of our 10th SCUG anniversary and also the PCW GIDE test on Hi-8. I copied this to VHS NTSC and sent a copy to Don Maslin and Michael Crafton. I don't know if anyone of them can duplicate VHS cassettes in decent quality, but if so, they are welcome to do so.

Special Feature

Intermediate Users

PLD Application

Hands on with PLD's

by Robert Brown

INTRODUCTION

The introduction of programmable logic devices (PLDs) was a great boon to the field of digital hardware design. The second generation PLD, the GAL (which stands for Generic Array Logic, a trademark of Lattice Semiconductor) is particularly suited for the small scale hardware designer. GALs offer the following benefits to the hardware designer:

Flexibility - GALs are very flexible devices, they can implement both combinatorial logic functions (AND, OR, NAND etc.) and registered logic functions (counters, shift registers etc.) on the same chip.

PAL replacement- The GAL16V8 and GAL20V8 each can directly replace over 20 of the common PAL (Programmable Array Logic - the first generation PLD) types each. This means you only need to stock 2 GAL types to handle your PLD needs.

Space savings - In my experience each GAL has typically replaced between 2 and 4 standard TTL chips, saving a large amount of board space.

Speed - GALs are fast devices with propagation delay down as low as 7 ns. Typical GALs have a propagation delay of only 15 ns - faster than standard 7400 or 74LS series logic.

Reprogrammability - Not only are GALs programmable giving the ability to correct design errors and make board layout easier, they can be reprogrammed up to 100 times. Erasing and programming takes only a few seconds.

Cost - In addition to the savings in PC board real estate, standard speed GAL16V8s and GAL20V8s (25 and 15 ns) cost only a few dollars even in small quantities.

There are several varieties of GALs but I will limit this article to the GAL16V8 and GAL20V8. They are easy to design with and are the least expensive and most readily available GAL devices. Rather then get bogged down with the internal details of the devices, we'll cover what is needed to use these PLDs in your designs and then look at a real life design example.

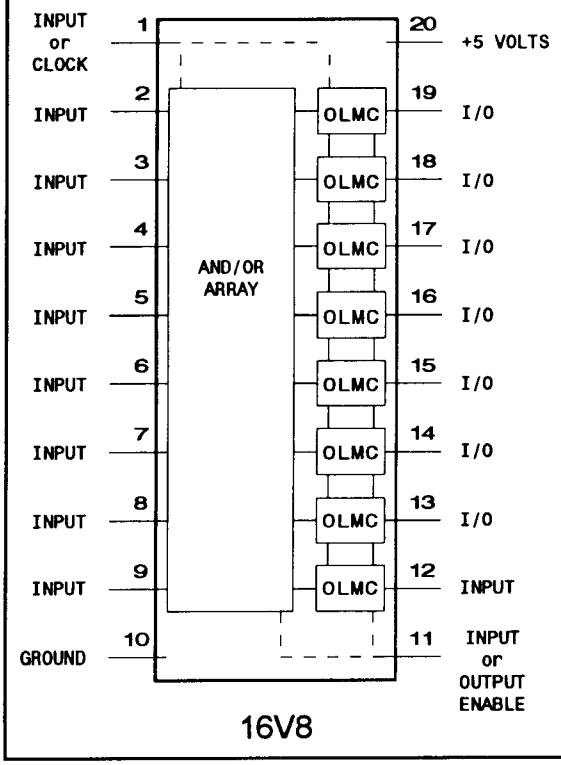
The Device Architecture

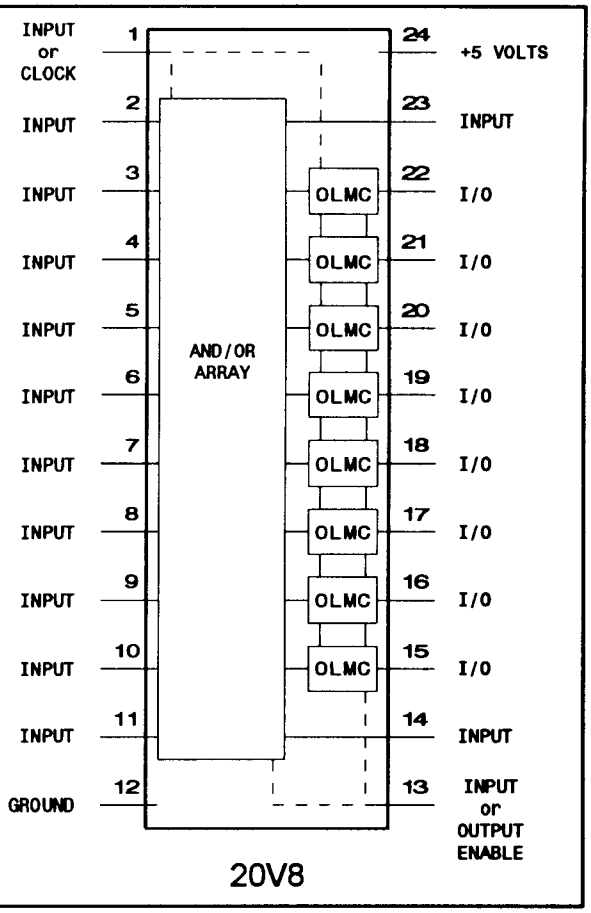
The GAL16V8 is commonly packaged in a standard 20 pin DIP and the GAL20V8 is commonly packaged in a 24 pin skinny DIP (a 24 pin skinny DIP is 0.3 inches wide, the same width as a 20 pin DIP and half the width of a standard 24 pin DIP). The pinout for both devices is shown in figure-1. For the GAL16V8 pin 10 is the ground pin and pin 20 the +5 volt pin (VCC). Pins 12 through 19 are each connected to Output Logic Macro Cells (OLMC). The OLMC allows these pins to act as inputs, combinatorial outputs, registered outputs and input/output pins. Pins 2 through 9 are always general purpose input pins. If any of the OLMC are configured as registered outputs then pin 1 is a Clock input and pin 11 is the Output Enable for the registered outputs. If none of the OLMC are registered then pins 1 and 11 are general purpose inputs. Internal to the chip is an array of and/or logic that is configured with each OLMC when the chip is programmed. The 20V8 has a similar design, the main difference from the 16V8 is the four additional input pins.

The Design Tools

In addition to your PC you will need only three tools to do design work with PLDs, a text editor, a logic compiler and a device programmer. A logic compiler is a program that translates a high level design file, in which the relationship between inputs and outputs is expressed in the form of equations, to a low level file device specific file for the programmer. The low level file used by the programmer is called a JEDEC file and is sometimes referred to as a 'fuse map'. (Earlier PLDs were programmed by literally blowing up fuses internal to the device leaving only the desired connections - of course they could not be reprogrammed - you threw away your mistakes.) National Semiconductor used to offer a FREE logic compiler before they got out of the PLD business. You can still get a copy of their PLAN or OPAL logic compilers off of many BBSes (including TCJ's). For the 16V8 and 20V8 you can use either PLAN or OPAL. The high level design file for PLAN is called an equation file and uses the extension .EQN, the output JEDEC file uses the extension '.JED'. Since PLAN is available to everyone I will use it in the examples, the concepts however are universal, not specific to PLAN. The equation file is a standard ASCII text file and can be produced using any text editor.

Figure 1.
GAL16V8 and GAL 20V8 Architecture





Designing with Equations

If you normally design with standard TTL devices, shifting to design using GALs might take a slight adjustment. However the underlying concepts are the same. In the equations a + is used to represent OR, a * to represent AND a / for NOT or inversion. In figure 2a show the equation represent by an AND gate. Each group of signals ANDed together is referred to as a product term. Figure 2b shows the equivalent representation for a two input OR gate. In figure 2c a more complicated piece of logic is represented, it includes the use of the / symbol to show inversion. Notice how the equation is organized. The equations are written in a sum of products format, a useful convention is to list each product term on a separate line. The inversion can also occur on the output as shown is in figure 2d. In the 16V8 and 20V8 up to eight outputs can be defined in this way (each of the eight OLMC). The inputs for the equations can come from any of the input or output pins either normal or inverted. A maximum of seven or eight product terms are allowed for each OLMC, this depends on the exact configuration of the OLMC. Given this, it is obvious that a single GAL can replace several packages of AND, NAND, NOR and OR gates. But this is only the start.

So far all the examples have used combinatorial logic, in addition GALs can also handle sequential logic such as shift registers and counters. To do this the OLMC is configured as a register (D flip flop). If any of OLMC are configured as registered then pin 1 is the clock input to the register. Where as an = symbol is used to show a combinatorial output in an equation, a := symbol is used to show a registered equation output. For example:

```
Q :=   D
```

The := indicates that the output Q is registered. This means that Q will take on the value of D following the rising edge of the clock on pin 1. Two or more outputs can be combined to form counters and shift registers as shown in figure 3. In this case it shows a two bit counter with a terminal count. The two outputs Q0 and Q1 will count from 0 to 3 continuously and the terminal count indication will be active when the count is at its maximum value of 3.

The registered outputs have a common output enable at pin 11 on a 16V8 and pin 13 on a 20V8. When output enable is low the registered outputs are all enabled. If output enable goes high all the registered outputs will be disabled (tri-stated). Even when the outputs are tri-stated the register outputs are still available internally as feedbacks (so the counter would continue to work even if the outputs were disabled). Combinatorial outputs can each have an output enable defined, this is limited to a single product term. For example:

```
X.OE = B * C
```

This would indicate that output X should be enabled when B AND C are high.

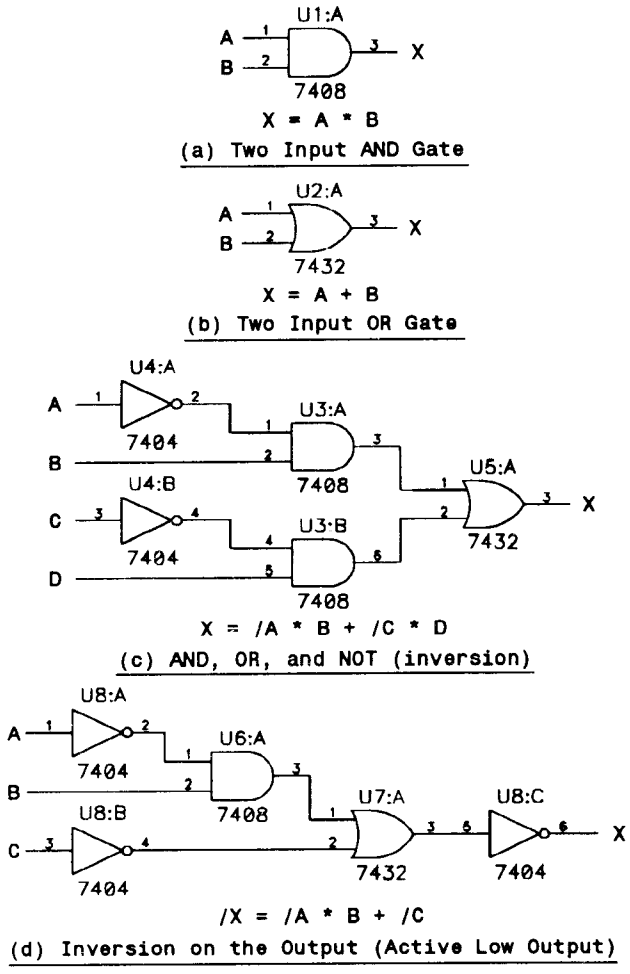


FIGURE 2 - COMBINATORIAL LOGIC EQUATIONS

A Real Design Example

To illustrate the use of GALs in a real design, I will use the main board from a high speed, low cost 16 channel logic analyzer. The logic analyzer main board uses a total of 17 ICs, of these 2 are static RAMs, 4 are octal latches, 4 are octal buffers and the remaining 7 are programmed GAL16V8s. All standard logic was handled by the 7 Gals, they replace about 20 high speed TTL ICs and make the logic analyzer buildable. (For more information on the logic analyzer design download RGBLOGIC.ZIP from the TCJ BBS, the schematic and all of the EQN files are included in the file, so you can review the complete design.)

Central to the logic analyzer is the clock selection GAL. This allows the analyzer clock to be selected from 5 different internal clocks, an external clock or a software controlled clock. The PLAN equation file, CLOCK.EQN is shown in figure 4. The lines that start with a ; are comment lines, they are ignored by the compiler and are used for documentation. The equation file must contain two sections, the declaration section and the equation section. The declaration section should appear first and is indicated by the keyword CHIP. The line:

```
CHIP clock 16V8
```

begins the declaration block, it gives the chip a name (clock) and selects the device type (16V8). The next part of the declaration block is optional, but will be needed in most cases. In this section we assign symbolic names to each pin on the chip, as follows:

```
clk nc ext self s2 s1 s0 dir wr gnd
/oe sysclk1 wrdat sysclk0 q4 q3 q2 q1 q0 vcc
```

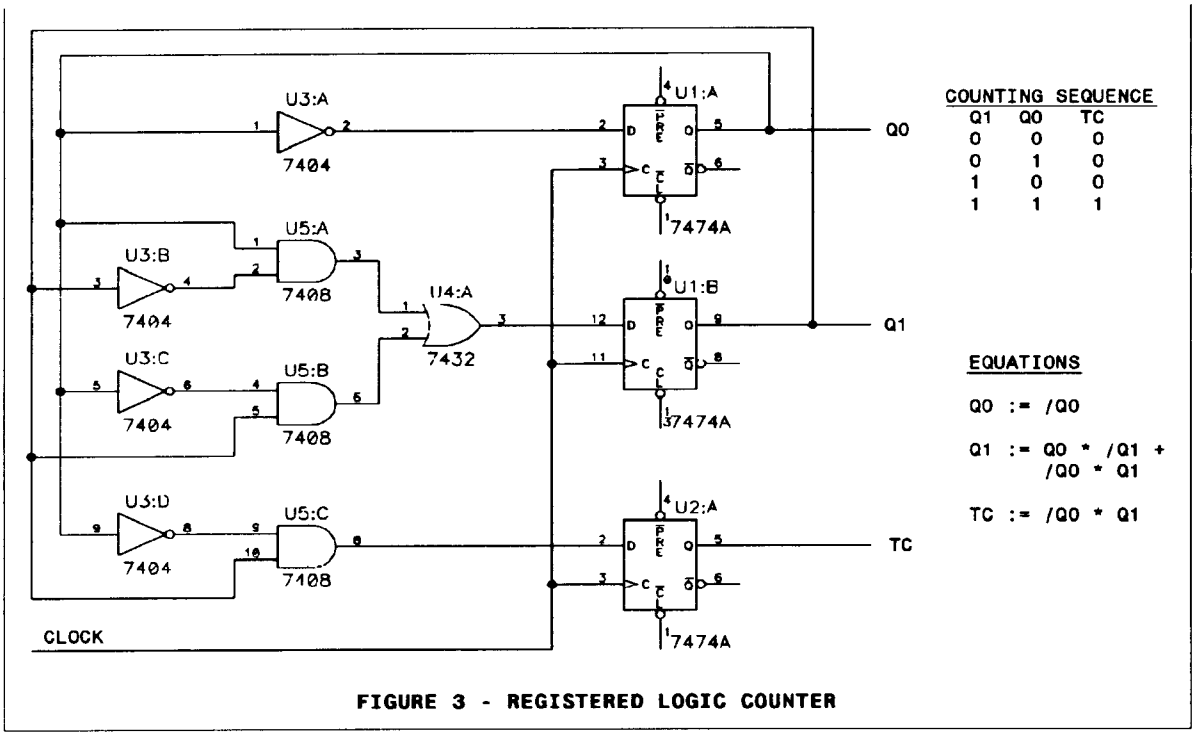


FIGURE 3 - REGISTERED LOGIC COUNTER

The symbolic names start with pin 1 and are assigned in order through pin 20. In this case pin 1 is assigned the name clk, pin 9 the name wr, pin 12 the name sysclk1 and pin 19 the name q0. If we later need to change the pin assignments, we simply rearranged the names given here.

The equation section of the file is indicated by a line with the word EQUATIONS. Outputs q0, q1, q2, q3 and q4 form a 5 bit counter (a straight forward expansion of the 2 bit counter used before). This counter provides several reference frequencies at the outputs, q1 is half of q0, q2 is half of q1 etc. With a crystal oscillator of 40 Mhz connected to pin 1, q0 provides a 20 Mhz clock, q1 a 10 Mhz clock, q2 a 5 Mhz clock, q3 a 2.5 Mhz clock and q4 a 1.25 Mhz clock. The outputs sysclk0 and sysclk1 use identical equations, so I refer to them together as sysclk. The sysclk outputs are combinatorial, they allow the system clock to be selected from q0, q1, q2, q3, q4 or the inputs self or ext. The inputs s0, s1 and s2 select which clock is output to sysclk. From the equations you can see that if s0, s1 and s2 are all 0 then sysclk is the same as the input self. If s0, s1 and s2 are all 1 then the last product term will apply and sysclk will follow q0. The final output defined in the equations is wrdat, this is obviously just a simple two input OR function. I remember that Bill Kibler said he wanted to see the equivalent circuit in standard logic for all programmable logic used in TCJ. It is not always possible, but in this case I had first looked at designing the logic analyzer using standard logic. The equivalent circuit to this GAL is shown in the schematic in figure 5, notice the savings in chip count and cost.

The equation file is compiled with the command line:

```
EQN2JED -N CLOCK
```

EQN2JED is the PLAN program that produces the JEDEC file, this will take our CLOCK.EQN file check it for errors and if error free produce the JEDEC file CLOCK.JED. The -n in the command line tells the program to produce a new log file, rather then append to the existing log file. The log file produced is CLOCK.LOG. The log file has a lot of interesting information about the programmed device the use of each device pin, the product term usage and the device pinout.

A Memory Decoder

Let's take a look at a real life example that was suggested to me by Dave Baldwin. Let's say that we are designing a Z80 based system and we would like the memory map to include an 8K EPROM at address 0000h, a 2K EEPROM at address 2000h and have the remainder of the 64K address space filled with static ram (2 32K devices). So a table of our memory map looks like:

Address (hex)	Device
0000-1FFF	EPROM
2000-27FF	EEPROM
2800-7FFF	SRAM0
8000-FFFF	SRAM1

This type of decoding is a pain in the butt if we use standard TTL devices. We would need several devices and have to deal with the problems of propogation delays through the several levels of devices. If we use a 16V8 or 20V8 we can easily handle this decoding with one 20 or 24 pin device. Our total propogation delay will be the single propogation delay of the chip. This means we can have a propogation delay as low as 5 ns with a GAL rated at 5 ns. In addition we can include the decoding of the I/O space on the same chip.

Figure 4.

```
; GCLK.EQN Logic Analyzer U15
; This is the declaration section
CHIP GCLK 16V8

; Pin labels here
CLK NC EXT SELF S2 S1 S0 DIR WR GND
OEN SC1 WRDAT SC0 Q4 Q3 Q2 Q1 Q0 VCC

; Next is the equation section
EQUATIONS

; Q0 - Q4 form a 5 bit binary counter

Q0 :=   /Q0

Q1 :=   Q0 * /Q1 +
    /Q0 * Q1

Q2 :=   Q0 * Q1 * /Q2 +
    Q2 * /Q0 +
    Q2 * /Q1

Q3 :=   Q0 * Q1 * Q2 * /Q3 +
    Q3 * /Q0 +
    Q3 * /Q1 +
    Q3 * /Q2

Q4 :=   Q0 * Q1 * Q2 * Q3 * /Q4 +
    Q4 * /Q0 +
    Q4 * /Q1 +
    Q4 * /Q2 +
    Q4 * /Q3

; SC0 and SC1 output the system clock as selected
by
; S0, S1 and S2

SC0 =   /S0 * /S1 * /S2 * SELF +
    S0 * /S1 * /S2 * EXT +
    S0 * S1 * /S2 * Q4 +
    /S0 * /S1 * S2 * Q3 +
    S0 * /S1 * S2 * Q2 +
    /S0 * S1 * S2 * Q1 +
    S0 * S1 * S2 * Q0

SC1 =   /S0 * /S1 * /S2 * SELF +
    S0 * /S1 * /S2 * EXT +
    S0 * S1 * /S2 * Q4 +
    /S0 * /S1 * S2 * Q3 +
    S0 * /S1 * S2 * Q2 +
    /S0 * S1 * S2 * Q1 +
    S0 * S1 * S2 * Q0

; WRDAT is a simple OR function of WR and DIR

WRDAT = WR +
    DIR
```

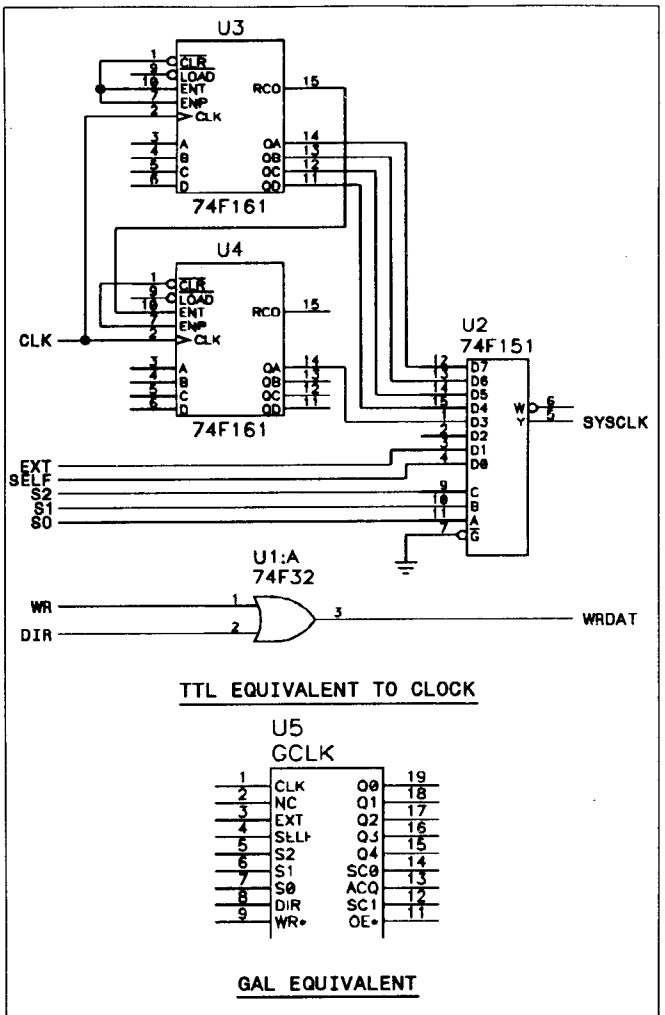


FIGURE 5 - TTL and GAL CIRCUIT COMPARISON

In this case defining the equations for each chip select output is very straight forward. We want the EPROM chip select to be active when address lines A15, A14 and A13 are low and the Z80 MREQ signal is low. Since the active state of the EPROM chip select is low we would express the equation as:

```
/EPROM = /A15 * /A14 * /A13 * /MREQ
```

The EEPROM chip select is only slightly more involved. The address range 2000-27ff is selected when A15, A14, A12, A11 and MREQ are low, while A13 is high. This gives the equation:

```
/EEPROM = /A15 * /A14 * A13 * /A12 * /A11 * /MREQ
```

The first static ram has the most complicated equation (but still well within the capabilities of the 16V8). The 2800-7FFF address space can be thought of as three regions 2800-2FFF, 3000-3FFF and 4000-7FFF. The product terms defining each of these regions is then ORed together to define the complete equation as follows:

```
/SRAM0 = /A15 * /A14 * A13 * /A12 * A11 * /MREQ +
/A15 * /A14 * A13 * A12 * /MREQ +
/A15 * A14 * /MREQ
```

It is obvious that the equation for the second static RAM's chip select is:

```
/SRAM1 = A15 * /MREQ
```

The complete equation file with the I/O decoding using a 16V8 is printed above. If we wanted finer granularity on the I/O decode we could use a 20V8, this would give us 4 addition inputs for address lines, that could be included in the equations. If we wanted to have a 10 chip select outputs we could use a GAL22V10. Note that the pin out selected is arbitrary, in this case we could swap any of the input pins or any of the output pins just by redefining the pin list. This is a great aid if you do your own PC board design.

I think this example shows you why I use programmable logic wherever I can. We have reduced our decode logic to one 20 pin device. In doing so we have reduced the number of interconnects, saved PC board space, saved money and saved both circuit and PC board design time. In addition we have increased our design flexibility and helped our parts inventory. If we later find that we must have a 16K EPROM instead of the 8K EPROM, we can change our decode circuit by just reprogramming the GAL16V8. We can also replace our bin of spare TTL chips with few 16V8s. Once you start using programmable logic you can see the advantages continue to pile up.

```
; Z80 memory and I/O decoder example - TCJ
; R. G. Brown - ALTA ENGINEERING (860) 489-8003
; Memory Map
; 0000-1FFF EPROM
; 2000-27FF EEPROM
; 2800-7FFF SRAM0
; 8000-FFFF SRAM1
;
CHIP decode  16V8

; set pinout - can be altered later if needed
MREQ A15 A14 A13 A12 A11 IOREQ A7 A6 GND
A5 EPROM EEPROM SRAM0 SRAM1 IO0 IO1 IO2 IO3 VCC

EQUATIONS

/EPROM = /A15 * /A14 * /A13 * /MREQ

/EEPROM = /A15 * /A14 * A13 * /A12 * /A11 * /MREQ

/SRAM0 = /A15 * /A14 * A13 * /A12 * A11 * /MREQ +
    /A15 * /A14 * A13 * A12 * /MREQ +
    /A15 * A14 * /MREQ

/SRAM1 = A15 * /MREQ

/IO0 = /A7 * /A6 * /A5 * /IOREQ

/IO1 = /A7 * /A6 * A5 * /IOREQ

/IO2 = /A7 * A6 * /A5 * /IOREQ

/IO3 = /A7 * A6 * A5 * /IOREQ
```

Circuit Layout

When using PLDs you must use the same care in circuit layout as you would need to with any high speed logic device. Completely covering the topic of high speed circuit design would fill a book, but here are some things to look for. Be careful with the ground and power layouts to reduce the impedance of these signals to the chip. Use a decoupling capacitor as close as possible to each chip's power and ground pins.

Device programmers

The device programmer will program the device with logic as defined in the JEDEC file. Several companies produce universal programmers in the $500 range. For getting started with programmable logic the lowest cost programmer that I know of is my ALTA ENGINEERING PLD programmer kit (check the file RGBPLD21.ZIP on the TCJ BBS for more info). Check the resource list for information on programmer companies.

If you would like to see more on Programmable Logic in TCJ, please let me (and Dave Baldwin) know what topics would be of greatest interest.

Robert G. Brown has authored many articles for other technical publications. His company, ALTA ENGINEERING specializes in the use of programmable devices to produce low cost, high quality electronic kits. He can be reached at (860) 489-8003 (Voice/FAX/FAX Back) or via EMAIL at 72477.2616@compuserve.com. You can visit the ALTA ENGINEERING web site at either: http://ourworld.compuserve.com/homepages/alta or http://www.gutbang/alta.

Resource list

Advin Systems Inc.
1050-L East Duane Ave.
Sunnyvale, CA 94086
(408) 243-7000

Alta Engineering
58 Cedar Lane
New Hartford, CT 06057-2905
(860) 489-8003
http://www.gutbang/alta

B&C Microsystems Inc.
750 North Pastoria Avenue
Sunnyvale, CA 94086
(408) 730-5511

Bytek Corporation
543 NorthWest 77th St.
Boca Raton, FL 33487
(407) 994-3520

Lattice Semiconductor Corporation
P.O. Box 2500
Portland, Oregon 97208
(503) 681-0118
(800) FAST GAL

Link Computer Graphics, Inc.
4 Sparrow Drive
Livingston, NJ 07039
(201) 994-6669

Logical Devices, Inc.
130 Capitol Drive
Golden, Colorado 80401
1-800-331-7766

System General Corporation
1603-A S. Main St.
Milpitas, CA 95035
(408) 263-6667

Xeltek
757 North Pastoria Avenue
Sunnydale, CA 94086
(408) 524-1929

Regular Feature

Intermediate

Memory Management

Dr. S-100

By Herb R. Johnson

Introduction

For this month's column, I'll discuss how the S-100 bus accesses and manages memory. This column will also be an exercise in how to read old schematics! Follow along with the schematics from your S-100 system or from back issues of TCJ. If you have my previous column, it will give you a brief description of the S-100 bus and its revision as the IEEE-696 bus. A technical description of the Intel 8080, 8085, or Zilog Z80 will also be helpful: the original S-100 bus is based on these processors.

Mow the Snow & other stuff

Well, nature has replaced the fall grass with early winter snow, so despite my hopes that I could stop mowing to work on S-100 stuff, I had to clear the driveway of snow! There seems to be no end to home maintenance! Speaking of home, please note my **change of address** - actually, just a change from my Princeton office box to my Ewing NJ home address, as a convenience to me. Hope your holidays were pleasant and that the new year is a good year. Santa got me some boots, a faster computer, and some new engineering work very nearby!

I'd like to thank Jeffrey Doerschler of Wethersfield CT for providing his **Cromemco system** to the pile o' computers. It's a Cromix system (Cromemco's 1985 answer to Unix) and a pile of docs. It will eventually make for a good TCJ column or two. And thanks to Keith Andress for contributing a **Big Board Z80 system**, some Fulcrum (IMSAI reincarnation) cards and docs, and various sets of electronics parts, in exchange for some 8-inch diskette file conversion. The "Dr." does take in some homeless computers from time to time, or at least I'll refer other people to them. For instance, any takers on the Big Board? Meanwhile, progress on the **GIDE (IDE hard drive to Z80 interface) project** is discussed in another article in this issue.

Help: Monitor Dynamics docs?

I've had a couple requests recently for documentation for a Monitor Dynamics hard disk controller. I have a few of these, and some software, but no manuals. Anyone out there with a copy?

Topic: memory and the S-100 bus

Memory management is no small issue, as the S-100 bus started out in 1975 when the biggest memory chips were 256 bytes (not kilobytes) by one bit, namely the 2101 chip. So the original Altair/IMSAI bus address space of 64 Kbytes was **plenty of room**. Of course, the original 8080 processor had only 16 address lines and could only address 2**16 address locations, namely 64K. Memory cards came in 1 kbyte sizes: then 4 KBytes, then 8 Kbytes (sixty-four 2102 chips at that!). But in just a few more years, larger memories were available, and bigger programs were written to fill them up. Then other processors appeared for the S-100 bus: the Motorola 68000, the Intel 8086 and 8088 most notably, processors that could address more than 64K locations AND processors that could read 16 bits at a time. So how did S-100 designers deal with these issues? And do it without a lot of expensive (at the time) chips and circuit board space?

The Old Days

The original S-100 bus was based on the signals and timing of the Intel 8080 microprocessor. For instance, lets look at the control circuits of a Morrow memory card, the **SuperRam 16K** of 1978. The short version of a memory read sequence is: set up the address, set up the memory read status, and strobe the data! Each operation corresponds to a bus cycle, as timed by the bus clock signal (phi)2; but in the old days the clock signal was not always used! And, the working practice of the day was to consider hardware gates as producing "high" or "low" values, and a gate was "active" when the inputs satisfied the logical conditions of the gate. So, we'll use that methodology here as well.

Memory read

To do a memory read, first the address lines **A0-A15** are enabled (**Figure 1**) with the proper address. This memory card uses exclusive OR gates to read each upper address line and to compare it to a switch setting for that line. If the state of the line (high or low) is different from the state of the switch, the gate is inactive and the output is high. Note that the four gates have their outputs tied together: these chips (74LS266) have what is called an "open collector" output which allows such a connection. But a common collector output can only pull a line **down**: if the gate is not active, a resistor must pull the line **up**. So for the select line to remain high, **all four outputs** must be high; so each address line must **not match** the state of its corresponding switch. So, a single four-gate TTL chip, and four switches and several resistors can select one of sixteen addresses.

The last four address lines, **A15-A12**, select one of 16 4-K banks of memory: this card has four sets of such circuits, for four bank of memory, for a total of 16K of memory. But this signal alone cannot select a memory board, because the address lines can be used anytime: for memory read or write, I/O read or write, interrupts, or even "between" bus activities!

To be read, the memory card must know the bus operation to be performed is a memory read. **Figure 2** shows the circuits that examine the bus status lines. For a memory read, **SMEMR** (memory read status) is active and high, **SWO*** (write status) is inactive and high, and **SOUT** (I/O write) is inactive and low (again, these are how the 8080 processor works!). So, the NOR gate which examines SWO* and SOUT is active, the output is low, and thus has no effect on the next NOR gate. But SMEMR is high, so the next NOR gate is active and low. The following NOR gate has an input from an inverter connected to the PHANTOM line. We will presume that input is low, so the NOR now has both inputs low: consequently, it's output is inactive and high for the period when SMEMR is active.

Figure 3 shows a NAND gate that combines the SELECT signal from one of the address NOR gate sets (when the correct address is on the bus) and the signal from the NOR gates just described (when memory read status is on the bus). When both these signals are active high, the output of the NAND gate is active low: this /BANK signal is used to select the memory bank (via the memory chip's chip select line, not shown).

At this point, we've satisfied two conditions: a good address, and a good bus status. So the memory card can determine this is a memory read, and that it has been selected. The memory chips can now take the time to access their data. But, to actually put the data from memory on the bus, we need the bus "read strobe" signal. **Figure 4** shows a NAND gate with bus signal **PDBIN** as one input, and a NAND gate with /BANK signal inputs as the other input. When any of the /BANK signals are active low, the corresponding NAND gate output is inactive and high - that's good! That becomes the input to the NAND gate with PDBIN. When PDBIN is active high, and the /BANK gate is inactive high, the NAND output is active low! The output is used to enable (**Figure 5**) the 74LS367 tristate buffer, which connects the data outputs from memory (MEM DO signals) to the data input lines of the bus, **DI7, DI6, DI5,** and **DI4**. Another chip performs similar service for the other four bus data input lines, **DI3, DI2, DI1** and **DI0**.

Memory Write

Whew! That completes the memory read cycle. Before we forget all this logic, we may as well run through the **memory write** cycle! In this case, the sequence of operations are: set up the address, set up the memory write status, and then strobe the data into the memory!

The address lines are set up as before, no change here. In **Figure 2**, this time the **SMEMR** signal is inactive low, the **/SWO** signal is active low, and the **SOUT** signal is inactive low. So the SWO signal makes the NOR output active low, the following NOR goes inactive high, and once again this high selects the bank (**Figure 3**). Instead of a read strobe, the bus provides a "write strobe" signal. When **MWRITE** (memory write) is active high, the NOR gate is active low. This output becomes the "write enable" active low signal to the memory chips.

But, before it reaches the RAM chip, it passes through a 74LS32 OR gate (**Figure 7**), which also has a switch as its other input. For each bank, a corresponding switch acts as a "write enable" switch to otherwise write protect the contents of that memory bank. This may seem to be an odd feature, but it was not unusual in its time.

When MWRITE becomes active, the processor has already put the data onto the bus: **Figure 6** shows that bus data out lines **DO7** and **DO6** are connected via tristate buffers to the memory chip data input lines; and that those buffers are always enabled (the /ENBL line is tied low).

Bank switching

We have described a scheme to allow any memory card to take a position in the 64K memory space of the S-100 bus. Memory cards are selected by banks, some part of the 64K memory addressed by 16 address bits. What if you had **more memory**? For example, some ROM that you only used on power-up? Or some pieces of programs that you only needed part of the time? Or a memory mapped video card, that you only addressed when you needed to update the display? Then 64K is not enough memory space!

Solution: since memory is already in banks anyway, create a scheme to switch memory **under program control**. You need two pieces of circuitry to accomplish this: a scheme to "initialize" or "reset" memory to some original state, like on power-up; and a scheme to selectively disable and enable different memory banks in the same area of memory. We'll first describe this **bank switching** scheme; then we'll discuss the how it gets initialized.

We saw in the previous section that a bank of memory can be selected by address and bus status. Why not make this selection conditional, by adding a circuit we can **programmatically** address? **Figure 8** shows such a circuit from an IEEE-696 reference book. This circuit is accessed with **I/O read and writes,** which operate similarly to memory reads and writes only with different bus status and strobe signals. The port address - almost always 40H and only the first 8 address lines in the S-100 world - is read from lines **A0 through A7** in a circuit similar to **Figure 1**. This port address signal, active low, is inverted (to active high) and is AND'ed with inverted bus signal **PWR*** (active low for bus writes) and bus signal **SOUT** (active high for I/O writes). Therefore, the NAND gate is active low when the port address is active and the bus status is an I/O write. This gate signal clocks the 74LS74 flip-flop - er, data latch - latches the state of its data line D, and

continued on Page 24

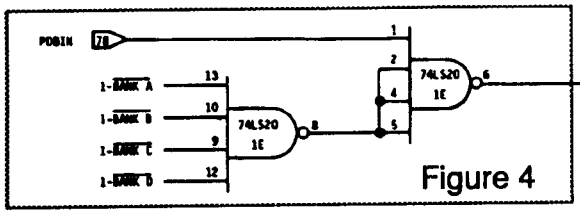


Figure 4

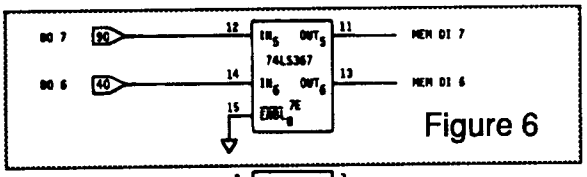


Figure 6

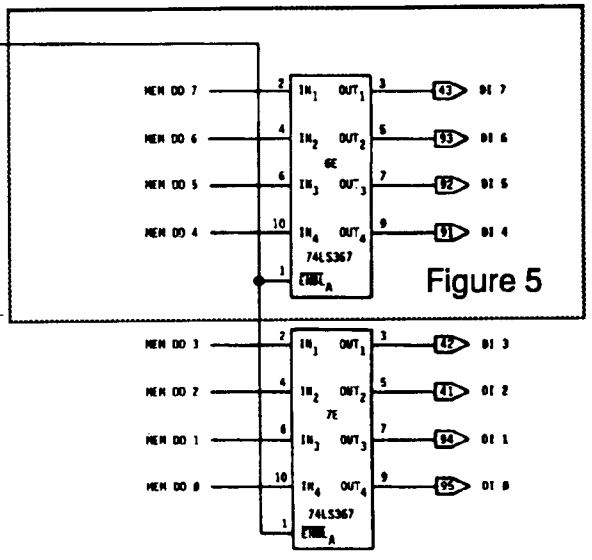
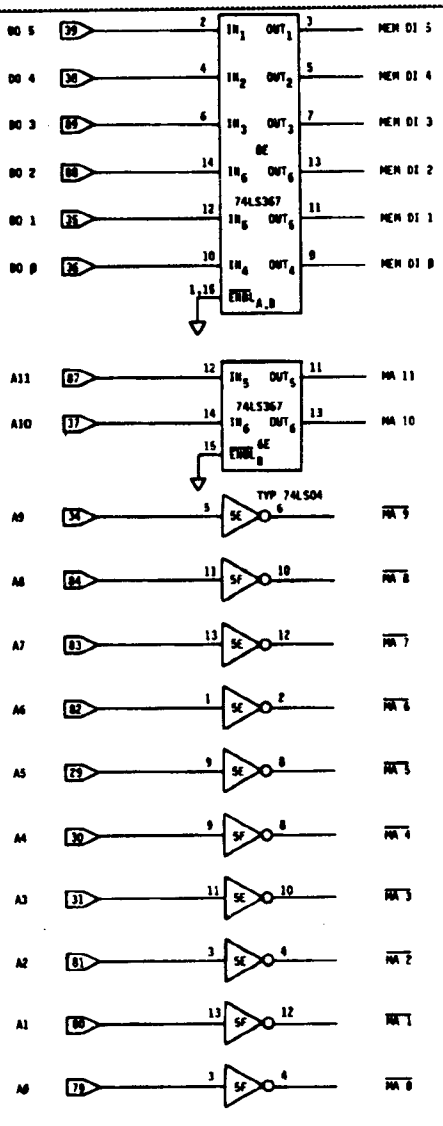


Figure 5



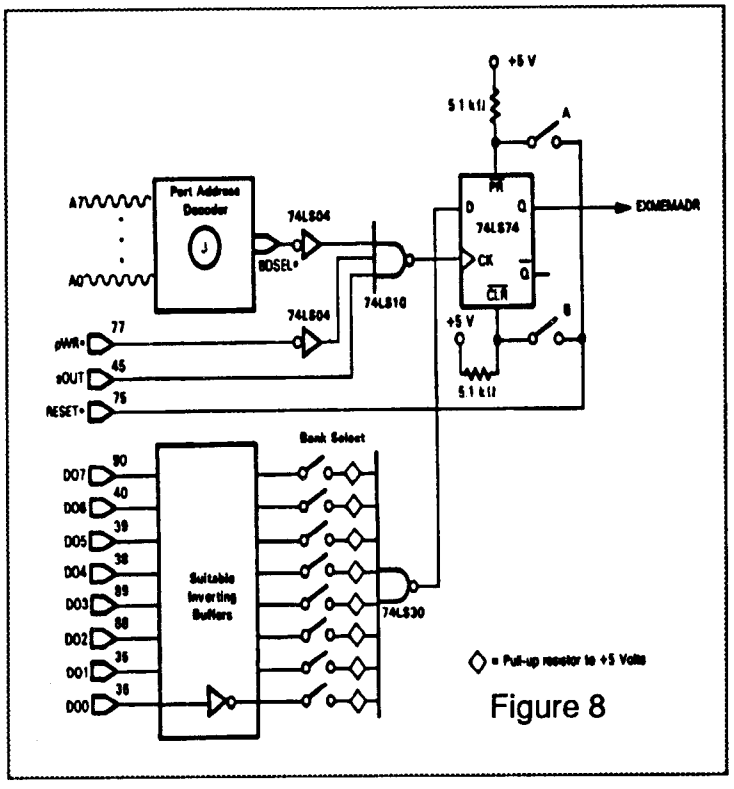
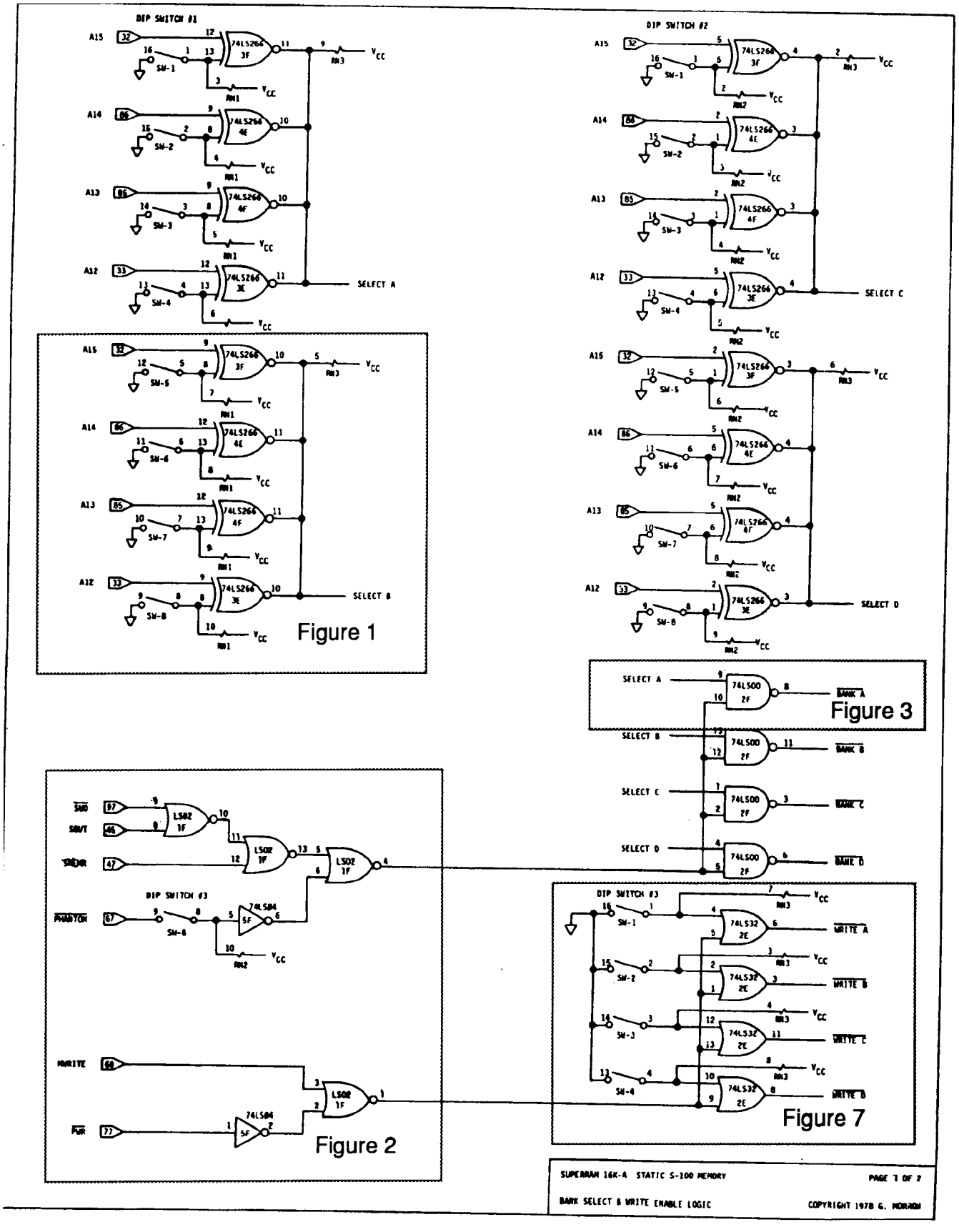


Figure 8


DIP SWITCH #1
DIP SWITCH #2
SELECT A
SELECT B
SELECT C
SELECT D
Figure 1
Figure 3
Figure 2
Figure 7
DIP SWITCH #3
SUPERRAM 16K-A STATIC S-100 MEMORY
PAGE 1 OF 2
BANK SELECT & WRITE ENABLE LOGIC
COPYRIGHT 1978 G. MORROW

sets the output Q to that state. This Q output, labeled "EXMEMADR", gives us a programmable signal to enable or disable a memory card's address logic!

And where does the data for this latch come from? The bus data out lines, **D0 through D7**. The circuit shows the data out lines are tied through inverting buffers to "bank select" switches, one per data line. All switches are connected to the inputs of an 8-input NAND, the 74LS30, inputs that are otherwise tied high. If a bus data line is high, the corresponding switch input is low. IF the switch is closed, that low pulls down the corresponding input to the NAND gate. As all the other inputs are presumably high, the gate becomes inactive and the NAND gate's output goes low. Consequently, when a switch-selected data line is high, the NAND output - and the data input to the latch! - is high. And, if this occurs when the port is "addressed", the latch's output goes high **and stays high until the latch is addressed and programmed low**.

We've now got a mechanism for programming a memory bank to be continuously available or unavailable. Now we need to initialize this circuit. On power-up, the bus signal **RESET*** is active low. Our bank select circuit provides switches to conduct this signal to the latches' \PR to set the latch (enable the bank) or to \CLR to clear the latch (disable the bank). Now, on power-up or reset we can establish an initial memory configuration - say, with a ROM memory bank enabled. We can have a ROM program to disable itself and to enable a RAM bank, after the ROM program has done its business.

Your mileage may vary

I should note: while this memory banking scheme was common, it was not universal! Cromemco, and I think NorthStar, used similar but not identical schemes. Check your docs and schematics (or software, if you have source!) to confirm this scheme. The reader is welcome to send me an explanation of his or her system's methods!

Phantom and other methods

But there are other conditions where memory must be managed that are not appropriate to bank selection and reset defaults. How do you tell the processor to execute a ROM at power up? Or how does a front panel control the bus? You jam a jump or other instruction on the bus. So how do you tell a memory card to get out of the way of such an event? Look again at **Figure 2**, at the **/PHANTOM** line. When this line is active low, and the switch connected to it is closed, the signal is inverted to active high and sent to the OR gate to **disable** the memory card's read and write strobe decoding. Consequently, the card can be selected to be disabled (or not disabled) during PHANTOM operation. I'll leave details of such operations to another column.

In addition, really really **old** memory cards (IMSAI, Altair, Godbout, etc.) used the S-100 lines **PROT** (pin 70) and **UNPROT** (pin 20) to set or clear a latch. The output of this latch would disable or enable (respectively) memory write operations for the addressed bank of memory. The latch would also make active the **/PS** (Protect Status) bus line. This method was a "memory-mapped" way to provide write protect, much like the "I/O mapped" bank select, and was generally performed by a front panel.

An important note: these lines became obsolete pretty quickly, and pins 20 and 50 are often grounded on most S-100 cards. Beware! this will disable your front panel, so you need to either unground these pins, or slip a piece of tape over the pins to isolate them from the bus connector.

When 16 address bits (or 8 data bits) are not enough

By the early 1980's, memory was cheaper and programs were bigger. And, as I mentioned earlier, the 8086 and other processors were available that could address more than the 64K of memory provided by a 16-bit address. Compupro and other companies began to produce card to an expanded version of the S-100 bus, which became the IEEE-696 bus. For this discussion, I'll simply note that eight more address lines were added, **A16 through A23**. It is a simple matter to decode these lines, using circuits similar to the ones just described, and to add the result to the address selection logic. In the Compupro RAM 17 64K memory card, (not shown) the eight lines are compared to eight switches with a single chip, the 25LS521 octal comparator, producing an active low signal when the upper address bits match the switch settings. By now, I think you have the idea.

I'm afraid space does not permit me to discuss how the IEEE-696 bus supports the **real-time** conversion of the two 8-bit unidirectional data lines into a 16-bit bidirectional data path. It's kinda neat. I should also note the IEEE-696 specification includes another bus signal, **PSYNC**, that is active high at the start of every bus cycle (like memory read or write); and **pSTVAL***, which is active low when the address and status lines are active and stable on the bus. Maybe next time.

Graphics

Figure 1 through 7 are from the Morrow Designs Superram 16K-A manual. Figure 8 is from the book "Interfacing to S-100/IEEE-696 Microcomputers" by Mark Garetz and Sol Libes, M&T Publishing Inc, 1988 (previously by Osborne/McGraw-Hill), courtesy of Supermicro Magazine, Provo UT. Are they still around?

TCJ Center Fold CPU280

Tilmann Reh

Special Feature

All Users

Zilog Z280 CPU

Now it's been exactly four years since I described the CPU280 in TCJ issue #53. In that early article, I described the circuit principles and also some details, but we didn't include a schematic of the board (though in TCJ #54, a photograph of the CPU280 board was printed (p.6), which was taken at our club meeting in Germany). Now Dave, our new editor, has suggested to include these drawings as the Centerfold of this TCJ issue, to give you an example of current 8-bit computer technology.

So you should now see the circuit drawing of the CPU280, along with the PCBs part layout and some accompanying information like bus connector pinout and GAL contents, on these Centerfold pages.

To give our new readers a first impression: the CPU280 is a powerful single-board computer using the Z280 processor, which is an enhanced Z80 core with extended instruction set (about twice as much instructions and addressing modes) and many on-chip peripherals. The CPU280 is build on a standard EuroCard (100x160 mm, or about 4x6.2 inches) and connects to the ECB bus (a european standard 8-bit bus using VME-type connectors).

As mentioned in that early description, the circuit design of the CPU280 is really straightforward, though it might not seem so at the first glance. The drawing might appear rather complex because it is all combined on a single sheet, while the functional circuit blocks are much simpler. We might devide the circuit into five parts:

1. CPU basics
2. Memory interface
3. Onboard I/O
4. ECB bus interface
5. Glue logic & GALs

1. CPU basics

On the CPU280, the Z280 processor is used in 16-bit mode. This means we'll have a 16-bit multiplexed address/data bus, and "Z-bus" type control signals (as used by the Z8000). So we definitely have to latch the 16 lower address bits on each memory or I/O access (IC2,IC3) to get a static 24-bit address, and we also have to decode the Z-bus lines (especially its status lines ST0..3) to get appropriate access signals. Upon reset, some basic operating parameters of the Z280 processor can be set up by asserting /WAIT during the rising edge of /RESET and placing the configuration data on the lower data bus. After /WAIT is released, the processor starts operation. This configuration is done by the circuit around IC5, with /WAIT delayed by R1/C5, while RN2, RN6, and R4..7 supply the configuration data to the processor (depending on the jumpers J1..4). Note that the reset signal to the Z280 itself is fed through a GAL to satisfy the CPUs rise time needs. The Z280 chip also contains a serial interface (UART) and a four-channel DMA controller, which are used within the CPU280.

2. Memory interface

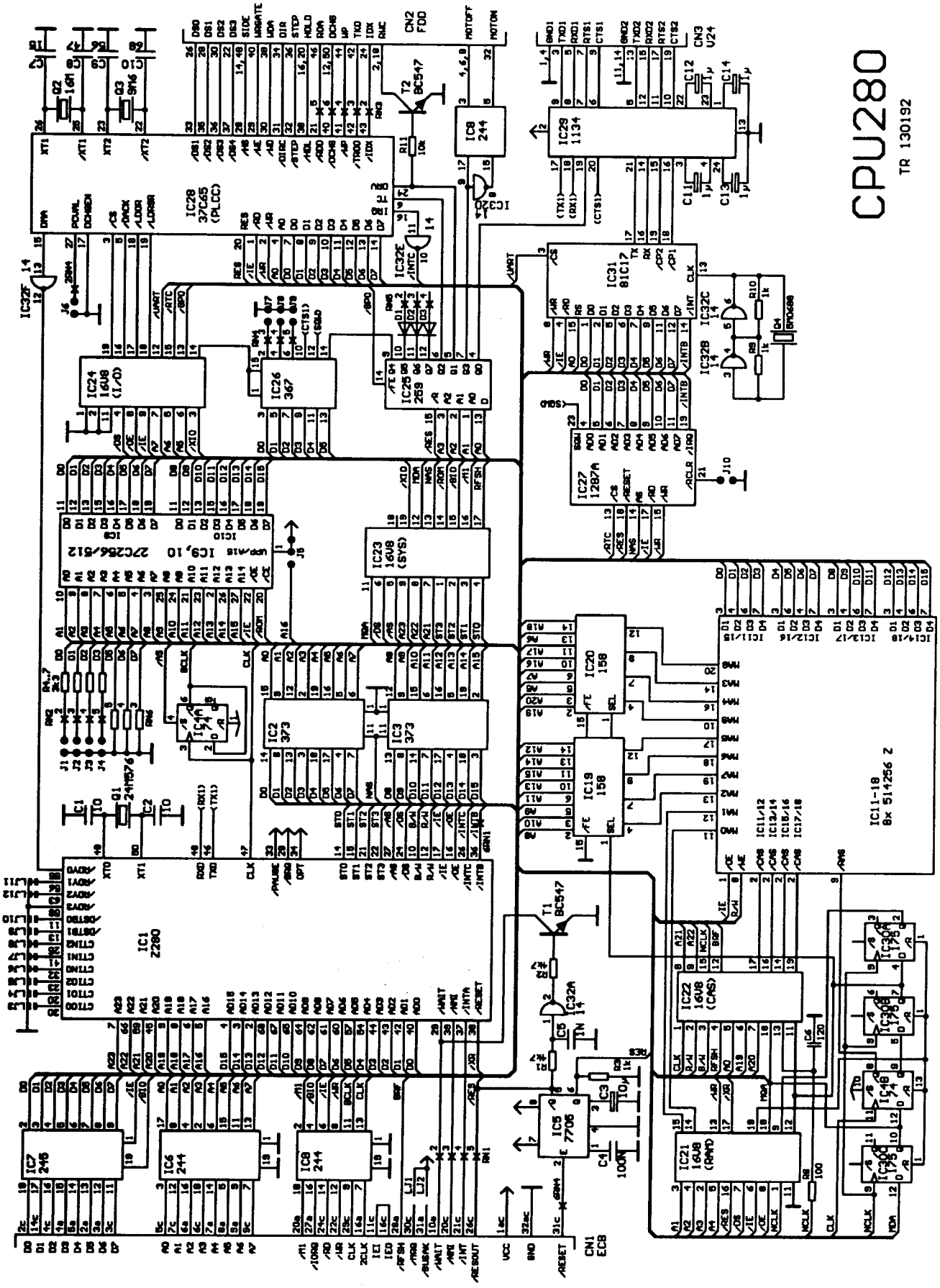
All memory accessible by the Z280 is located onboard. Since the Z280 supports different timing for two memory areas, EPROM memory and RAM memory each use a different area. There are two EPROMs (IC9..10) to hold the 16 bits of data, while four 4-bit RAMs are connected "in parallel" for 16 bit. Interfacing to the EPROMs is as simple as can ever be, however the RAM circuit is somewhat more complex. It uses a fully synchronous timing chain (IC30,IC4) and some glue logic contained in GALs (IC21..22) to provide the necessary signals for the dynamic RAMs (IC11..18) and their address multiplexers (IC19..20). (The principle of the timing chain is that a logic '1' is shuffled through the chain with each clock pulse during memory accesses.) To support the Z280s burst mode (used to fill up the internal cache memory), the GALs also contain logic to increment the lower address bits and generate separate /CAS pulses for each in-burst access.

3. Onboard I/O

The I/O circuit is rather simple again. All decoding is done in a GAL, which provides differently sized address areas for each select signal. The on-board I/O consists of the floppy controller (IC28), a second serial interface (IC31), and a real-time clock with non-volatile RAM (IC27). There further are two general purpose ports for bitwise input and output, GPI resp. GPO (IC26,IC25). The floppy controller uses one channel of the Z280s DMA controller to transfer data to/from memory.

4. ECB bus interface

The interface to the ECB bus supports I/O transactions only. Due to the speed limits of the bus, I/O accesses are stretched to meet the standard 6 MHz Z80 timing, while appropriate Z80-type control signals are generated. The processor clock is also divided by two for the bus, and the


CPU280
TR 130192

ECB-bus pinout:

Physical connector: VME-type 64-pin, rows a+c used.

Row a	Pin No.		Row c
+5V	1		+5V
D5	2		D0
D6	3		D7
D3	4		D2
D4	5		A0
A2	6		A3
A4	7		A1
A5	8		A8
A6	9		A7
/WAIT	10	*	D8 (A16)
/BUSREQ	11		IEI
BAI (A18)*	12	*	D9 (A17/A19)
+12V	13	*	D10 (A18/A20/-12V)
D11 (A19/-12V)*	14		D1
-5V	15		-12V
2xCLK	16		IEO
BAO (A17)*	17		A11
A14	18		A10
+12V	19	*	D13 (A16)
/M1	20		/NMI
D14 (A22)*	21		/INT
D15 (A23/RDY)*	22		/WR
(BAI)*	23	*	D12 (A21)
+UBAT	24		/RD
NxCLK (BA0)*	25		/HALT
(RDY)*	26		/RESOUT
/IORQ	27		A12
/RFSH	28		A15
A13	29		CLK
A9	30		/MRQ
/BUSAK	31		/RESIN
GND	32		GND

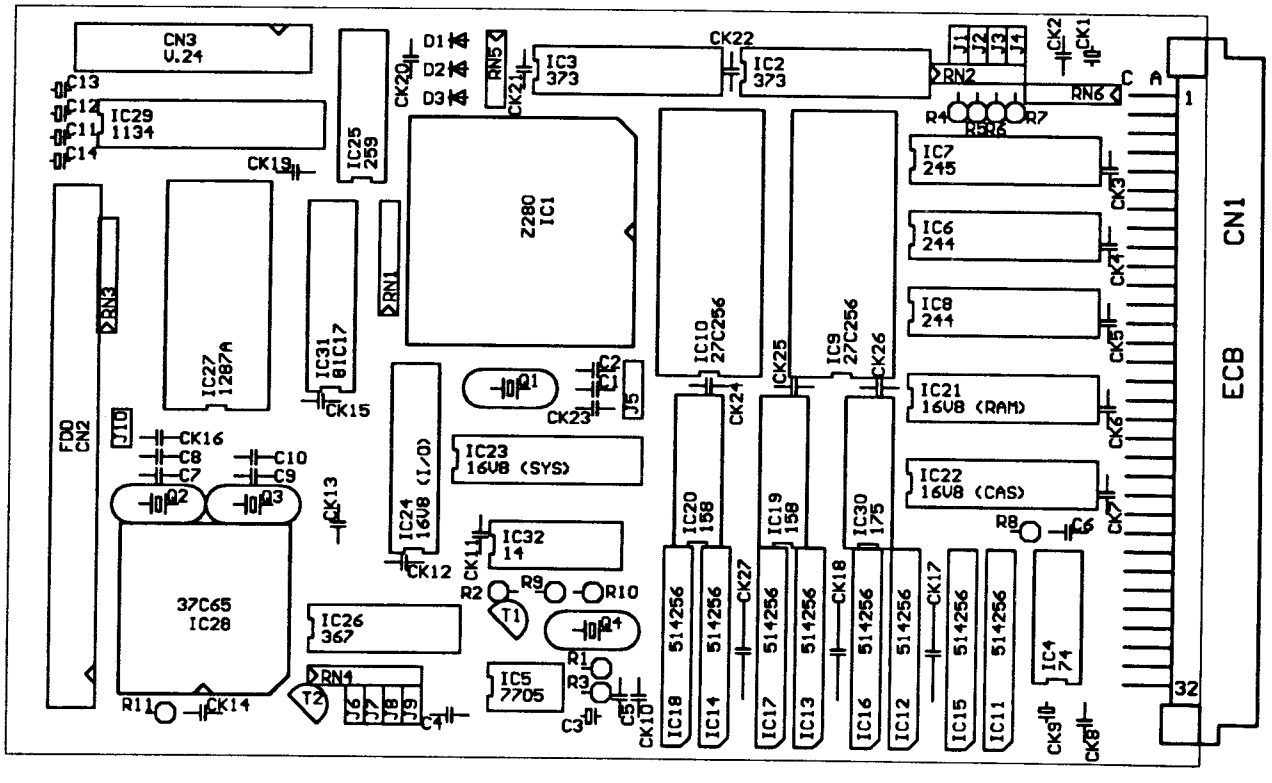
The 13 pins marked with an asterisk have different meaning, depending on the maker of the boards. The original meaning defined by Kontron (the company which introduced the ECB-bus) is always printed first. Additions or redefinitions by other makers are printed in parantheses. The differences concern three signal types: higher order address/data lines, DMA priority chaining, and -12V power supply. The latter easily produces grey smoke if boards of different makers are combined!

Note that Kontron defined the bus with 16 address and 16 data lines. When more memory address space was needed for Z80-based computers, people simply added address lines from A16 up, using some of the "free" bus lines. Unfortunately, each maker took different ones. Maybe this was part of the (common) philosophy to prevent buyers from purchasing anything from another maker...

Luckily, most of the more simple boards (or those which need the bus only for simple tasks, like the CPU280) use only the common subset of the bus, so there are many cases where no problems arise.

Just for information, the 16-bit variant for which Kontron reserved those high-order data lines was rarely seen. I don't think it ever had a significant meaning in the ECB-bus market.

The meaning of the bus signals exactly follows the Z80 processor signals, since this is where this bus is derived from. So, for timing and meaning of the signals, please refer to the Z80 CPU data sheet. Besides the power supply, additional signal lines are just the priority chains for interrupts (IEI/IEO) and DMA (BAI/BAO/RDY).



divider flipflop is synchronized for every I/O access to exactly reproduce the Z80s phase relationship between control signals and clock. To provide support for external interrupts from the bus, some extras were added (/M1 to simulate interrupt acknowledge, special I/O decoding to simulate RETI instruction fetch for the bus). External interrupts might use external vectors (Z80 peripherals) or an internal (fixed) vector. The non-maskable interrupt also is connected to the bus. Note that there also is a refresh signal (BRF) generated for the bus, although there is no external memory support. This signal is to support I/O-based RAM-disks built with dynamic RAM.

5. Glue logic & GALs

Looking at the four GALs containing all the "glue logic", I must admit that it took some time to design this part of the circuit. When designing a complex circuit like the CPU280, I first make drafts of all the main functions I need, and don't care much about the glue logic which is necessary to make them work. After all functional blocks are designed, I collect all the glue logic and try to combine it in a senseful manner. This results in some standard TTL parts (like flipflops, multiplexers, inverters, buffers etc.) and some remaining "glue" which I then try to fit in as few GALs as possible. I could also have used a larger PLD, but I still prefer the smaller GALs for some reasons. So after I knew the functions which should be programmed into GALs, I started puzzling inputs and outputs to need a minimum number of GALs. This surely is more work than is obvious when seeing the final result! Though much more versatile than PALs, GALs are restricted in what you can do, mostly because there are only eight outputs per chip. Some times I thought I had found the optimum combination just to realize that one of the GALs would need nine output lines for it (of course two outputs were free at another GAL then, or even some inputs at the particular one!). One of the main rules when optimizing GAL circuits is to connect any given signal to a minimum number of GALs. For this, you sometimes have to "cascade" two or more GALs, like I did with the I/O decoder on the CPU280: the system decoder GAL (IC23) decodes *that* there is an on-board I/O access, while the I/O decoder GAL (IC24) uses this signal to further decode *which* I/O device was selected. (When cascading GALs, however, you must keep an eye on the overall timing.) During GAL optimization, sometimes you also modify the functional block circuits to make more efficient use of the glue logic. At that point, you have to enter the loop again and restart collecting and optimizing all "glue" needed. I did this with the CPU280 several times, before the circuit made perfect use of all TTL parts and GALs, minimizing total chip count. But the result can be seen, at least in my opinion: With four cheap standard GALs, the total parts costs are much lower, the PCB layout much easier, and support is much better, than with a single large PLD!

Tilmann Reh, Dec 1995

Ed: Just after Tilmann sent me this, Zilog announced that they would stop making the Z280 in early 1996.

CPU280 GAL EQUATIONS

```
TITLE          CPU280 RAM-TIMING AND NIBBLEMODE IC21
AUTHOR         TILMANN REH
COMPANY        REHDESIGN
DATE            23.07.1990

CHIP Z280RAM PALCE16V8

NCLK A3 A1 A2 A4 IE DS OE MQD GND
QOE MUX WR MA0 MA1 RES CPURES MQA FFR VCC

EQUATIONS

/WR  = /OE * /DS
MA0 := /MQA * A3 + MQA * /MUX * A1 + MUX * /MA0
MA1 := /MQA * A4 + MQA * /MUX * A2 + MUX * MA0 + MUX * MA1
/FFR = IE * DS * MQD
CPURES = RES
```

```
TITLE          CPU280 CAS-DECODER IC22
AUTHOR         TILMANN REH
COMPANY        REHDESIGN
DATE            20.09.1992

CHIP Z280CAS PALCE16V8

CLK RW BW RFSH A0 A19 A20 A21 A22 GND
MUX BRF NCK CAS1L NCLK CAS0H CAS0L MQA CAS1H VCC

EQUATIONS

/BRF    = RFSH * MQA
NCLK    = /CLK

/CAS0L  = MUX * /RFSH * /A22 * /A21 * /A20 * /A19 * BW * A0
     + MUX * /RFSH * /A22 * /A21 * /A20 * /A19 * /BW *
     ( CLK + NCK + /RW ) + RFSH * /MUX * MQA

/CAS0H  = MUX * /RFSH * /A22 * /A21 * /A20 * /A19 * BW * /A0
     + MUX * /RFSH * /A22 * /A21 * /A20 * /A19 * /BW *
     ( CLK + NCK + /RW ) + RFSH * /MUX * MQA

/CAS1L  = MUX * /RFSH * /A22 * /A21 * /A20 * A19 * BW * A0
     + MUX * /RFSH * /A22 * /A21 * /A20 * A19 * /BW *
     ( CLK + NCK + /RW ) + RFSH * /MUX * MQA

/CAS1H  = MUX * /RFSH * /A22 * /A21 * /A20 * A19 * BW * /A0
     + MUX * /RFSH * /A22 * /A21 * /A20 * A19 * /BW *
     ( CLK + NCK + /RW ) + RFSH * /MUX * MQA
```

```
TITLE          CPU280 SYSTEM-SIGNALS IC23
AUTHOR         TILMANN REH
COMPANY        REHDESIGN
DATE            02.12.1990

CHIP Z280SYS PALCE16V8

ST3 ST2 ST1 ST0 AS DS A21 A22 A23 GND
MQA MDA NAS ROM BIO M1 RFSH NNAS XIO VCC

EQUATIONS

NAS  = /AS
NNAS = NAS
/ROM = ST3 * /A23 * /DS
MDA  = ST3 * A23 * /A22 * ( /AS + NAS + NNAS )
     + /ST3 * /ST2 * /ST1 * ST0 * ( /AS + NAS + NNAS )
     + MQA
RFSH = /ST3 * /ST2 * /ST1 * ST0
/BIO = /ST3 * /ST2 * ST1 * /ST0 * /A23 * /A22 * /A21 * /DS
     + /ST3 * ST2 * /ST1 * /ST0
/XIO = /ST3 * /ST2 * ST1 * /ST0 * /A23 * /A22 * A21
     + /ST3 * /ST2 * ST1 * /ST0 * /A23 * A22 * /A21
/M1  = /ST3 * ST2 * /ST1 * /ST0
     + /ST3 * /ST2 * ST1 * /ST0 * /A23 * /A22 * A21
```

```
TITLE       CPU280 IO-ADRESS-DECODER IC24
AUTHOR     TILMANN REH
COMPANY    REHDESIGN
DATE       24.03.1992

CHIP Z280IO PALCE16V8

NC NC XIO DS A6 A5 A7 OE IE GND
NC UART GPO GPI RTC DACK LDOR LDRSR FDC VCC

EQUATIONS

/RTC    = /XIO * /A7 * /A6
/FDC    = /XIO * /A7 * A6 * /A5
/DACK   = /XIO * /A7 * A6 * A5
/LDOR   = /XIO * A7 * /A6 * /A5
/LDRSR  = /XIO * A7 * /A6 * A5
/UART   = /XIO * A7 * A6 * /A5
/GPI    = /XIO * A7 * A6 * A5 * /IE * /DS
/GPO    = /XIO * A7 * A6 * A5 * /OE * /DS
```

The First TRS-80

By Gary Ratliff

Older Systems

All Readers

TRS-80 Model 1

INTRODUCTION

The Radio Shack line of computers was one of the most prolific lines ever manufactured. They made computers for the Z-80, the 6809, the 68000, and the 80x86 lines of chips. These of course utilized many different operating systems and offered many varying flavors of the BASIC computer language. XENIX was offered on the computers which used the 68000 chip, while OS-9 was offered on the COCO with the 6809 chip set. The Z-80 chip had several operating systems available for it including the CP/M system with its vast line of business software. But there was also TRSDOS, and LDOS among many others from Radio Shack.

For this article, we'll concentrate on the TRS-80 Model 1 where it all began. The Model 1 is the oldest model of the TRS-80 line and therefore the most likely computer to lack support and documentation. This was (with the Apple I and the Commodore PET) the very first of the 'ready to use' computers. Other computers at this time were assembled from kits while these were immediately usable once they had been removed from the box and connected.

The December 1975 issue of Popular Electronics ushered in the computer age for the masses which makes the personal computer twenty years old. These first kits came with 256 bytes of RAM, and a set of toggle switches for entering programs. They used an 8080 microprocessor chip and were mostly built from scratch by the purchaser of the system.

There were those who were fascinated by the new computer, but unwilling or unable to build their own computer. So once the new computer kits began to sell like hotcakes, other manufacturers began to enter the market with ready-to-use computers. The PET, Apple, and the TRS-80 dominated the market for these types of computers.

The material for this article was gathered from the books listed in the Bibliography. A 1978 guide to buying one of these new computers called THE HOME COMPUTER HANDBOOK states that all three of these companies' computers (PET, Apple, and the TRS-80) have the S-100 bus while none of them did. So some of the books of this era will contain errors.

THE TRS-80 MODEL 1

So what was the Model 1 and what was available in the Level 1 instruction set? The smallest configuration for the Model 1 computer consisted of the screen display, a keyboard, a cassette tape recorder, and 4K RAM. Radio Shack soon offered the following line of expansion kits for this computer:

1) 16K RAM (Up to 3 of these for a total of 48K RAM max.)
2) Level I to Level II ROM (added many BASIC statements and made the computer suitable for business use.)
3) Lower Case Kit (at first only Upper case letters were available.)
4) Numeric Keypad Kit
5) RS-232C Interface Board (this added the ability to link your Model 1 to other computers and data services.)
6) Double Density Kit.

All that was required to use the Model 1 was to plug it in and connect the keyboard and screen display cables. A cassette unit was required for storing and retrieving programs. There were initially problems with both the keyboard and the cassette unit. There were also different model cassette and keyboard units. The CTR-40 and CTR-41 tape units which came with the first Model 1's required that the lead into the 'REM' jack be un-plugged prior to repositioning the tape in the cassette unit. The CTR-41 also needed to have a dummy plug in the MIC jack at all times. The CTR-80 which was introduced by Radio Shack in early 1980 corrected this problem and does not require the use of the dummy plug at all.

Another common problem with the earliest cassette units was selecting the proper recording level. Cassette units made prior to 1980 were offered a free repair to correct this problem. This repair was to install a small capacitor to filter out a noise spike and make the cassette unit less sensitive. It was recommended that you try to save some programs and re-load them. If this doesn't work, adjust the volume level in tiny increments and, once a reliable setting is found, mark the setting with "white out". The recommend volume setting for the CTR-40 and CTR-41 unit was between 4 and 6; for the later CTR-80 unit was between 3.5 and 4.5.

After much use, the cassette unit required de-magnetizing. There were special wands for this. Whether this was actually needed was debated. Anyway, the process was quite a bit of fun. The cassette unit was opened and the buttons

pushed so that the head was protracted. Then and only then was the wand plugged in. You started at about 18 inches from the cassette unit and passed the want in a path overt the length of the cassette unit. On each pass the wand was brought a little closer to the read-write heads. The wand was passed at a slow steady pace and eventually reached as close to the heads as you would dare come. Some wands had a clear plastic cover to prevent the tip from scratching the read-write heads should they inadvertently make contact. Now the process was reversed and the wand was slowly moved further away from the heads until the starting distance was reached. Here the wand was un-plugged and the cassette unit declared de-magnetized.

Another common problem was keyboard de-bouncing. There was a special tape which was loaded upon first starting the computer session. This tape was also supplied free to purchaser's of the first Model 1 units. This problem was eliminated from the later models of the TRS-80.

The use of the Model 1 TRS-80 is almost the same as knowing how to program in the BASIC computer language which this model used. For completeness, we will introduce the Level 1 commands. But if someone was serious about developing useful computer programs, they needed the upgrade to install the Level II ROM set or to buy a Model I with it already installed.

The Level One BASIC command set was limited to these instructions:

```
LET, IF, THEN, REM, MEM, ABS, INT, RND, RND(0), AT,
CLOAD, CSAVE, DATA, INPUT, INPUT " (Prompt Message.)
", INPUT#, PRINT, PRINT AT, PRINT TAB, PRINT#, READ,
RESTORE, TAB, USING, CLS, POINT, RESET, SET, CONT,
LIST, LIST # (list program from line number # to
end, or from start # to end #), NEW, RUN, RUN # (Run
from line number #), END, FOR, GOSUB, GOTO, NEXT,
ON, RETURN, STEP, STOP, TO.
```

In addition, Level I BASIC supported the logical operations AND and OR, but used the symbols ' and + respectively. A Level II BASIC Statement like "IF A AND B" could be coded in Level I as "IF A ' B" and the statement "IF A OR B" would be coded "IF A + B". The common math operations addition (+), subtraction (-), multiplication (*), and division (/) were supported. The logical operations less than (<), greater than (>), equals (=), not equal (<>), less than or equal (<=), and greater than or equal (>=) were also supported.

An interesting feature of the Level I BASIC system was that it supported a method of entering these keys using shorthand. (The Commodore PET also had a shorthand method for key entry and these were detailed in an early article I wrote for MICROCOMPUTING called "PET Shorthand COMPLEAT".)

These were achieved mostly by entering the first letter of the keyword followed by a period. The complete list of keywords which could be entered via the shorthand method is as follows:

```
PRINT P., NEW N., RUN R., LIST L., END E., THEN T.,
GOTO G., INPUT IN., MEM M., FOR F., NEXT N., STEP
S., STOP ST., CONT C., TAB T., INT I., GOSUB GOS.,
RETURN RET., READ REA. (both are four key strokes so
there are no net savings) DATA D., RESTORE REST, SET
S., RESET R., POINT P. PRINT AT P.A.
```

As I mentioned earlier, the information comes from printed sources which may contain errors. This source lists an A. without any clue as to which command is being given shorthand. Also is list RESET as R. and RND as R. while also having RUN as R. logically these can not be true. The other inconsistency was that S. could be used for both SET and STEP while R. is given for RUN, RND and RESET. One of the disadvantages of writing about antique computers would seem to be the fact that the data supplied by books which are long out of print may not be verified unless the antique computer in question is part of you computer collection. Although I did shop in the flea markets and other outlets for a Radio Shack computer to use to verify the article, the oldest I could locate was a Model 4 computer. The only consolation I can offer for the possible fact that the material may not be correct is that I have seen the rumor to the effect that a mint condition Apple I computer commands $25,000. If you have a mint condition Level I BASIC Model I TRS-80 it may also command a high price.

MODEL I LEVEL II BASIC

After the Model 1 Level I BASIC, there was LEVEL II BASIC. Level II BASIC gives the user a very complete set of the BASIC instructions which may be used for math and business. Level I BASIC was not even able to be ported to LEVEL II BASIC without the use of a special program called CONV. There were several differences in the syntax of statements. An example of these differences are: Change the Level I PRINT TAB(5), "HELLO" to PRINT TAB(5) "HELLO" or PRINT TAB(5); "HELLO". This is the purpose of the CONV program.

The easiest way to determine if you have the Level I or Level II Model 1 ROM set is to use PRINT AT and PRINT @. The first is acceptable with LEVEL I while the latter is acceptable using Level II ROM's. The encyclopedia mentions that there is very much upward compatibility between this and the other BASIC's offered by Radio Shack. Hence, the examples presented should be usable on any Radio Shack computer using the Z-80 chip.

Since this article can't be a complete course in using the BASIC language for the Model 1, we will concentrate on some specific areas. These will be the use of the graphic commands, using the EDIT mode to correct errors, and the use of machine language with the Model I.

GRAPHICS

The key to graphics is the use of the SET, RESET, POINT, CLS and PRINT @ commands. The screen of the Model 1 is divided into 16 rows of 64 columns; this is a total of 1024 elements. The first line would be numbered from 0 to 63; the second from 64 to 127; the sixteenth from 960 to 1023. (A Basic program to obtain the address of the first location of the 16 screen lines would be as follows:

```
10 FOR I = 0 TO 1023 STEP 64
20 PRINT I
30 NEXT I
```

A run of this yields these addresses: 0, 64, 128, 192, 256, 320, 384, 448, 512, 576, 640, 704, 768, 832, 896, 960.)

For the purposes of the SET, RESET, and POINT commands, these are broken into finer areas. Each location is broken into 2 vertical units and 3 horizontal units. These range from 0-127 in the vertical direction and from 0 to 47 in the horizontal direction. The SET command will turn that specific screen area to white and the RESET command will turn that area to black. The POINT command is a logical operator which returns the value true if the location is SET and false if the location is RESET.

You can use the CHR$ function to obtain much faster graphics than from setting and resetting individual locations. The non-standard ASCII characters from 128 to 191 are used for graphic characters which could otherwise be generated by the set and reset commands. Each character location is divided into two by three locations. These would be assigned weights 1, 2, 4, 8, 16, and 32. To these a base number of 128 is added. Therefore, the command SET (0,0) which lights the extreme left hand corner pixel could be obtained by the alternate command:
PRINT @ 0,CHR$(129).

The form of SET, RESET and POINT is SET(X,Y) where X is the vertical coordinate and Y is the horizontal coordinate. Our example SET(0,1) would light the block in the second row in the extreme left. A check of the above weights shows that the value to add is 4. And since 128 + 4 is 132, this would be achieved with PRINT @ 0 CHR$(132). Where the savings is realized is if a graphic were to be constructed which would use the SET(0,0) to be followed with the SET(0,1) command. Here two pixels are to be lit in the same "character". These have weights of 1 and 4. So lighting these two areas is realized by the command PRINT @ 0 CHR$(133) where 133 = 128 plus the combined weights of 1 plus 4.

This result may also be obtained by using the screen address and the POKE commands. The weight to use is calculated in the same manner as the value to use in the CHR$ statement (128 plus the individual weights). For the Model 1 computer the beginning address of the screen is 15360. So the same effect may be obtained from PRINT @ 0 CHR$(129) as from POKE 15360,129. And from the second example PRINT @ 0 CHR$(132) would be POKE 15360,132.

The range of the screen is the same 1024 screen positions. The addresses which the screen occupy are from the initial address which is 15360 to the initial address plus the screen length or 16383. Since the PRINT @ and POKE in this example perform the same task, using the above information it is easy to translate from one form to another.

One common task is to fill the screen with all set items. This is to "white out" the screen. Since we want all six blocks set and these have weights as explained in an above section, we combine the weights and add to that the initial 128 to obtain the value: 191.

The following statements each will "white out" the screen:

```
FOR I = 0 TO 1023: PRINT @ I CHR$(191); : NEXT I.

FOR I = 15360 TO 16383: POKE I,191 : NEXT I.
```

Note that in this example another feature of LEVEL II BASIC is demonstrated; that is the use of the : to allow for many statements on the same line. This technique is often used to save computing time in statements. However, it also makes the programs more difficult to follow. From the structured programming standpoint each statement should have one command and be on a separate line.

Creating graphics as you can see from the details presented above requires much patience and planning. A sheet of graph paper will aid in determining which blocks should be set or reset to create your work of art.

BASIC AND MACHINE LANGUAGE USE

As the last section introduced an important memory address and demonstrated how to manipulate that address directly via the POKE statement, the next area we wish to explore is the use of linking to machine language.

The important commands in this type of task will be: PEEK, VARPTR, USR(x), SYSTEM, and POKE. VARPTR is used to find the location of a variable in memory. Its form is B = VARPTR(A$), B = VARPTR (C) etc. The USR function is used to run a machine language routine. Its address must first have been POKED into memory locations 16526 and 16527. It must have also been converted into proper Z-80 address format.

At this time it would be appropriate to discuss the memory layout of the Model 1 Level II BASIC TRS-80 computer. This example will assume the full 48K complement of RAM memory. The memory addresses from 0 to 12287 are used for the 12K ROM which contains the Level II BASIC Interpreter. As previously mentioned 15360 to 16383 contain the video screen. At 16384 is the start the Working Storage area used by the interpreter. This area is followed by the area used to store the program itself. Then this area is followed by the area reserved for storing the variables and arrays used by the program and the area used by the system stack. The rest of memory is free memory.

If you intend to use machine language routines, the first step would be to reserve an area in the top of memory which will not be altered by the BASIC interpreter. In the 48K RAM system we have just described the memory runs from addresses of 0 to 65535. To reserve memory, just enter an address value when the Level II system asks for "MEMORY SIZE?". Here entering 64000 would reserve memory location addresses from 64000 to 65535 for use in the machine language routines.

The machine language program may be entered by using the Editor/Assembler tape, by hand assembling the program and poking it into the system, or by forming the assembly routine into a string which is built up using the CHR$ command. Each of these methods is covered in the book by William Barden Jr cited in the bibliography. Since it's probably impossible to obtain a tape based assembler/editor at this date in time, the POKE method will be demonstrated. For doing this a complete reference to the Z-80 instruction set is mandatory. Two works which have very complete appendixes are cited in the bibliography.

The routine we will code is the machine language version to "white out" the screen which was presented in the previous discussion of graphic statements. As mentioned earlier, this article is a compilation of information gathered from various books which I was able to locate. The routine presented below is from the book by Barden. (It is not wise to try to paraphrase machine language programs!!):

```
LABEL:INSTRUCTIONS COMMENT          (Decimal Code)
WHITE
    LD  HL,15360    ;Screen address    33 0 60
    LD  DE,1024     ;fill count        17 0  4
LOOP
    LD  A,191       ;white char        62 191
    LD  (HL),A      ;white screen      119
    INC HL          ;bump screen       35
    DEC DE          ;decrease count    27
    LD  A,D         ;get msb           122
    OR  E           ;merge lsb         179
    JR  NZ,LOOP     ;loop if not done  32 247
    RET             ;return to basic   201
```

Let us now create a simple basic program to read the above machine language program into the memory area we have previously reserved, set up the machine language pointers for the USR function, and call the machine language routine to "white out" the screen. This is presented below:

```
10 DATA 33,0,60,17,0,4,62,191
20 DATA 119,35,27,122,179,32,247,201
30 FOR I = 0 TO 15
40 READ X
50 POKE 64000 + I, X
60 NEXT I
70 POKE 16526, 0
80 POKE 16527, 250
90 A = USR(0)
100 GOTO 100
```

The above program has the decimal values for the machine language program which was hand assembled placed in data statements. These are poked into the 16 memory locations starting at 64000. This area was previously reserved when the machine was first turned on. The next two pokes in the program set up the memory locations 16526 and 16527 to hold the address of the assembly routine we have just assembled. Addresses in the Z-80 are stored least significant byte first followed by the most significant byte. Here the concept of page addressing may make this clearer.

The 65535 byte address space may be thought of as 256 pages of memory each of which contains 256 bytes of memory. The proper form for storing an address would be byte followed by page. Here the address 64000 is very easy as it is an even multiple of 256. (64000 / 256 = 250 with a remainder of 0) So the byte value is 0 and the page value is 250.

Once the addresses for the USR function have been set, we call the machine language routine and then we enter an infinite loop to avoid having the READY message disturb the freshly drawn screen. Pressing the Break key will get you back to the interpreter.

The Barden text also shows how to place the routines into a dummy string and use the VARPTR to find this address and then call the assembly language routine.

The advantage of the assembly language is that it is much faster than the earlier examples of BASIC code to accomplish the same function. Because the Z-80 chip uses relative branching instructions which the earlier 8080 chip did not have, it is easier to code routines which are able to be located anywhere in memory. Also as this example shows, machine code is generally shorter than the equivalent BASIC statement. This is especially true if you get a full memory map and utilize the power of the assembly routines which are already in the Level II ROM's.

EDIT MODE COMMANDS

This brings us to the third and final area which we are going to discuss in some detail. That is using the BASIC EDIT command to correct syntax errors. If the lines are short and have only one instruction it is perhaps easier to retype the line, but when multi-statement lines or long lines are used the EDIT command provides a convenient method to correct program code.

Let us now create a short program with some errors to illustrate the use of some of the EDIT commands:

```
10 FOT I = 1 TO 25
20 PPRUNT I
30 NIXT I
```

Now when the 'RUN' command is issued, the BASIC Interpreter reports '?SN ERROR IN 10' and types the 'READY' message and automatically enters EDIT mode by typing the offending line number on a separate line: '10_' (Where the _ shows the position of the cursor as it waits for our command to use the EDIT commands to correct the line). Here hitting the 'L' key will list the offending line as '10 FOT I = 1 TO 25'. We observe that the 'T' is the culprit of this particular error message so we press the 'S' key followed by 'T' to search the line for the first occurrence of the letter 'T'. The cursor advances to the 'T' in 'FOT'. Next, press the 'C' key followed by the 'R' key to change the offending 'T' to 'R'. Now that we have corrected the offending line, press the ENTER key to exit from EDIT mode.

We can now list the program again by typing in the command LIST. And the Basic Interpreter will follow these instructions to produce:

```
10 FOR I = 1 TO 25
20 PPRUNT I
30 NIXT I
```

Running the "corrected" program will now produce a '?SN ERROR IN 20' followed by the 'READY' message and the EDIT mode will be invoked with the cursor on line '20_'. Here we again use the 'L' key to list the offending line '20 PPRUNT I'. The command '2SP' will search out the second occurrence of the letter 'P' in the word 'PPRUNT'. Here pressing the 'D' key will delete the 'P' character changing 'PPRUNT' effectively to 'PRUNT'. However, the computer will display 'P!P!' to show that the letter surrounded by the exclamation points will be deleted when the enter key is pressed. Continue to correct the line by entering the 'S U' keys to search for the letter 'U' followed by 'C I' to correct this letter to 'I'. The entire line should be correct now, so exit the EDIT mode by pressing the ENTER key.

Re-listing and running the program as before yields another and the final syntax error message. We know that this is the final error since there are no more lines in the program. In any event, this line is corrected in the same manner.

The cursor waits at "30_". We enter 'L' and then 'S I' and 'C E' followed by the ENTER key. By now we know that this command sequence will search the line for the 'I' in the word 'NIXT' and change it to an 'E' yielding this program when listed:

```
10 FOR I = 1 TO 25
20 PRINT I
30 NEXT I
```

As mentioned, it is often easier and less time consuming to just key in the line again. However, this shows how the EDIT mode may be used to correct program lines within the program.

The EDIT mode may also be invoked directly by entering the command EDIT followed by the line number you wish to change. The EDIT mode allows for these commands. Each obtained by pressing a single key or key sequence. The ENTER key is pressed to exit from edit mode.

CONCLUSION

This article has now covered all the areas which we have outlined. As mentioned earlier, an understanding of Level II BASIC should be readily usable on another version of BASIC in the later Models of the TRS-80 which followed the introduction of the computer in 1977. Many of the books which explained the use of this computer when it first appeared have long gone out of print. In fact, many of the articles mentioned in the bibliography were obtained from libraries discard sales.

So where can you turn for help? The BASIC language is fairly standard and there are many books to teach BASIC.

continued on page 38

TABLE OF EDIT MODE COMMANDS

Command	Function
EDIT line#	This enters EDIT mode on the line number entered.
ENTER	This ends the edit session and returns to command mode.
SHIFT ^	Escape from sub-function. (The ^ is the up arrow.)
n SPACE	Move the cursor n spaces to the right.
n <--	Move the cursor n spaces to the left. (The <-- is the back arrow.)
L	List remainder of the line and go to beginning of the line.
X	List remainder of line, move to end of the line and enter the INSERT sub-command mode.
I	Insert the sequence of characters at the current cursor position. (SHIFT ^ exits this mode.)
A	Cancel changes made and place cursor at start of the line.
E	End editing, Save changes, Go to Command mode.
Q	Quit editing, Cancel change, Go to command mode.
H	Delete from cursor to end of line and enter Insert sub-command function. (See SHIFT ^.)
nD	Delete n characters at the cursor position.
D	Delete character at cursor position.
nC	Change next n characters replacing then with the sequence of the next n characters entered.
C	Change the character at cursor replacing with the next character entered.
nSc	Search for the nth occurrence of the character c.
S	Search for the first occurrence of the next entered character.
nKc	Delete all characters from the cursor position up to the nth occurrence of the character c.

Regular Feature

68xx/68xxx Support

C & Assembly

Small System Support

By Ronald W. Anderson

This is an "aside" column. It is here because it is a typical example of a computer project that I find to be pure recreation. I am going to include some C program source listings in the hopes that we have gone far enough with the series that you will be able to understand them. I think now that we have covered pointers, that will be the case. These listings are in the ANSI "C" style with the slightly different function parameter declaration syntax, though I haven't bothered with function prototypes since there are only a few functions.

In the course of preparing the column with the C lesson that covered pointers, I made some remarks about how it seemed that every program I have written lately in C would be done running by the time I got my finger off of the ENTER key after typing in the command to run the program.

That got me thinking about something I had done a long time ago and so I did some digging. I found some copies of columns that I did years ago for '68' Micro Journal... some information on a project to write a program to find prime numbers.

Just in case you haven't run across the term or the problem before, let me explain. A prime number is a number that is not evenly divisible by any integer other than 1 and itself. Any number can be "factored" until all of the factors are prime numbers. For example 15 is 5 X 3. Neither 5 nor 3 are divisible by another number (I won't repeat the "except 1 or themselves" but you will understand me to have said it). 16 is divisible by 2, and it's prime factors are 2,2,2,and 2. Obviously 1 is a prime number. So is 2. All other even numbers are divisible by 2, so all the remaining primes are odd. The list of primes begins:

1 2 3 5 7 11 13 17 19 23 29 31...

The non primes are:
9 (3 X 3)
15 (3 X 5)
21 (3 X 7)
25 (5 X 5)
27 (3 X 3 X 3)

You can see that a pretty healthy slug of small numbers are primes. The ratio of primes to non-primes thins out quite a bit as the numbers get larger, approaching something in the area of 6% of the numbers less than 16,000,000.

The obvious approach is to test all the odd numbers in the desired range by dividing them by all the other odd numbers up to the one being tested. Of course as soon as we find an even divisor (remainder or MOD function = 0) we have proven the number not to be prime so we can stop there.

Though this certainly works, it is very inefficient as we will see shortly. Let's test the number 15 for example. We can start our test at 3 and of course 3 is a divisor, so we can stop there. Now let's look at 169. It turns out that 169 is 13 X 13 so we will test for 3, 5, 7, 9, and 11, and then find that 13 is a divisor. This is a "worst case" non-prime, a perfect square. Now let's test 167. We test 3 through 11 again, and find that 13 fails too. The insight is that 167 divided by 13 yields a result less than 13. If 167 were divisible by an integer less than 13 we would have already found it! The result of this thought exercise is that we have found that we only have to test divisors up to the square root of the number being tested. That is, when the test divisor squared is greater than the number being tested we can stop and we have proved the number to be prime.

This is an enormous saving. To test 1001, for example, we only have to use test divisors from 3 up to 31. There are only 15 such numbers. If we were to test all odd numbers up to 999 we would have to make 500 tests.

There is a smaller saving that can be realized, which I didn't do on my first try at the program. Only prime numbers need be tested as test divisors. We can show this simply. If a number is not divisible by 3, for example, it won't be divisible by any multiple of 3 either, since if it were divisible by, say 9, it would also be divisible by the prime factors of 9 (i.e. 3).

The list of primes through 31 is given above. We don't, of course have to test 1 or 2, so we only have to test 10 divisors.

I just timed the C program on my office 486 system. It found 6324 primes between 1 and 65535 in 1.25 seconds as nearly as I can estimate. This time is just for calculating them without printing them out.

Looking at my old programs, I found that to find the primes less than 1000 took some 43 seconds in TSC extended basic on a 6809 system. In the best (read that the fastest) compiled language I had at the time it took 4 seconds. These times are for a 2 MHz 6809 system.

The point of all this is that back then I spent a month of evenings fine tuning a program to make it run faster. Finding primes to the limit of 10,000 still took three or four minutes. Now with a simple, only slightly optimized algorithm I can find primes to the limit of unsigned integers in less time than it takes me to type in the command line. If I want to print them out I can do so in a little more time. The first try program listing using integers and a limit of 10000 is in Listing 1.

I've included the print statements to print out the primes, but I have commented them out. They need to be printed to insure that the program is working properly, but then can be commented out to test the speed of the calculations only.

This just emphasizes my note of last time that it used to be possible to write a program that took a significant amount of computer time to execute. That doesn't seem to be true anymore.

We can do something else. If the program runs too fast make the problem bigger. What if we use long integers? (Have we discussed all the C data types yet? Long integers are 32 bit integers). I set the limit at 65535 using long ints and the program ran 5 seconds. Obviously we are paying a penalty for the long int divides, a factor of about 4. The question then is what are the times for larger limits. Below is a table:

Limit	Number of primes	Time(seconds)
65,535	6,543	5
125,000	11,735	35
250,000	22,045	118
500,000	41,539	340

Obviously, even using a fixed long int arithmetic, the times get longer faster than the increase of the limit. Increase factors for doubling the limit seem to get smaller as the

Listing 1

```
// prime numbers by division

#include <stdio.h>
#include <math.h>

#define TRUE 1
#define FALSE 0

void main()
{
int n, d, prime, count;

count = 3;
// printf("1 2 3 ");
for(n=5;n<=10000;n+=2)
{
prime = TRUE
for (d=3; d*d<=n; d +=2)
{
if (n%d == 0)
{
prime = FALSE;
break;
}
}
if (prime)
{
//        printf("%d ",n);
count++;
}
}
printf("\nnumber of primes %d\n",count);
}
```

Listing 2.

```
// prime numbers by test division

#include <stdio.h>
#include <time.h>
#define TRUE 1
#define FALSE 0

clock_t start_time, end_time;

int is_prime(long number)
{
int inumber, tst_div;

inumber = (int) number;
for (tst_div=3;tst_div*tst_div<inumber;tst_div+=2)
    {
    if (inumber%tst_div == 0) return FALSE;
    }
return TRUE;
}

void main()
{

float timeval;
unsigned long candidate, test_div, prime, count;

start_time = clock();

printf("1  2  3");
count = 3;
for (candidate = 5; candidate < 256; candidate+=2)
{
prime = TRUE;
for (test_div = 3; test_div * test_div
    <=candidate; test_div+=2)
{
if(is_prime(test_div))
{
if (candidate % test_div == 0L)
{
prime = FALSE;
break;
}
}
}
if(prime)
{
printf("%ld ",candidate);
count++;
}
}
end_time = clock();
timeval = (end_time - start_time)/CLK_TCK;
printf("\n\nFound primes to limit of %ld\n",
    candidate);
printf("There were %ld primes.\n",count);
printf("execution time %f\n",timeval);
}
```

numbers get bigger. Doubling the number from 65,000 to 125,000 took 7 times longer. Going double again took about 3.3 times longer, and doubling the last time took about 3 times longer again. I'll add a bit of code to read starting and ending times and run this overnight sometime for a really large limit. Before doing that however, I decided to look harder at the algorithm. I decided to test the test divisors to see whether they are prime, and skip them if they are not (See Listing 2).

I did that, and the time for 500,000 items was reduced to 227 seconds from 340, enough to have made the complication worthwhile. Here is a table of further results:

Limit	Number of primes	Time (seconds)
250,000	22,045	80
500,000	41,539	227
1,000,000	78,499	602
2,000,000	148,934	1621
4,000,000	283,147	4391
8,000,000	539,788	9360 *
16,000,000	1,031,131	25456 *

* with the 66 Mhz processor. Others run with 50

I ran tests for primes doubling the range each time again, and found that at 2,000,000 it took 27 minutes. Based on that and the continuingly smaller ratio of time for doubling the range, I predicted that 4,000,000 limit would take 73 minutes. It timed at 73 minutes and 12 seconds. I ran limit of 16,000,000 overnight. It ought to take between 7 and 8 hours on the 486SLC 50/2 in my office. I've added the software to read the clock and count seconds, so I'll run that test tonight here at home. When I run something that long I like to have something happen on the screen once in a while, but I can't figure out what to do that won't involve a lot of testing and consequently add to the execution time. It would be nice to print a star for every 100,000 numbers tried, but I'd have to test every number! — Later, test run and results reported above. 25456 seconds translates to 7 hours, 4 minutes and 16 seconds.

Having gotten this far I was really hooked on trying to find a few more improvements. I set up the table of primes to use as test divisors and found that it made a considerable improvement. There are some 6500 primes less than 65000. However, 6000 squared is 36,000,000 so the primes less than 6000 will do for all the tests I am prepared to make. There are 781 primes that need to be stored for this case so the array for test divisors and for them squared can be dimensioned at 800. I used 1000.

The first part of the program finds those first 800 or so primes the hard way. It only takes a second, so who cares if that part is not optimized?

Now I can calculate the primes between 0 and 16,000,000 in 9644 seconds! That is 2 hours, 40 minutes, and 44 seconds. After some hard thinking I suspect I see what is going on. As I said earlier, as the range of numbers gets larger and larger the fraction of the numbers that are primes continually decreases. As we calculate for higher and

Listing 3.

```
// prime numbers by test division

#include <stdio.h>
#include <time.h>
#define TRUE 1
#define FALSE 0

clock_t start_time, end_time;
// clock_t is a data type defined
int test_divs[1000];          // in time.h
long testsqr[1000];

void calc_divs() {
unsigned candidate, dvisor;
int prime, i;

test_divs[0] = 3;
testsqr[0] = 9;
i = 1;
for (candidate=5; candidate<8000; candidate +=2)
{
prime=TRUE;
for (dvisor=3;dvisor*dvisor<=candidate;dvisor+=2)
{
if(!(candidate % dvisor))
{
prime = FALSE;
break;
}
}
if(prime)
{
test_divs[i] = candidate;
testsqr[i] = (long)candidate * (long)candidate;
i++;
}
}
printf("there are %d test dvisors\n",i);
}


void main()
{

float timeval;
unsigned long candidate, prime, count;
int k;

start_time = clock();
calc_divs();
printf("finished calc divs\n");
//printf("1  2  3");
count = 3;
for (candidate=5;candidate<16000000;candidate+=2)
{
prime = TRUE;
for (k=0; *(testsqr+k) <=candidate; k++) {
    //   printf("%ld %d",testsqr[k],test_divs[k]);
    if (!(candidate % test_divs[k])) {
        prime = FALSE;
        break;
        }
    }
if(prime) {
    //   printf("%ld\n",candidate);
    count++;
    }
}
end_time = clock();

timeval = (end_time - start_time)/CLK_TCK;
printf("\n\nFound primes to limit of
    %ld\n",candidate);
printf("There were %ld primes.\n",count);
printf("execution time %f\n",timeval);
}
```

higher limits, using only primes for test divisors keeps increasing the efficiency of the program. I suspect that at some point doubling the range will less than double the time required. Supposing I could print 10 prime numbers to a line and 50 lines per page (i.e. 500 numbers to a page), this last effort would result in 2060 pages of printout of prime numbers! The final program is in Listing 3.

The commented out print statements were used for debug. I like to leave them in place for further debug when I make a change. I switched from array notation to pointer notation in the big main loop of the program in the hope that pointer notation would generate more efficient code. The times did not change measurably. Note the mod function which uses the % symbol in C. The mod function returns the remainder of a division. Thus if the division was even, the mod function returns 0. 6%3 = 0. 7%3 = 1, the remainder of the division. I've always thought it was too bad that the originators of C were so "symbol minded" that they had to use the same symbol for more than one meaning in different contexts. The % as we learned before, has a special meaning in a format string for the printf function. Outside of that function it means "mod".

Time for a little reality check here. I calculated the primes to the limit of 1,000,000 with this new algorithm on the 66 MHz system. It finished within a fraction of a second of 4 minutes. Just for fun I ran the same program on a 12 Mhz 286 system. That took 29 minutes and 17 seconds, a bit more than 7 times slower.

There is another algorithm that is considerably faster. It is called the "Sieve of Erastosthenes" named after the Greek mathematician who invented it. Basically, you set up an array of bytes, one location for each of the integers you will be testing. To find all primes less than 10000 requires an array of dimension 5000. You write "TRUE" into all the locations. Then you start eliminating non-primes. The first array location represents 1 and the second 3. If you count odd numbers on your fingers you find that the 5th location contains 9 which we know is a multiple of 3. Starting there and every third location beyond, we write "FALSE". We've essentially crossed out all multiples of 3. Now starting at the location 5 past the contents 5, we cross out every 5th location eliminating the multiples of 5. We continue this process until we have crossed out the multiples of the square root of the limit.

When we're done, we scan the array and use a little arithmetic on the indices to recover the primes. Though it sounds complex it is very fast. Of course there is one drawback. If we want to find primes to the limit of 2 million, we need 1 million array locations. We soon would run out of memory even on an 8 or 16 Megabyte PC. You end up using the HUGE memory model in which things run a lot slower but you can have an array larger than 64K.

I was re-reading some old columns from '68' Micro Journal a couple of days ago and I found that I had some times for a sieve program with a limit of 10000. One person had found them in 0.256 seconds, and another that found them in 0.187 seconds, both on a 2 MHz 6809 system. One reader ran the algorithm on an IBM 3033 and found it to run in under 0.005 seconds. I got hooked and coded a simple implementation of what I described above and it ran and reported 0.00000 seconds. I put a loop around it to make it run 1000 times and it reported 2.97 seconds. That means that the program found the 1230 primes between 0 and 10,000 in 0.00297 seconds, a bit faster than the IBM mainframe of ten or eleven years ago! The erroneous 0.00000 time reported above is because the clock ticks aren't that close together. The time is accurate for the 1000 times loop. See Listing 4.

Do you see a problem here? We count the primes after we say we are done with the calculation. If I wanted to print them out I would have to calculate the value of each prime from the array index of each location containing a TRUE value. In this case the value represented by index n is 2n+1. That is array location 5 represents the number 11. Where does the calculation of primes stop? Is it with the crossing out phase? Should we include the counting phase? Should we include calculating the actual number values (whether we print them out or not)?

Listing 4.

```
// program to find primes to 10000 by sieve method.

#include <stdio.h>
#include <time.h>
#define TRUE 1
#define FALSE 0

clock_t start_time, end_time;
char primes[5000];


void main()
{
float timeval;
unsigned long candidate, prime, count;
int k, l, n;

start_time = clock();
//printf("1  2  ");

// do it 1000 times so we can measure
// the execution time
for(l=0; l<1000; l++)
{
// initialize the array
for (k=0; k<5000; k++) primes[k] = TRUE;

// cross out the non-primes
for (k=3; k<100; k+=2) // outer loop
{
for (n=k+k/2; n <5000; n+=k) primes[n] = FALSE;
} // done generating primes
}
end_time = clock();
timeval = (end_time - start_time)/CLK_TCK;
printf("execution time %f\n",timeval);
// count them
count = 2;
for(k=1; k<5000; k++) if (primes[k]) count++;
printf ("there are %d primes less than
    10000\n",count);
}
```

While the divide method is clearly not as fast as the sieve of Erastosthenes, that method would require 8 megabytes of memory for the array plus space for the program to find the primes to 16,000,000! It appears that it would not be feasible to use it for finding very large primes. We could probably devise a method to find primes from 0 to 100,000, then from 100,000 to 200,000, etc. Actually at the expense of some "bit fiddling" we could define an array of character, i.e. 8 bit entities, and use each bit as a flag. We have 8 * 64K or 512K bits in that 64K byte array. We could use this method without resorting to the large memory models to find primes up to 1,048,575.

Turn a fast computer loose for a week or so (even with the "slow" divide algorithm), and one could really find some big primes. I calculate roughly that I could get to the limit of long integer arithmetic (that is about 2,000,000,000) for signed integers and twice that for unsigned ones) and find all the primes by runing the program for about 600 times as long as it took to find the primes to 16,000,000 (that number for the signed long int limit). That is about 1650 hours, about 69 days. Going all the way to the unsigned long integer limit would take about 2.5 times as long again, or about 170 days. I guess I've found a problem big enough to keep my computer busy for more than a few seconds!

Incidentally, as I understand them, the public key encryption schemes make use of prime numbers in the range of 50 to 100 digits. They use a number that is the product of two huge primes. Finding the key would involve finding the prime factors of this product. It would take a very long time on a current PC, but the way things are going, such things may get to be easy to solve. My system is already slower by a factor of 3 or so than the latest high speed Pentium systems. They could solve the above 69 day problem in 23 days. Next year it will be 8 days, then 3, and then ... That is, unless this continued increase in computer performance finally levels off somewhere.

Speaking of leveling off, some 20 months ago I found a best buy 170 megabyte hard drive for $325. Last week I bought a 420 megabyte for a neighbor for $219! The computer store had advertized a 210 for $169 a week earlier, but they were sold out. "We replenish our stock every week because the prices are coming down so fast we'd lose money if we bought too many." The performance may be leveling off, but the cost certainly is not, and the performance per dollar invested is certainly soaring upward endlessly. Note: January '95: I saw a 730 Mbyte drive advertized for $270!

Whatever computer equipment I buy today will be obsolete in a year or two. Somehow that doesn't worry me a lot. I generally buy two or three year's ago state of the art for about 1/4 of what this year's would cost.

If anyone reading this has any good ideas or suggestions for faster algorithms, as Ross Perot said, "I'm all ears". Well, next time it will be back to C and Assembler plus whatever miscellaneous thoughts come to mind.

The First TRS-80 *continued from page 33*

You can also call the technical services of Radio Shack by dialing 1-800-THE-SHACK (1-800-843-7422).

From the technical rep I spoke with, I learned that you can find out if Radio Shack has a service manual or users Guide in stock by calling 1-800-442-2425. Also I was able to verify the original configuration of the Model 1 which has not been sold in many a year. By calling the 800 number, I was able to learn that these service and information volumes are still in stock for the venerable Model 1 TRS-80 computer:

1) Model 1 User's Guide	$ 2.10
2) Model 1 Tech Manual	$13.89
3) Model 1 Tech List Ref.	$26.99

The tech explained that these would contain full schematics and theory of operation etc. for the computer. The following bibliography may also contain pointers to where much more info on the Radio Shack line of computers may be obtained.

The HOME COMPUTER HANDBOOK
by: Edwin Schlossberg, John Brockman, and Lyn Horton.
1978, Bantam Books Inc.

THE HOME COMPUTER BOOK
by: Len Buckwalter.
1978, Pocket Books Inc.

PROGRAMMING TECHNIQUES for Level II BASIC
by: William Barden Jr.
1980, Radio Shack.

EXPLORE COMPUTING with the TRS-80 (& common sense)
by: Richard V. Andree and Josephine P. Andree.
1982, Prentice-Hall, Inc.

THE TRS-80 User's Encyclodepia (Models I, III, and 4)
by: Gary Phillips and James E. Potter.
1984, Arrays Inc.

THE TRS-80 User's Encyclopedia (Model 100)
by: Gary Phillips, Jacquelyn Smith and Julia Menapace.
1984, Arrays Inc.

THE TRS-80 User's Encyclopedia
(Color Computer and MC-10)
by: Gary Phillips and Guler S. Wright III.
1984, The Book Company (a division of Arrays Inc.)

HOW TO PROGRAM THE Z-80
by: Rodney Zaks.
1980, SYBEX Inc.

Z-80 ASSEMBLY LANGUAGE PROGRAMMING
by: Lance A. Leventhal.
1979, McGraw Hill Inc.

New Feature

Chip and Device

Programming

Program This! The Z80 SIO

By Dave Baldwin

Ed: This is the first installment of a new column in TCJ. Program This! will feature a different chip or device each issue and go through the steps necessary to program it and make it work. The intention isn't to give you a complete application, but to give you enough information and code for you to copy and adapt it to your own requirements.

The Z80 SIO is used in many single board Z80 CP/M systems like the Ampro Z80 Little Board, the Kaypro's, the Big Boards (1 and II), the Xerox 820's and many network interfaces including the old 2 Megabit Lantastic boards. It's a very versatile chip with a number of operating modes. It also drives many people nuts trying to program it.

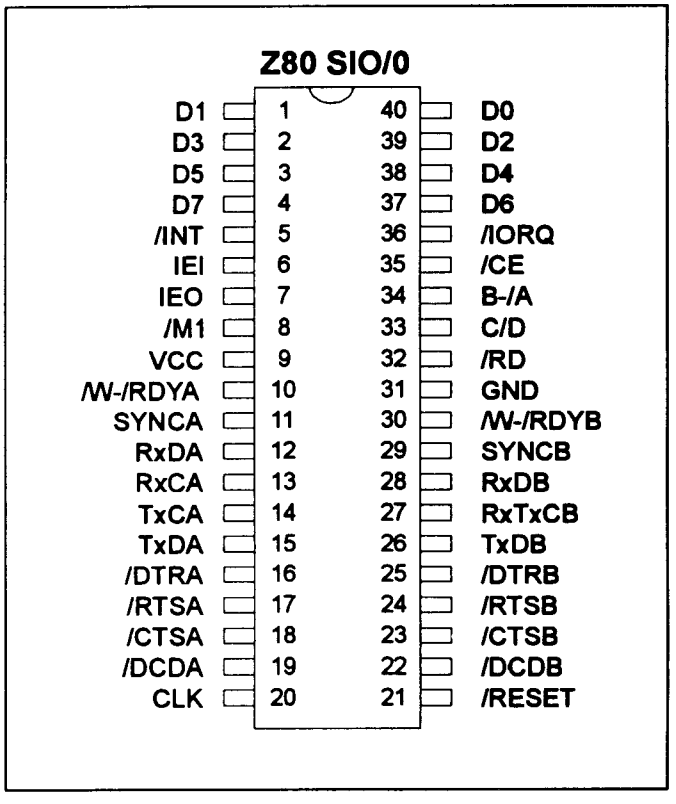
Parts of two programs for 'asynchronous' operation are shown here, one for polled mode operation, and one for interrupt mode operation. The polled mode program, PCDN.Z80, is a complete XMODEM download program that operates up to 19.2kb and has been tested on a Davidge DSB-4000 and a Big Board 1. The interrupt mode program, DIBSIO.Z80, is a demo program that runs at 9600 baud that's been tested on a 2.5Mhz Z80 SBC. The complete source code for these programs is too long to print here, but it will be available on the TCJ Web page and the TCJ/DIBs BBS.

The Simple Facts

The Z80 SIO is the Serial Input/Output chip in the Z80 peripheral family. It includes two complete serial channels including modem control signals and can operate in 'asynchronous' or 'synchronous' modes. Like the other Z80 peripheral chips, it implements the Z80 Mode 2 interrupt structure and daisy chain and it's designed to work properly with the narrow pulses from the Z80 CTC for the Receive and Transmit clock inputs.

The R/T clock inputs are limited to the system clock divided by 5. This means that the maximum 'baud rate' for a 4 MHz part in asynchronous mode with a 16x clock is 50 kilobaud and the highest standard rate would be 38.4kb.

When the SIO was designed, they ended up with 41 connections that they had to fit into a 40-pin DIP package. They decided to release three different versions with slightly different pin connections. The first member, the SIO/0, is



shown. It combines the receive and transmit clocks for channel B together on pin 27. The SIO/1 has separate clocks, but SYNCB connection. The SIO/2 leaves off the DTRB connection. When newer packages came, the SIO/3 and SIO/4 were released. The SIO/3 is in a 44-pin Quad Flat Pack and is available only in CMOS. The SIO/4 is in a 44-pin Chip Carrier (PLCC) and is available in both NMOS and CMOS as are the SIO/0, 1, and 2.

Programmers' View

Each channel of the SIO has two port addresses, one for data and one for control. The control port is set up so you can use the Z80 block output instruction, OTIR, to load the control registers. There are 7 Write Registers (WRx) in the A channel and 8 in the B channel. WR2 is the interrupt vector register and exists only in channel B. For this article, we don't need to do anything with WR6 and WR7 because they are only used in 'synchronous' mode. To access any register other than WR0, you must first put the

register number in WR0 and then send the data for that register. WR0 also has other functions including 'reset channel' and resetting some of the interrupt functions.

There are 2 Read Registers (RRx) for status in the A channel and 3 in the B channel. Once again, RR2 is the interrupt vector register and is available only in channel B.

SIO Initialization

After power-up and to set the SIO into a known state, you have to 'initialize' or 'setup' the SIO by writing all the necessary data to the Write registers. Listing 1 shows a normal initialization routine for polled mode operation and the sequence that's required. The difference between polled and interrupt mode initialization is in the data that is sent. Note that the 'channel reset' command doesn't attempt to address a second register. A 'channel reset' also resets WR0 so the address would be discarded anyway.

Commands other than the 'channel reset' can be combined with the address for the next register. In Listing 1, the 'reset ext/status interrupts' command is combined several times with the addressing for the next register. This is one of the most used commands for the SIO because, in addition to resetting the ext/stat interrupts, it also unlatches the status inputs like CTS and DCD. These inputs are latched on an up or down transition so you can detect changes. Since several changes may have occurred since they were latched, you have to 'unlatch' them to get the current state. For the details of all of the Write registers, you need to get the manual or databook. There isn't room here.

```
LISTING #1
;
;   initialize Z80-SIO to be used in a polled manner
;
SIOINT:
    LD    HL,INITST     ;point to initializing string
    LD    B,INTEND-INITST ;get length in b
    LD    C,SIOCTL      ;sio control/status port
    LD    B,INTEND-INITST ;# of bytes/commands
    LD    HL,INITST
    OTIR
    RET
;
INITST:
    DEFB  00011000B     ;channel reset
    DEFB  14H           ;select wr4 and reset ext/
                        ; stat interrupts
    DEFB  01000100B     ;16x clock, 1 stop bit, no
                        ; parity
    DEFB  01H           ;select wr1
    DEFB  00000100B     ;allow vectored interrupts,
                        ; don't use any
    DEFB  13H           ;select wr3 and reset ext/
                        ; stat interrupts
    DEFB  11000001B     ;receiver 8 data bits,
                        ; receive enabled
    DEFB  05H           ;select wr5, transmit
                        ; controls
    DEFB  11101010B     ;dtr on, 8 data bits,
                        ; transmit enabled, rts on
    DEFB  10H           ;reset ext/stat interrupts
    DEFB  10H           ;and again, just because
INTEND:
; end of initst
```

Polled Mode

In polled mode, the program is written as loop that checks each item that needs attention on each pass. The main loop in PCDN checks first for a char from the console. If there's no char from the console, it skips down to check for a char from the 'modem'. If there is a character available from the console, it checks to see if it is a command character. If it is, it does the command. If not, the character is sent to the modem. Then it falls through to the modem receive test. If there isn't a char available from the modem, it jumps back to the top of the loop and starts all over again. If there is a char from the modem, it's echoed to the console and then the program jumps back to the top of the loop.

The main loop is the 'terminal' part of PCDN. When a '^R' (control-R) is received from the console, the program goes to the XMODEM file transfer routines. In the file transfer routines, there are several loops for the different parts of the XMODEM protocol because the data received gets handled in several different ways.

Polled Read/Write

In PCDN, the serial read and write routines are written with the status tests as subroutines. Listing 2 is an in-line version of the SIO read and write routines. If you need routines that don't wait, change the 'jp nz, xx' to 'ret nz'. The I/O routines used don't change the flags so your calling routine just tests for 0 for success.

```
LISTING #2
; Simple polled receive and transmit routines
;
; Status equates
rx_mask   equ   00000001b   ; "Rx ready" status mask
rx_ready  equ   00000001b   ; recv char available
tx_mask   equ   00000100b   ; "Tx ready" status mask
tx_ready  equ   00000100b   ; transmit buffer empty
;
;   Simple receive char routine,
;   loops until char received
;   exit with received char in A
;
rcvchr:
    in    a,(SIOCTL)    ; get the modem status
    and   rx_mask       ; mask out all but the bits
                        ; we want
    cp    rx_ready      ; modem got a char?
    jp    nz,rcvchr     ; loop until char ready
    in    a,(SIODAT)    ; get the character
    ret
;
;   Simple transmit char routine,
;   loops until char sent
;   enter with transmit char in A
;
trnchr:
    ld    b,a           ; store char in b temporarily
    in    a,(SIOCTL)    ; get the modem status
    and   tx_mask       ; mask out all but the bits
                        ; we want
    cp    tx_ready      ; transmit buffer empty?
    jp    nz,trnchr     ; loop until transmit ready
    ld    a,b           ; char back in a
    out   (SIODAT),a    ; send the character
    ret
;
```

Interrupt Mode

Before you can use the SIO in interrupt mode, you have to set up Mode 2 interrupts and the interrupt vector table. There is no sense in trying to use Mode 0 or 1. The SIO (along with the other Z80 peripheral chips) is designed for Mode 2 and has the hardware for Mode 2 built-in. Listing 3 shows the setup routine.

In interrupt mode, the serial read and write routines have to be written differently. One of the major differences is that you almost always have to use buffers in memory to store the data so the routines have a fixed place to put and get the data. In the DIBSIO demo program, I use 256-byte buffers on a page boundary so I can use simple pointers and counters. The buffers are organized as FIFO's (First In, First Out). Data goes into the buffers at the head (tbhed,rbhed) and is taken out at the tail (tbtal,rbtal). The byte counters (tblng,rblng) are 0 when the buffers are empty and 255 when they're full.

At the beginning of the Interrupt Service Routines (ISRs, Listing 4), all of the registers that are going to be used have to be preserved. Pushing them on the stack is one way. The routines here use the Z80 alternate registers for the ISR's. At the end of the ISR, the registers need to be restored to their previous state, either by popping them off the stack or by swapping the alternate registers back like is done here. The last two things in the ISR must be 'EI' to re-enable the interrupts (they were disabled when this interrupt was acknowledged) and the 'RETI' return from interrupt instruction. The SIO and the other Z80 peripheral chips monitor the data bus and the /M1 signal and when the RETI instruction is seen on the bus while /M1 is active, the interrupt daisy chain is reset so that another interrupt can occur.

Because interrupts can occur at any time in a program, you need to be careful how you access any that has to do with the interrupts and the ISRs.

The first example is in the 'sndlst' routine which is shown in Listing 5. That routine needs to access the status of the SIO and act accordingly. If you allow interrupts to occur while the routine decides what to, the status fetched at the beginning may have changed by the time something is

LISTING #3

```
;
;   setup I/O and interrupts
;
begin:
    di                  ;interrupts off during setup
; clear pointers, counters, and buffers
    xor   a             ;make 0
    ld    hl,6000h      ;start address
    ld    de,6001h      ;destination
    ld    bc,3ffh       ;length
    ld    (hl),a        ;clear first
    ldir                ;clear rest
;set 'I' reg, high byte of interrupt vector
    ld    a,high(ivtab);
    ld    i,a           ;
;set interrupt mode
    im    2             ;mode 2 interrupts
; serial interrupt vectors for port B
    ld    hl,isrwrt     ;transmit isr
    ld    (siobtv),hl   ;init vectors
    ld    hl,isrred     ;recv isr
    ld    (siobrv),hl   ;
    ld    hl,isrerr     ;error isr
    ld    (siobev),hl   ;
    ld    hl,isrspv     ;special vector
    ld    (siobsv),hl   ;
    call  sioint        ;initialize sioB for
                        ; interrupts
; init recv fifo params
    ld    hl,rcvbuf     ;
    ld    (rbhed),hl    ;
    ld    (rbtal),hl    ;
; setup trans fifo params
    ld    hl,trnbuf     ;transmit buffer
    ld    (tbhed),hl    ;start of fifo
    ld    (tbtal),hl    ;save next char pointer
; ready to go now
    ei                  ;enable interrupts
;
```

LISTING #4

```
;+++++++++++++++++++++++++++++++++++++++++++++++++
;   Interrupt service routines for serial port
;+++++++++++++++++++++++++++++++++++++++++++++++++
;
; serial write ISR
isrwrt:
    ex    af,af'          ;use alternate registers
    exx
;
    ld    a,00101000b     ;disable tran int
    out   (sioctl),a      ; until next char loaded
    ld    a,(tblng)       ;
    or    a               ;
    jp    z,isrwr5        ;no characters, abort
    dec   a               ;
    ld    (tblng),a       ;new byte count
    ld    hl,(tbtal)      ;get ptr
    ld    a,(hl)            ;get char
    out   (siodat),a      ;send char
    inc   l               ;point to next, 'L' only
    ld    (tbtal),hl      ;save pointer
isrwr5:
    ex    af,af'
    exx                   ;restore register set
    ei
    reti                  ;end of isrwrt
;
; serial read ISR
isrred:
    ex    af,af'            ;use alt register set
    exx
;
    ld    hl,(rbhed)      ;get pointer
    in    a,(sioctl)      ;get RR0 status
    ld    b,a             ;put in b, not used here
    ld    a,1             ;point to RR1
    out   (sioctl),a      ;
    in    a,(sioctl)      ;get RR1
    ld    c,a             ;put in c, not used here
    ld    a,30h           ;reset error flags
    out   (sioctl),a      ;
;
    in    a,(siodat)      ;get char
    ld    (hl),a            ;save char
    ld    a,(rblng)       ;get length
    inc   a               ;
    ld    (rblng),a       ;save new count
    inc   l               ;point to next available
    ld    (rbhed),hl      ;save next ptr
isrre5: ex    af,af'            ;restore reg set
    exx
    ei                    ;
    reti                  ;end of isrred
;
```

done with it. Matter of fact, I can guarantee that, at some time, it will. To make sure that we're acting on valid information, we disable interrupts (DI) at the beginning of the routine so that an interrupt can't occur and change the status in the middle of the routine. If the status check shows that the transmit buffer is currently empty, we jump to the ISR and fill it which restarts the transmitter if necessary. If the transmit buffer is full, we just re-enable interrupts (EI) and return.

A second example is in the routines that read and write from the buffers. The byte counts stored in memory are used both by the read/write routines and the ISRs. We get the byte count at the beginning of the read routine to see if there is anything in the buffer. If there is, we get the data. Now we need to decrement the byte count because we've removed a byte from the buffer. As before, we need to make sure that an interrupt can't interfere. In this case, we do it with a single instruction 'INC (HL)'. This works because interrupts occur between instructions, not in the middle of them. Except for repeating block move instructions. See next paragraph.

Also note that any access to SIO registers other than WR0/ RR0 takes at least two separate instructions and you must disable interrupts before access and re-enable afterwards. This includes the repeating block I/O instructions because on the Z80 they are interruptible between repetitions.

```
LISTING #5
;
; 'send' loads bytes into fifo
;  if fifo is empty, it restarts transmitter
;  if fifo is full it waits
; uses all registers
send:
    ld    b,a           ;put byte in 'b'
send01:
    ld    hl,(tbhed)    ;next available pos in fifo
    ld    de,tblng      ;address of byte counter
    ld    a,(de)        ;get byte counter
    cp    255           ;buffer full
    jp    z,send01      ;wait for transmit to catch
                        ; up
    ld    (hl),b        ;put char in fifo
    inc   l             ;inc 'L' ONLY
    ld    (tbhed),hl    ;save new ptr
    ex    de,hl         ;
    inc   (hl)          ;inc 'tblng' fifo counter
                        ; inc memory location
                        ; instead of register
                        ; to avoid being screwed up
                        ; by an interrupt
; Required to start transmitter operating again
; Note that interrupts are disabled while we're
;  checking so the SIO can't change states due to
;  an interrupt while we're checking
sndtst:
    di
    ld    a,10h         ;
    out   (sioctl),a    ;reset status
    in    a,(sioctl)    ;get sio status
    and   sndtst        ;mask off
    cp    sndrdy        ;check for tbe + ?
    jp    z,isrwrt      ;send char with isr
    ei                  ;
    ret
;
```

Conclusions

The XMODEM program, PCDN, could be written with interrupt driven routines, but they wouldn't be as simple as the DIBSIO demo program. The data received with the XMODEM protocol has to be handled in several different ways while the interrupt demo just puts the data in a buffer.

The XMODEM protocol has a start phase, a block transfer phase, an end-of-block phase, and error signals that have to be recognized. Only during the block transfer does the data go into a buffer and in PCDN, the buffer is all of free memory which would require more complicated calculations to determine whether the buffer was full and where the next byte would go. In the start and end phases, something else has to be done with the received data. The elegant, but more complicated way to do this is to keep track of the current state or phase, and have the current ISR change the ISR vector to point to a different ISR when the next phase is required.

References

Z80 Microprocessor Family Databook, DC 8321-00
1994, Zilog Inc.

Z80-SIO Technical Manual, 03-3033-01
1977, Zilog Inc.

Feature

All Readers

Morrow Repair

Morrow MD-3P Repair

By Jay Huddleston

In a past user group article the MORROW computer has been compared to the Volkswagen bug. The bug was inexpensive, reliable, functional, and easy to have fixed. As the old VW put on miles and lost value for resale, owners started doing many of their own repairs to better understand the machine and to save money which the repair people were embarrassed to charge for such a devalued machine. Books came out such as the VW REPAIR MANUAL FOR THE COMPLETE IDIOT. This probably derived from the much older work by Izaak Walton called THE COMPLET ANGLER and foreshadowed more recent works such as THE INTERNET FOR DUMMIES. My wife pointed out that all these books had been written specifically for me as could be inferred from the synonyms in the titles. The MD-3 (desk top model) is easily repaired. The MD-3P (portable model) is not.

Just getting into the MD-3P is a bit of a trick. There are two types of cases manufactured for the portable. Unplug the power cord, keyboard cord, parallel ribbon, and serial cord to your printer/modem. Note that the serial and parallel connections are upside down compared to MD-3 desk model because the mother board has been turned with the soldered components inward to offer greater protection since the housing is plastic rather than metal. Use a permanent marker across the seams on one side to show how the cabinet, front and keyboard best mate when putting it back together. Two out of three portables that I've seen have a unit shell housing which is fastened by three small machine screws on the front and three on the back near the seam. Remove these six screws and with the video display facing down, slide the casing up and off. The other style casing is a split design which comes in halves like a clam shell. It is snapped together into the chassis frame by four plastic bayonet type prongs closest to the video screen end of the machine. If anyone has ever had it apart before, it probably has one or more of these prongs broken off. On the back end of the case (handle end) are two slots in the seam on either side where a large straight slot screw driver can pry apart the two halves. This pulls the two male/female plastic connectors in the back end apart opening the clam shell. Then carefully use the same large screw driver to outwardly and upwardly pry the two triangular prongs of each half shell out of the chassis with out breaking the front housing lip into which the clam shell inserts.

At this point the insides are revealed. Let's assume that the machine is face down, i.e. the video screen facing down. So looking down at the metal back, the handle, power plug socket, keyboard socket, on-off switch, and on some models the 115/220 volt selector switch are visible. On either side of the back are screened intake and outflow fans. On some models at the bottom is an access slot for the video logic board's dip switch. This switch is factory set to have the #1 and #8 switches off and the others on. At the top of the back are two twenty-five pin RS-232 male sockets. The outboard one is not used as it is directly wired for the terminal, and the inboard one is the one used for serial printer or modem. A male parallel centronics connector is inboard of the RS-232 sockets and is used for the printer. The male sockets are all on the mother board which is the top side of the machine when running. The mother board has on the opposite or front side of the machine two more parallel male sockets for the disk drives, A: drive outboard and usually blue and B: drive inboard and usually white. On the bottom side of the machine is the video logic board which contains the 6512 microprocessor for decoding the keyboard signals; 68B45 video controller for the vertical and horizontal sync pulses, character RAM address signals, display enable signals, and cursor signals; and the 8251A UART for serial communications with the video terminal. The left side facing the machine when running has the analog video board which contains the high voltage transformer, and adjusting controls for the video screen width, height, and focus. In the front are the two Shugarts SA455 5.25 inch DSDD disk drives. The older clam shell model had 5.25 inch Qume QUMETRACK 142 drives which were belt driven. To the left of these is the video terminal itself which is a ZENITH 1.4 amp 13.0 volt 9 inch amber terminal which corresponds to an ADM-31 type terminal.

The last major modular component is the power supply which is attached to the chassis and the back metal plate. The power supply is the same found on the hard disk MORROW model # PS-640 by SHEO SHIN. MORROW service and sales literature variously says this power supply is a switching

power supply rated at either 80 watts or 100 watts depending on the the literature. According to Silicon Valley Surplus' Fred Whittaker the MD-3P uses the MD-11 power supply without its metal case. He further elaborates that the voltages are the same but the wattage is higher to handle the video terminal. According to the service literature the terminal itself normally has a 40 watt power supply as a stand alone terminal. These power supplies are no longer available, but it shouldn't be hard to find a switching power supply rated at 100 to 120 watts in electronic supplies catalogs. The difficulty is the fitting into the MD-3P. The outer dimensions are 4.37 X 8.75 X 2.5 inches. The actual regulator board inside the support frame is 7.75 X 3.87 inches. The voltages required are +5, +12, and -12 volts. If you replace the power supply, even with one cannibalized from a hard disk MORROW, you will still need to solder the wiring harnesses from the old supply onto the new one.

To remove the back metal plate, remove six machine screws from the motherboard RS-232s and video logic board RS-232. There are four selftapping screws to remove from the power supply. Loosen the nut and bolt combination to release the keyboard socket. If the 115/220 volt switch is present, remove the two screws, washers and nuts to release the switch. Pull apart the the two fan quick disconnect plugs. Gently pry the three wire bayonet power plug from the power supply and lift the back off by the handle. Although the power supply is now immediately before you, it is difficult to remove without clearing away the mother board and video logic board.

At this point a few cautions are warranted. The video terminal can maintain capacitance for a long time of upwards to 20,000 volts. Use a length of copper wire with an alligator clip at either end. Attach one end to the metal chassis and the other to a very small straight slot screw driver. Holding the screw driver by the insulated handle, gently pry the rubber boot of the video tube away enough to insert the screw driver to the high tension wire in the center. This discharges the voltage often with an audible snap and visible arc of light. The printed circuit boards are susceptible to damage from static electricity so touch something metal such as the chassis each time before handling the PCBs. Use a permanent marking pen liberally to mark how things should go back together. It is helpful to use cups which nest into each other such as Dixie cups to put the screws for each assembly as it is removed in order. It is worthwhile to use safety glasses when clipping wires, soldering, or handling the video tube which could shatter.

To remove the mother board unplug the four wire power supply plug. Note that this plug connects to the bottom of the five posts projecting from the mother board labeled # 1 through #4. Pull apart the two wire quick disconnect plug in the middle of the board labelled JR to the reset button. Remove the black wire at post #2 on JP3, the red wire on post #2A on JPA, and the yellow wire on post #1B on JPA. Remove the two floppy drive 34 wire parallel ribbons from the bottom of the board. Note that as you remove them the red line is on the right. The outboard ribbon plug is blue and goes to the A: drive. Slide the mother board up and out of the plastic track supports. Mark on the chassis where the plastic supports go and remove them too.

To remove the video terminal logic board gently pry with a small screw driver the seven wire bayonet plug loose from the the eight prong receptacle on the board at P2 from the power supply. Remove a similar plug on the other side at P1 to the video terminal. Gently slide the board up and out of its plastic track. Mark the chassis at the track base and remove the plastic tracks too. It is very easy to break loose one of the six fine wires from the keyboard to the solder connections on the video terminal logic board at J1. These wires are close to the top of the board and seen from the inside where all the soldered components face, the order from the top is yellow, red, black and below these blue, green, and white left to right. Use 30-35 watt soldering iron and solder braid with solder paste flux to remove the solder from the hole where the broken wires must go. Then carefully touch solder back into place. It helps to use .032 inch thin wire 60/40 rosin core light duty solder. Judicious use of clamp-on heat sinks can help to prevent heat damage to components nearby.

To remove the power supply remove the four wire power plugs from the disk drives, unscrew the four machine screws from the power supply support frame, and lift it out. This may be all the disassembly required if the power supply is at fault.

To remove the disk drives, label and remove the 34 pin ribbon connectors. Remove the two chassis support rails secured by three 1/4 inch hex nuts on each rail. Note that the rails are L-shaped and open toward the inside and up. This clears the way to remove the six front cover 1/4 inch hex screws and remove the front. Unscrew the four machine screws for each disk drive frame support (two on either side) and slide each disk drive out the front.

To remove the analog video board, remove the four screws with spacers and nuts. Use a 3/8 inch open wrench under the dimmer button and gently pry the button loose with a straight slot screw driver from the post to which it is glued. Unscrew the retaining nut and remove the dimmer switch from the body of the machine. Use a small screw driver to gently pry up the rubber boot on the video tube and using a pair of needle nose pliers squeeze the two prongs together at the center of the high tension wire and remove from the hole in the tube. Gently lift the cannon plug off the end of the video tube after having marked its position with a permanent marker. Lift the video board away.

To remove the video tube remove the four screws from the video screen supports and lift away. Note that a grounding wire is fastened to one of the

screws. The machine is completely disassembled. As they say, reassembly is in the reverse order of disassembly as though all your problems are over.

To test the power supply out of the machine, remove the power plug from the metal back of the machine and plug its bayonet plug into the power supply. Use the wall plug cord to plug into that. Clip the ground wire for your multimeter to the the metal chassis of the power supply and the probe into the various harness plugs to check voltages. I like to leave the power switch on and simply plug or unplug the wall cord between settings to be measured. The voltages shouldn't be off by more than 1 volt from the chart below. For testing in the machine (primarily for the analog video board voltages) you must leave the outer case off and measure with the machine running, the ground to the chassis and probe the wire at the board carefully.

UNLOADED POWER SUPPLY VOLTAGES
(measured off the machine)

MOTHER BOARD PLUG		DISK DRIVE PLUGS	
black	0.0 v	blue	+12.8 v
yellow	+5.5 v	black	0.0 v
pink	+13.0 v	black	0.0 v
blue	+12.1 v	red	+5.5 v

TERMINAL LOGIC BOARD (to P2)		FAN 1	
black	0.0 v	hot wire	+13.0 v
2 red	+5.5 v	ground	0.0 v
3 missing	N/A		
4 black	0.0 v		
5 blue	-12.0 v		
6 black	0.0 v		
7 pink	+15.0 v		
8 pink	+13.0 v		

LOADED VOLTAGES TO ANALOG VIDEO BOARD (from P1)

5	+1.5 mv
6	+0.8 v
7	+11.8 v
8	+300. mv
9	+4.9 v
10	+25. mv

There are MORROW service manuals for reference in repairs which go into great detail including schematics. So I won't try to improve upon their work by paraphrasing it all here. This article contains supplementary information not included in the manuals. There are a few more pearls to note however.

On the mother board has a jumper connection at JP4 on the Japanese board (or E5 on the Korean) when jumpered will boot up to an internal ROM diagnostics menu for trouble shooting. There are eight tests. The first three transmit a "barber pole" pattern to screen, serial printer and parallel printer respectively. The fourth checks the integrity of the 8251 USART which requires sticking a wire between pins 2 and 3 and also between 5 and 20 at the serial port plug. The fifth tests the RAM and displays any bad addresses. The sixth does a read and write floppy disk test (so use a blank disk!). The seventh does a floppy disk seek test. The ninth requires an oscilloscope to measure frequency. The last menu item is boot to normal operating system.

While you have access to the mother board it would be worthwhile to add the capacity to use a modem by adding a slip- on connector (as Radio Shack calls them) at the unoccupied jumper connection on JPB. Note that all the jumper pins are jumpered 1A-1B through 8A-8B with a conspicuous gap at 4B-5B. Put the jumper pin there. There is also a jumper pin at 12B-13B which should give nine jumpered connections when you're finished. This has no other effect on your computer and you can't run a modem without it.

It is also very easy to install a real time clock DS1216-E available for about $25 from JDR Microsystems in San Jose, California. You simply pry out the ROM chip on the mother board and insert the clock chip in between the ROM and mother board sandwich style. Complete instructions and all the software needed to run it on the MORROW are available on the MACK DISKETTE supplied by MOR. It runs Z80DOS which is a rewritten BDOS for the MD-3 and the prompt shows the current time in [brackets]. For the MD-3P however there is little space available to piggy back the two chips. I found that the thin metal plate separating the floppy disk drives from the mother board can be marked for the position of the ROM chip against it, cut with tin snips vertically to form a tab which bends the metal inward toward the floppy disk drive about a 1/4 inch and bent again downward to reform the barrier between the mother board and disk drives. This allows the double thickness of chips to have the room necessary to fit comfortably in the machine. You use sysgen to install the new system tracks and you're set. You can return to your old CP/M anytime and the machine will work unaware of the clock. I will supply the diskette to the editor if you can't find it.

One of the two most common maladies of computing with the MORROW is non-working keys on the keyboard (the other is failed power supply, of course). To clean the keys gently pry the key cap up with dental floss or a couple of small screw drivers. Under the cap is a three pronged plastic device which remains in place by way of tiny barbs facing away from the center. Insert a length of paper clip into the prong forcing it back toward the center and while holding it askew to prevent re- hooking the barb, work the other two free as well. Lift it out. There is a little rubber donut under this. The black spot in the middle makes the contact necessary with the two metal strips below it to signal the key stroke. The contacts become corroded and dirty. Clean them with a cotton swab and rubbing alcohol. Dry it and put it back together.

Only a hobbiest could find this article interesting. Just like fishing, Volkswagen repair and the Internet you have to find tinkering worthwhile for its own merit. There is still a lot of tinkering to be done yet don't you tink?

Regular Feature

Contact Listing

SUPPORT GROUPS FOR THE CLASSICS

TCJ STAFF CONTACTS:

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

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

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

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

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

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

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

REGULAR CONTRIBUTORS:

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

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

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

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

USER GROUPS

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

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

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

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

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

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

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

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

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

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

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

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

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

Vancouver Island Senior ADAMphiles, ADVISA newsletter by David Cobley, 17885 Berwick Rd. Qualicum Beach, B.C., Canada V9K 1N7, (604)752-1984.

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

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

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

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

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

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

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

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

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

OTHER PUBLICATIONS

The Z-Letter, supporting Z-System and CP/M users. David A.J. McGlone, Lambda Software Publishing, 149 West Hillard Lane, Eugene, OR 97404-3057, (541)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, 3375 Alma, Suite 263, Palo Alto, CA 94306-3518. An ASCII text file distributed by Internet, issue #1 was July 1993. Home page: http://www.chac.org/chac/ E-mail: engine@chac.org

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

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

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

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

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

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

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

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

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

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

SUPPORT BUSINESS:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The Computer Journal Back Issues

Sales limited to supplies in stock.

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

Volume Number 2:
- Issues 10 to 19
- Forth tutorial and Write Your Own.
- 68008 CPU for S-100.
- RPM vs CP/M, BIOS Enhancements.
- Poor Man's Distributed Processing.
- Controlling Apple Stepper Motors.
- Facsimile Pictures on a Micro.
- Memory Mapped I/O on a ZX81.

Volume Number 3:
- Issues 20 to 26
- Designing an 8035 SBC
- Using Apple Graphics from CP/M
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K
- Extending Turbo Pascal: series
- Unsoldering: The Arcane Art
- Analog Data Acquisition & Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC
- NEW-DOS: series
- Variability in the BDS C Standard Library
- The SCSI Interface: series
- Using Turbo Pascal ISAM Files
- The Ampro Little Board Column: series
- C Column: series
- The Z Column: series
- The SCSI Interface: Introduction to SCSI
- Editing the CP/M Operating System
- INDEXER: Turbo Pascal Program to Create an Index
- Selecting & Building a System
- Introduction to Assemble Code for CP/M
- Ampro 186 Column
- ZTime-1: A Real Time Clock for the Ampro Z-80 Little Board

Volume Number 4:
- Issues 26 to 31
- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside Ampro Computers
- NEW-DOS: The CCP Commands (continued)
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS
- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation: Disassembling Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis & Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program in Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying the CP/M Disk Parameter Block for Foreign Disk Formats
- Starting Your Own BBS
- Build an A/D Converter for the Ampro Little Board
- HD64180: Setting the Wait States & RAM Refresh using PRT & DMA
- Using SCSI for Real Time Control
- Open Letter to STD Bus Manufacturers
- Patching Turbo Pascal
- Choosing a Language for Machine Control
- Better Software Filter Design
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 1
- Using the Hitachi hd64180: Embedded Processor Design
- 68000: Why use a new OS and the 68000?
- Detecting the 8087 Math Chip
- Floppy Disk Track Structure
- Double Density Floppy Controller
- ZCPR3 IOP for the Ampro Little Board
- 3200 Hackers' Language
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 2
- Non-Preemptive Multitasking
- Software Timers for the 68000
- Lilliput Z-Node
- Using SCSI for Generalized I/O
- Communicating with Floppy Disks: Disk Parameters & their variations
- XBIOS: A Replacement BIOS for the SB180
- K-OS ONE and the SAGE: Demystifying Operating Systems
- Remote: Designing a Remote System Program
- The ZCPR3 Corner: ARUNZ Documentation

Issue Number 32:
- 15 copies now available -

Issue Number 33:
- Data File Conversion: Writing a Filter to Convert Foreign File Formats
- Advanced CP/M: ZCPR3PLUS & How to Write Self Relocating Code
- DataBase: The First in a Series on Data Bases and Information Processing
- SCSI for the S-100 Bus: Another Example of SCSI's Versatility
- A Mouse on any Hardware: Implementing the Mouse on a Z80 System
- Systematic Elimination of MS-DOS Files: Part 2, Subdirectories & Extended DOS Services
- ZCPR3 Corner: ARUNZ Shells & Patching WordStar 4.0

Issue Number 34:
- Developing a File Encryption System.
- Database: A continuation of the data base primer series.
- A Simple Multitasking Executive: Designing an embedded controller multitasking executive.
- ZCPR3: Relocatable code, PRL files, ZCPR34, and Type 4 programs.
- New Microcontrollers Have Smarts: Chips with BASIC or Forth in ROM are easy to program.
- Advanced CP/M: OS extensions to BDOS and BIOS, RSXs for CP/M 2.2.
- Macintosh Data File Conversion in Turbo Pascal.

Issue Number 35:
- All This & Modula-2: A Pascal-like alternative 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 experimenter hardware, CPUs in series, software options.
- SPRINT: A review.
- REL-Style Assembly Language for CP/M & ZSystems, part 2.
- Advanced CP/M: Environmental programming.

Issue Number 37:
- C Pointers, Arrays & Structures Made Easier: Part 1, Pointers.
- ZCPR3 Corner: Z-Nodes, patching for NZCOM, ZFILER.
- Information Engineering: Basic Concepts: fields, field definition, client worksheets.
- Shells: Using ZCPR3 named shell variables to store date variables.
- Resident Programs: A detailed look at TSRs & how they can lead to chaos.
- Advanced CP/M: Raw and cooked console I/O.
- ZSDOS: Anatomy of an Operating System: Part 1.

Issue Number 38:
- C Math: Dollars and Cents With C.
- Advanced CP/M: Batch Processing and a New ZEX.
- C Pointers, Arrays & Structures Made Easier: Part 2, Arrays.
- Z-System Corner: Shells and ZEX, Z-Node Central, system security under Z-Systems.
- Information Engineering: The portable Information Age.
- Computer Aided Publishing: Introduction to publishing and Desk Top Publishing.
- Shells: ZEX and hard disk backups.
- Real Computing: The National Semiconductor NS320XX.
- ZSDOS: Anatomy of an Operating System, Part 2.

Issue Number 39:
- Programming for Performance: Assembly Language techniques.
- Computer Aided Publishing: The HP LaserJet.
- The Z-System Corner: System enhancements with NZCOM.
- Generating LaserJet Fonts: A review of Digi-Fonts.
- Advanced CP/M: Making old programs Z-System aware.
- C Pointers, Arrays & Structures Made Easier: Part 3: Structures.
- Shells: Using ARUNZ alias with ZCAL.
- Real Computing: The National Semiconductor NS320XX.

Issue Number 40:
- Programming the LaserJet: Using the escape codes.
- Beginning Forth Column: Introduction.
- Advanced Forth Column: Variant Records and Modules.
- LINKPRL: Generating the bit maps for PRL files from a REL file.
- WordTech's dBXL: Writing your own custom designed business program.
- Advanced CP/M: ZEX 5.0xThe machine and the language.
- Programming for Performance: Assembly language techniques.
- Programming Input/Output With C: Keyboard and screen functions.
- The Z-System Corner: Remote access systems and BDS C.
- Real Computing: The NS320XX

Issue Number 41:
- Forth Column: ADTs, Object Oriented Concepts.
- Improving the Ampro LB: Overcoming the 88Mb hard drive limit.
- How to add Data Structures in Forth
- Advanced CP/M: CP/M is hacker's haven, and Z-System Command Scheduler.
- The Z-System Corner: Extended Multiple Command Line, and aliases.
- Disk and printer functions with C.
- LINKPRL: Making RSXes easy.
- SCOPY: Copying a series of unrelated files.

Issue Number 42:
- Dynamic Memory Allocation: Allocating memory at runtime with examples in Forth.
- Using BYE with NZCOM.
- C and the MS-DOS Character Attributes.
- Forth Column: Lists and object oriented Forth.
- The Z-System Corner: Genie, BDS Z and Z-System Fundamentals.
- 68705 Embedded Controller Application: A single-chip microcontroller application.
- Advanced CP/M: PluPerfect Writer and using BDS C with REL files.

Issue Number 43:
- Standardize Your Floppy Disk Drives.
- A New History Shell for ZSystem.
- Heath's HDOS, Then and Now.
- The ZSystem Corner: Software update service, and customizing NZCOM.
- Graphics Programming With C: Routines for the IBM PC, and the Turbo C library.
- Lazy Evaluation: End the evaluation as soon as the result is known.
- S-100: There's still life in the old bus.
- Advanced CP/M: Passing parameters, and complex error recovery.

Issue Number 44:
- Animation with Turbo C Part 1: The Basic Tools.
- Multitasking in Forth: New Micros F68FC11 and Max Forth.
- Mysteries of PC Floppy Disks Revealed: FM, MFM, and the twisted cable.
- DosDisk: MS-DOS disk emulator for CP/M.
- Advanced CP/M: ZMATE and using lookup and dispatch for passing parameters.
- Forth Column: Handling Strings.
- Z-System Corner: MEX and telecommunications.

Issue Number 45:
- Embedded Systems for the Tenderfoot: Getting started with the 8031.
- Z-System Corner: Using scripts with MEX.
- The Z-System and Turbo Pascal: Patching TURBO.COM to access the Z-System.
- Embedded Applications: Designing a Z80 RS-232 communications gateway, part 1.
- Advanced CP/M: String searches and tuning Jetfind.
- Animation with Turbo C: Part 2, screen interactions.
- Real Computing: The NS32000.

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

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

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

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

Issue Number 50:
- Offload a System CPU with the Z181
- Floppy Disk Alignment w/RTXEB, Pt. 2
- Motor Control with the F68HC11

- Modula-2 and the Command Line
- Home Heating & Lighting, Pt. 2
- Getting Started in Assembly Language, Pt.2
- Local Area Networks
- Using the ZCPR3 IOP
- PMATE/ZMATE Macros, Pt. 3
- Z-System Corner, PCED/ Z-Best Software
- Real Computing, 32FX16, Caches

Issue Number 51:

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

Issue Number 52:

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

Issue Number 53:

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

Issue Number 54:

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

Issue Number 55:

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

Issue Number 56:

- TCJ - The Next Ten Years
- Input Expansion for 8031
- Connecting IDE Drives to 8-Bit Systems
- 8 Queens in Forth
- Kaypro-84 Direct File Transfers
- Analog Signal Generation

Issue Number 57:

- Home Automation with X10
- File Transfer Protocols
- MDISK at 8 MHZ
- Shell Sort in Forth
- Introduction to Forth
- DR. S-100
- Z AT Last!

Issue Number 58:

- Multitasking Forth
- Computing Timer Values
- Affordable Development Tools
- Mr. Kaypro
- DR. S-100

Issue Number 59:

- Moving Forth
- Center Fold IMSAI MPU-A
- Developing Forth Applications
- Mr. Kaypro Review
- DR. S-100

Issue Number 60:

- Moving Forth Part II
- Center Fold IMSAI CPA
- Four for Forth
- Debugging Forth
- Support Groups for Classics
- Mr. Kaypro Review
- DR. S-100

Issue Number 61:

- Multiprocessing 6809 part I
- Center Fold XEROX 820
- Quality Control
- Real Computing
- Support Groups for Classics
- Operating Systems - CP/M
- Mr. Kaypro 5MHZ

Issue Number 62:

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

Issue Number 63:

- SCSI EPROM Programmer part II
- Center Fold XEROX 820
- DR S-100
- Multiprocessing Part II
- 6809 Operating Systems
- Reminiscing and Musings
- IDE Drives Part II

Issue Number 64:

- Small-C?
- Center Fold last XEROX 820
- DR S-100
- Moving Forth Part IV
- Small Systems
- Mr. Kaypro
- IDE Drives Part III

Issue Number 65:

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

Issue Number 66:

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

Issue Number 67:

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

Issue Number 68:

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

Issue Number 69:

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

Issue Number 70:

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

Issue Number 71:

- Computing Hero of 1994
- Small System Support
- Center Fold: Hayes 80-103A
- Power Supply Basics
- PC/XT Corner: Stepper Motors
- DR. S-100
- Moving Forth Part 7
- Mr. Kaypro
- 8048 Emulator Part 1

Issue Number 72:

- Beginning PLD
- Small System Support
- Center Fold: Rockwell R65F11
- Playing With Micros
- Real Computing
- Small Tools Part 1
- DR. S-100
- Moving Forth Part 7.5
- 8048 Emulator Part 2

Issue Number 73:

- $10 XT
- Small System Support
- Center Fold: 640K XT
- IDE Part 6
- Real Computing
- Small Tools Part II
- DR. S-100
- Mr. Kaypro
- PC/XT Corner
- 8048 Emulator Part 3

Issue Number 74:

- Antique or Junk
- Small System Support
- Center Fold: S-100 Power Supply
- Moving Forth part 8
- Real Computing
- AMSTRAD PCW Now
- DR. S-100
- Mr. Kaypro
- Palmtech CPUZ180
- Disk I/O in Forth

Issue Number 75:

- The European Beat
- Small System Support
- Center Fold: Standard Bus I/O
- Moving Forth part 8
- Real Computing
- Embedded Control Using the STD Bus
- DR. S-100
- EPROM Simulator
- High-Speed Serial I/O for the Applicard
- Disk I/O in Forth, Pt. 2
- T9600 Source Code

Issue Number 76:

- Real Computing
- PC/XT Corner
- The European Beat
- Alternatives to the XT
- DR. S-100
- Center Fold: JADE Bus Probe
- PC Time Clock
- PC Security System
- Small System Support
- Floppy Disk Problems

Regular Feature

The Last Word

Forth Day

The Computer Corner

By Bill Kibler

Forth Day 1995

Well Forth Day came and went again, pretty much un-announced and under attended. Each year on a Saturday in November, FIG sponsors a full day event discussing Forth. This year's event happened on November 18 at Dr. Ting's place of work in San Mateo California. I drove down with three others from the Sacramento Forth group.

Our trip down was filled with discussions ranging from politics to Forth and embedded projects. The day started with donuts and social discussions. About 10 AM the official meeting started and short introductions from John Hall, president of FIG, and several chapter leaders, brought attendees up to date on Forth happenings. One visitor from Taiwan said their FIG chapter has over 300 members and is developing a commercial Forth (he can be reached at cchin@simon.pu.edu.tw).

After introductions, the first of several speakers started talking about their projects. Jeff Fox gave an update on F21, the mpu21 version for handling Fiber communications. There are a number of bugs and errors in this prototype run, but Chuck Moore says he has been able to work around them for testing purposes. Speeds are looking good and sometime early in '96 a chip with less bugs will be available. A European company is apparently funding and pushing the project on a fast track of development.

John Hall then talked about a RF signal tracking system using Forth Inc's ChipForth on a Innovative Integration SBC. They use VME style boards with TMS320C31 as the controller and signal processor. This is for UHF signal tracking with high speed switching between horn antennas mounted on a dome.

Dwight Elvey talked about DSP filtering in Forth. He indicated you can do 36Khz filtering on 486/33Mhz system, all using FPC and integer math. He provided sample code and extensive explanation of how it works.

After our lunch break, Alred Tang presented ideas on doing Exact Rational Math using forth. Leonard Morgenstern presented his latest RedTress: a way to achieve balanced binary trees. Al Mitchel from AMR (see back page) talked about using and selecting microprocessors for embedded control. Al compared price to performance and pointed out some of the good and bad features of those he uses and sells.

Mosaic's Patrick Campbell talked about the QED industrial controller which has been getting some good exposure lately. Bob Nash related his experience with using Express and Forth Inc's polyFORTH at SMUD (Sacramento Municipal Utility District). Bob has done things others gave up on and in times most thought impossible, all because of Forth's tools and his learned philosophy. Those concepts he uses are: keeping it small; modular projects; bottom up design; having a vision about the project; and making sure your design and concept is sound. In two years they have put on 6 systems using Express (the industrial control version of polyFORTH, a high level PLC type program) and leveraging their knowledge from the first successful project onto the next one.

John Baumgarner talked about a Forth project he is doing to augment pilots ability to navigate. The idea is to have visual images projected into the pilots view from navigation devices, a form of virtual reality for pilots. Dave Jaffe, our last year's Programmer of the year recipient, gave an update on the Finger Spelling hand. Three companies are investigating development: one in West Virginia to do the hand itself; a college in Israel on using the hand as proteus or hand replacement; as a device to do hand therapy - to move a hand and thus exercise the muscles; possible use in remote or dangerous situations needing tactile operations.

Chuck Moore gave his normal Fireside chat in which he updated and explained more about his progress with the F21. He covered his bad errors and how testing hadn't uncovered them. He discussed some design considerations and his options in dealing with them. Chuck also commented on a recent product release shboom, which may be an illegal copy of his previous work. As usual it was a very enlightening last talk of the day and got us already for the evening dinner.

As one of the day's events, the dinner speaker, Skip Carter, was able to draw over 25 members to a local Chinese restaurant to hear him speak. Skip talked about where he considers Forth is going and some considerations of changes needed. From there a round table discussion continued and eventually spilled out onto the street as we out ran our welcome in the restaurant.

Skip's feelings are based on his personal experience and his two ways of using Forth. He has used and is continuing to use it in embedded oceanographic projects. He came with a working six legged bug like robot

that spawned many jokes. Skip also found he still uses Forth to perform many scientific data projects and as tools to solve special problems. He see's forth making it's way into the new operating systems and is currently looking for Forth for his Linux system.

Skip Carter's WEB site currently holds most of the FIG's library of material, and he is also the librarian for the FIG Forth Scientific Library project. This is a project to collect library routines that are used in the Forth community and are all ANSI Forth compatible. It is the first of many ANSI standard projects.

I asked the question as to what is going on with ANSI and if any commercial vendors are selling one. Skip said he had talked with three vendors at Rochester and two of the three were interested, but later decided not to change from their current direction. That means there is a Forth ANSI standard, but no commercial versions available for use. A public domain version was suggested but nothing more than that.

Lunch Program Contest

I skipped over lunch in order to explain it better here. For many years FIG meetings have had lunch time contests or seminars. This year a programming contest was presented and three groups tried their hands. Bob Nash, Charlie Shattuck and myself comprised one team attempting to program traffic signals. I have included the circuit since I think it is an easy way of learning how to do real life problems using simple tools and devices.

As you can see the diagram is rather simple and shows just how a parallel port can do real I/O. The contest handout says this comes from Dr. Ting's "The Second Course" which is the second book of learning Forth (can be bought from FIG). The parallel or printer port has 12 output bits and 5 input bits.

This simple design limits the crossing signal operations to having both sets of light work the same. By that I mean the north south set of intersection lights will operate the same and force the left turn signals to cycle for both directions before turning on the green.

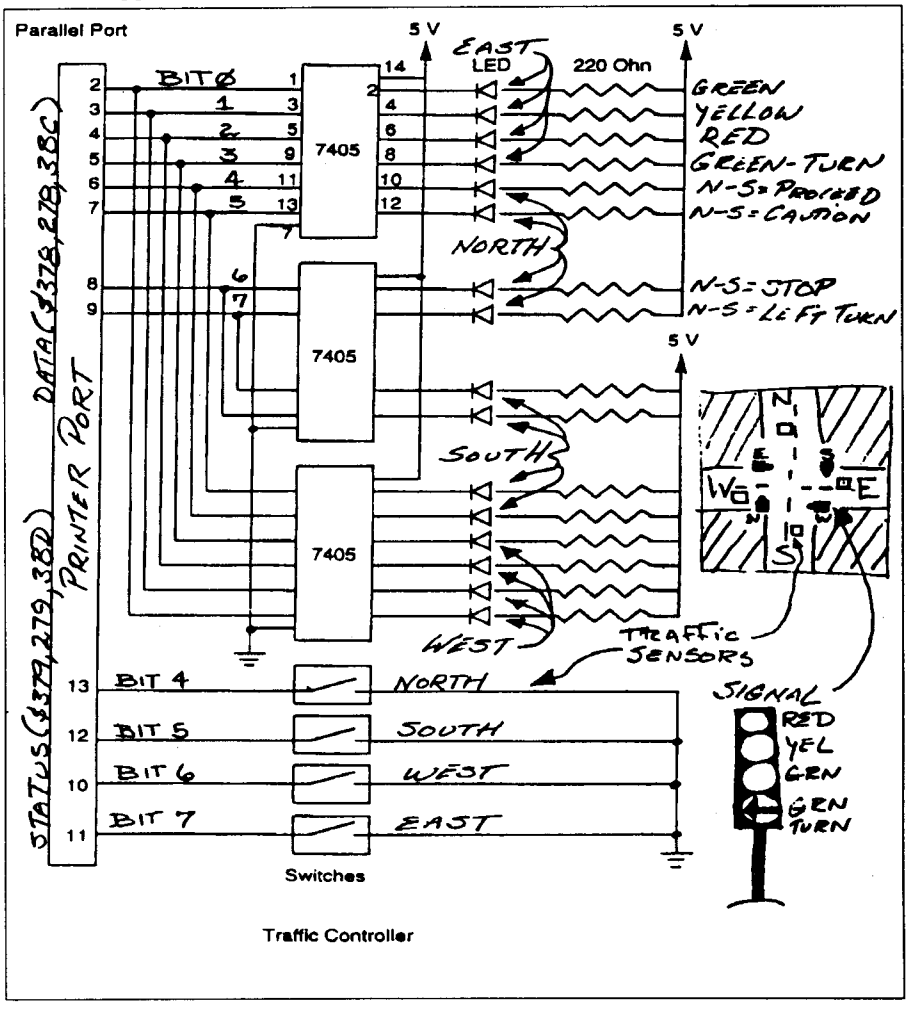
The contest went something like this, turn on all stop lights at power up. With no traffic turn on E-W green and N-S Stop. Then the fun starts, when traffic approaches you must use various amounts of delay with caution, stop, left turn, then go and so on. Basically you make it work as traffic signals normally do, all programmed in less than an hour.

Well our Sacramento group did fairly well against the other two groups. No group got it fully working, but we all got some amount of operation to happen. We had a couple of onlookers who where very much interested in seeing how we went through the design and programming stages. We basically did a bottom up approach. Create a basic set of words that control the numerous possible light combinations. If you analyze the possible conditions it quickly becomes apparent that you have only 8 output commands to the port.

Once the basic commands are set, the next step is tying them together with timing loops and then integrating it all into the possible traffic switch conditions. This last one is where we had the most problems and ran out of time doing. Overall I felt it is an excellent learning project and recommend you try this on your own. We have done similar things at local meetings and find it a great way to get others into Forth and just doing things for fun.

Overall I felt the Forth day this year a bit weak, definitely under advertised, and if nothing else shows a small lack of direction in the FIG organization. A number of new board members were to meet after Thanksgiving and discuss making changes. I have placed a request to hear what went on, but as yet haven't heard a word.

So till next time, keep hacking. Bill.



TCJ CLASSIFIED

CLASSIFIED RATES!
$5.00 PER LISTING!

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

Commercial Advertising Rates:

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

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

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

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

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

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

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

TCJ ADS WORK!

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

FOR SALE: Kaypro hard disk controller cards, WD series for 2/4/10s. Motherboards for all models now in stock. Complete replacement monitors and other new items for your Kaypro needs. Mr. Kaypro, Chuck Stafford. (916) 483-0312, eves/weekends.

TCJ Library Subscriptions

Thank you to our subscriber's that have donated subscriptions to their public libraries around the world.
Paul MacDiarmid
has contributed a subscription to the
Rotorua Public Library
in Rotorua, New Zealand.
This is an excellent way to support *TCJ* and spread the word.