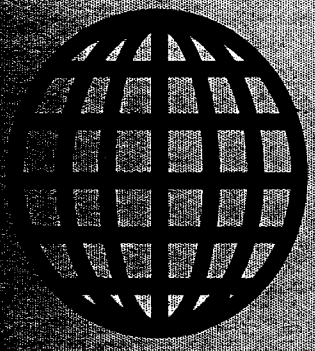
Providing Support Around The World



The Computer Journal

Issue Number 71 January/February 1995 US$4.00

Computing Hero of 1994

Small System Support

Power Supply Basics

Mr. Kaypro

Real Computing

Support Groups

Dr. S-100

Moving Forth Part 7

Centerfold - Hayes 80-103A

The Computer Corner

ISSN # 0748-9331

TCJ - For Having Fun With Any Computer!

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Bill D. Kibler

Technical Consultant
Chris McEwen

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

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

Subscription rates within the US: $24 one year (6 issues), $44 two years (12 issues). Send subscription, renewals, address changes, or advertising inquires to: *The Computer Journal*, P.O. Box 535, Lincoln,CA 95648.

Registered Trademarks

TCJ The Computer Journal

Issue Number 71 January/February 1995

EDITOR'S COMMENTS

Welcome to 1995 and the 20Th anniversary of the Altair computer. We have a special award to present in this issue to a deserving person. There are many 'Heroes" of the computer revolution, most of which have been ignored. They have spent many tireless nights fixing, creating, and enhancing tools, games, and programs that have spawned whole industries of their own.

At *TCJ* we want to make sure that some of those names that helped start the revolution, are at least recognized by us for what they started. In this issue you will find, just after the ever popular Reader to Reader section, our first "Computer Hero Award". As you will read, this person credentials shows how one persons efforts can launch something that becomes many times larger than their original work. The question is, would it have been so good without them? I doubt it!

With our honors past us, we stop next at Rick Rodman's Real Computing section for an update on TCP/IP addressing and more. His last few words conclude at the end of the next article, which is a review by Ken Smyth on power supply fundamentals. You beginning hackers need to read Ken's work, as he fills you in on what keeps those systems running.

Mr. Kaypro or Chuck Stafford follows with a review and detailed accounting of the ROM options and more, available to enhance your Kaypro. If plain old assembly language programming is where you need help, Ron Anderson continues his 6809 assembly "course." If your new to assembly this is a must read, even if your CPU is not a 6809!

Our centerfold will compliment the award this issue. The Hayes "80-103A Data Communications Adapter" was the main force that launched the communications side of using computers. You can see just what it took and also learn a little about talking to serial or COM ports as well.

Speaking of trend setting projects, our IDE interface is moving along with a new possible variation. Tilmann Reh gives us a one page report on his Generic IDE interface project. If you place orders now, you too can have an Z80 IDE interface board. Herb Johnson gives us his request for interested GIDE people in the Dr. S-100 article. Herb talks about the GIDE project and a free system he has learned more about since his last article.

I have been trying to get more embedded project reports, and J. G. Owens offered up his 8048 emulator information for use as a article. Since it is rather long and detailed, I am serializing it with the first part in this issue. You will find his comments most interesting as he points out some problems I recently encountered myself when you attempt to program the 8048 controller. I still ask, why did they do that? For those who mostly program Motorola CPU's Mr. Owens words will make sure you don't change to Intel!

I have been trying to catch up on the pile of articles yet to see the light and this issue two from Walter Rottenkolber made it to the surface. Walter explains his trip down bad math lane as he shows why Division can not always be as correct as we think (sort of a different view of the P5 problem only having to do with programming math operations.) Walter also explains some Kaypro Keyboard considerations using Forth.

Since trying to catch up is talking more space than I have, Brad Rodriguez's part 7 of Moving Forth only has half the code. This is the 8051 version which works well with our starting the 8048 emulator program, and my own current work on 8051's. By only printing half of Brad's code you will need to get it all from GEnie (or FTP) if you can't wait till next issue. It also allows him to take a little time off and focus on his studies. But have no fear, we will see the other parts of the project in later issues (6809 yet to come).

Our Group section is a little bigger as I talk about the Forth Day meeting held in November of last year (1994). I think this is the only publication you will find this information. I learned that FIG's own *Forth Dimensions* is already laid out well into next year. Don't worry about that problem here, I am so busy working and doing paper work at *TCJ* to never get farther ahead than the tortoise in a race with the hare.

That leaves my Computer Corner the remaining words of wisdom, as I talk more about PLC's and especially 8051 type controllers. Seems my work for money is moving from the big boys of PLC to the little ones that fly by the seat of their pants. This is just the beginning of several articles on embedded controllers. I have almost finished the first part where I explain what you get when you give your credit card number to the vendor at hand. Having done so myself several times, I have a number of systems to compare. Rather interesting comparisons in issue 72 coming your way.

Coming your way next issue will be more of our regulars and a few specials still drifting toward the top. I still have quite a few letters backed up in the queue as well. I hope also to get a series of articles that comment on others who might be good candidates for our "Computing Hero" of 1995 award. Also on tap should be some 20Th anniversary of computing articles.

Bill Kibler.

READER to READER

Letters to the Editor

All Readers

MINI Articles

TO: Bill Kibler

Is it possible to subscribe (and order back issue #69) by emailing you my credit card number or would it be better to mail you a check? What a great mag!

I had a long email conversation with Herb Johnson a while back during which I managed to slip in a word about the PDP-11 simulator I've been working on for the last year or so (I never miss an opportunity to plug pet projects), and he suggested that I submit an article on it to *TCJ*. I objected on the grounds that it definitely fits into the "pet project" category and that no one who doesn't share my twisted values would want to hear about it but he said yep, sounds like the right kind of thing for *TCJ*. So I'd like to ask, would you be at all interested in an article about a PDP-11 simulator written in assembly language for the IBM PC? This would be a descriptive article, I'm keeping the source code to myself and besides there are 25K lines of it, a bit much to list. Anyway if you think you might be interested, please let me know if you have any guidelines for writing articles (I've never written one) and what slant you think would be best. I figure no one (in the 8-bit world anyway) wants to hear about the particulars of PDP-11 devices or memory mapping, but it might be useful to cover each subsystem (instruction interpretation, operand fetching, memory mapping, interrupts, delayed I/O events, DMA) in a general sort of way to show how they can fit together and what tricks you need to ensure compatibility w/o sacrificing speed. The PDP-11 differs in all the particulars from typical micros that others might want to write simulators for, but basic things like memory mapped I/O and delayed I/O events and the fact that individual I/O devices have to have their own state machines, should be useful to everyone writing a system simulator.

Re issue #70: it appears that you worked this out but just to confirm it, WRT the discussion of replacing 8" drives on p. 7, yes it's the 5.25" AT-style drives which can generally replace 8" drives. 8" drives turn at 360 RPM and use a data rate of 500kHz (MFM, 250kHz FM), and have 77 cylinders; 5.25" drives in 1.2MB mode (the default) have the same parameters except that they have 80 cylinders, so they're a good match. I use them in my simulator to simulate 8" disks and it works great, and if an 8" drive were substituted (and arrangements made for TG43 etc.) then the media ought to interchange too (I'll know for sure as soon as I get an 8" drive for my CompatiCard IV) with the same programming. Using an AT 1.2MB drive to replace one of the old so-called "quad density" drives (i.e. double density but 96tpi instead of 48tpi) is a little trickier since minifloppies normally use 250kHz (125kHz FM) data at 300 RPM. Some 1.2MB drives (particularly older ones) have an "/RPM" line which slows the motor down to 300RPM when grounded, so if you modify your drive (or controller, or cable) to ground this line then it can replace a normal "QD" drive (such as the Tandon TM100-4). Newer drives may not honor the /RPM line since it's more common in PCs these days to speed the controller up from 250kHz to 300kHz instead of slowing it down from 360RPM to 300RPM, it's cheaper and runs faster anyway (and the controllers shield the difference from software).

1.44MB drives use a 500kHz data rate and turn at 300RPM, which makes them look like an 8" drive with 20% more bits per track besides having 3 extra cylinders like the 1.2MB drives do. Whether this will work with an 8" system w/o BIOS hacking depends... The main problem I can think of is with formatting. Chips like the NEC uPD765 do formatting largely automatically (for better or for worse), and will extend the final gap as long as necessary until it sees an index pulse. So they'll work fine with 1.44MB drives that are programmed like 8" ones, they'll just be a little slower and there will be a lot of wasted space on the end of each track. The WD179x chips however, format using a "write track" command which requires a byte of data (or a token representing marks or CRCs) for every byte (or byte pair anyway) on the track. That means if the track contains 20% more bits, the "write track" buffer needs to have 20% more data, and if the buffer doesn't have that much (it's always a good idea to add an extra dozen or so bytes of gap data at the end of the track with these controllers to allow for minor speed variations) then you'll end up writing a bunch of random memory (some of which may be interpreted as marks) on the end of the track until the index pulse comes around and terminates the write.

I'd like to second the endorsement of the SMC FDC37C65C+LJP floppy controller in Herb Johnson's column. I used one in an IDE/SCSI/FDC/RAM/COM*4 board I built for my old 8-bit IBM PC, all it took (over the buffering and address decoding that's shared with the other peripherals) was the SMC chip and five 150 ohm resistors. The data sheet warns that the chip is picky about ground planes, since I haven't made a PCB for this board (yet) I stuck a piece of pressure-sensitive copper foil to the

underside of the chip and soldered jumpers to the ground pins, before plugging it into the PLCC socket. What makes the chip cool is its support of 2.88MB drives and the fact that the single density mode works correctly (unlike other PC-oriented FDC chips, which either blow off SD mode entirely or else require external connections and/or components). Unfortunately it doesn't generate the TG43 signal required to write most 8" drives correctly, even if you put it in the mode where it generates the equivalent signal (called /RWC for reduced write current) it makes the switch at the wrong track. That can be done in software though if you add an output port for it; it could be done in hardware too but that would get a little baroque, you'd need up/down counters to count steps and clear on /TK00, and some comparator chips (all duplicated for each 8" drive of course).

Speaking of 8-bit IDE, I'll be very interested to see the IDE article in back issue #69. I built an 8-bit IDE interface as part of my PC multi-I/O card and rather than use an LSI chip to handle the conversion between bytes and words, I used four TTL chips and one $1.19 PAL (which could be easily replaced with a handful of "glue" chips, I was tight for board space and had access to a PAL burner at the time). So I can definitely understand people's misgivings about using a $40 gate array unless you're paying for drill time on the PCB.

Keep up the great work! John Wilson <wilsonj@rpi.edu>

Thanks John for filling us in a little better and checks are preferred as I have a better paper trail to follow in case of errors. Are you sure about 3 inch drives being 300 RPM, seems I missed that, sure thought it was 360, but then I have been doing TCJ for a few years and missing some of the finer points of hardware. Problem with being an editor is I only get to see what others are doing and have little time to do it myself.

Which brings me to the article format question for TCJ. I have gotten some recent comments on how TCJ is "doing" articles. So this seems like a good time to review our FORMATS. I try to have a regular stable of writers who speak on a single topic in such a way that they provide specific help, while trying to enlighten all readers with helpful hints they can take to their own platform of choice. The feature articles are suppose to be from people like and I look for a teaching form of discussion, where the topic provides a forum for laying the how and why of a project.

Your PDP-11 or building the I/O board for the pc would make great articles. You might think that an PDP-11 emulator is stretching it a bit, but only if you talked PDP11 and nothing else. But what are you really doing, laying out how to build emulators! That's right, an emulator on a PC platform with all their funny hooks and design limits that must be overcome to make it work. For people working on such projects we tend to take the problem as just another day of programming. For most of TCJ readers, they have little concept of what I mean by "funny" design problems, let alone where one would start on such a project.

So my article guidelines go something like this, tell us why decided to do the project, what the goals are and limits will be, then lay out what has been done and why it went that way or not, and in fact did it go the way you thought. You must remember that programming and hardware design are right side brain projects, or creative endeavors and as such we want to understand your "creative" experiences in hopes that some of the experiences and skills can "rub-off" on us by just reading about your work.

Sounds like a tall order I am placing with you, but plenty of others have started sending me articles and I haven't found a bad one yet. Just remember to keep in mind my objectives for an article, and then write it as if you were telling a good friend who knows little about computers (or enough to get in trouble) what you did and why. Or better yet, get a laptop, visit your favorite pub, and make believe your with a friendly group of peers, tipping a few and telling stories. That is the formula to being a number one writer.

So looking to see why you did the PDP-11 project and thanks for the drive information, John. Bill.

To B.Kibler

I'm the sysop of the CP/M RoundTable and would like to inform you of a special signup deal that GEnie is providing to *TCJ* folks. GEnie is giving *TCJ* members $50 of free usage your first month for just trying the system out.

The CP/M RoundTable has several thousand (5000 roughly) files for the CP/M computers along with an extensive bulletin board covering support for many of the different programs and computers you encounter. We have weekly conferences, and games that all the CP/M people can play online. In addition to myself, our staff includes Jay Sage, Jeff Marraccini, Don Maslin, and Helmut Jungkunz.

Helmut had these words about GEnie.....

When the times are tough, the tough get going. That's the basic message at the GEnie CP/M Roundtable. Meet the unbelievable combination of university people, real hackers, sysops, professional internet users, specialists for rare hard- and/or software - they are all there! Find exactly the same people come together not only for serious discussions of exciting issues in the "small world" of 8-bit computing, but to play games as well, like the all-time favorite HANGMAN word-guessing contest. Naturally, the subjects relate to computing, and it's always great excitement and lots of fun!

Join us all at the GEnie CP/M Roundtable! All you need is your regular equipment to log on and the good will to communicate, even with a smile

```
CP/M RoundTable
(Page 685)

Supporting all varieties of CP/M
ZCPR3, ZSDOS and Z3PLUS

BW.MILLER  Beery Miller   Sysop
JAY.SAGE   Jay Sage       Sysop
Jeff-CPM   Jeff Marraccini Sysop
HELMUT     Helmut Jungkunz Sysop
DON.MASLIN Don Maslin     Sysop
```

RoundTable Conferences every
Wednesday Night at 9:00 pm EST.
1st / 3rd Sunday 4:00 pm EST.

Now, if that was enough to convince you to join, here's the rest of the information you need to try the system out on a $50 FREE CREDIT!!!!!!

If you are interested in joining GEnie, have your credit card handy or you may use your checking account number ($2.00 monthly fee for all checking accounts) and load up your favorite Terminal Emulator program for your modem at 8 data bits, no parity, 1 stop, half duplex.

Dial 1-800-638-8369 (1-800-387-8330 for Canada), and immediately enter HHH as soon as you get the connect signal from your modem.

At the U#= prompt, enter JOINGENIE and then press <RETURN>

After a few seconds, you will be prompted for your screen width followed by the Key Code. You should enter MVC524 as the keycode.

Then follow the simple online instructions and within 48 hours, you should be officially connected to GEnie.

For more information, you may call GEnie Information Services at:

1-800-638-9636 or write: GEnie, c/o GE Information Services, P.O. Box 6403 Rockville, MD 20850-1785.

From: BW.MILLER

Thanks BW for the GEnie information and special. I know that the internet is very hot now, but for many the cost is just to high or too complex to get started. I know I like GEnie because it has just enough services for my limited free time. And yet I can get EMAIL from the internet, in fact any user can by just using the user's GEnie name (like mine B.KIBLER) followed by @GEnie.geis.com. I even send out my internet messages from there. So thanks for the good offer BW and see you soon on the roundtable. Bill.

Dear Bill:

I received the number 68 of *TCJ* and I am still reading it. I have a lot of reasons to write this letter:

1) I am in the search for a Commodore C64 or a C128 with a 3.5" Disk Drive, a Hard Disk, a mouse, compatible printer and the GEOS system. I'm trying to find these items but here in Mexico it's almost impossible; as you know, Mexico started to import and even make computers a few years ago and the preferred computer was and is the PC, then the Mac. Some months ago, I got a seller. He gave me a 64C (but I really wanted a C64). He said I could have it for a week, you know, for testing and to convince myself that it was a good deal. No, it wasn't. The Computer never worked. When turned on, the screen was blank and it changed to white when I removed the Basic or the Kernal chip marked 901227 02. The computer had a "reset button" which was a common momentary-normaly open switch connecting the reset contact (found in the card edge connector called the 'User Port') to ground; no diode protection, just two wires and a switch attached to the case through a hole made with a hot stick. The 64 came with a tape recorder and the manuals. The man wanted N$300.00 (almost $100.00 Dlls.) The box contents label said the GEOS programs and the last communications utility were included but no track of either. So I decided to return all.

Now I have an Atari 65XE (please don't laugh) and its tape corder (XC12) and I want a disk drive. I want to know if somebody can give me the original basic handbook and a schematic. I will buy an Atari 520ST if anybody can sell me one at a good price. If you or any reader have the 520ST please write to:

Aristarco Palacios
Hidalgo 116
Coatepec, Veracruz 91500
MEXICO

I have a good reprint of the 2 parts of a *Radio Electronics* article for making a RAM expansion board for the Timex/ Sinclair ZX81 if someone wants 'em.

2) Talking about Commodore (RIP) and ATARI; in the #68 obituaries, you said the C64 uses chips you see nowhere else. I'm holding the Jameco catalog No. 176 and in the page 9 there is a small list under the title "Commodore Series Integrated Circuits". It includes 23 different chips. You can contact Jameco at

Jameco Electronic Components
1355 Shoreway Road
Belmont, CA 94002-4100

'Bout Atari, well I have an address but without a street nor a PO Box number, does anybody have the complete address?

3) You should to give a look at the "XllR5 and GNU"; "Nova NeXT"; and "Sprite OS" CD ROMS from Walnut Creek, It would be a good article.

Finally, I hope the readers will communicate with me. Sorry for using a Typewriter "to do this letter (and the tons of liquid paper), but when the printers fail, they DO fail.

Very Truly Yours. Aristarco Palacios.
PS: Sorry for my bad English.

Ok Aristarco, hope printing this helps you get the items you need. Yes, I have seen a few vendors selling the older chips for many of the machines. My fear is that they are stripping old units of parts, not an idea I really like, but then I have done somewhat the same thing to keep older units running.

I did look at some of the other CD-ROMs available, but since I am not a Unix person right now, I have no use for their contents. The LINUX CD-ROM set is probably the best one out there as it allows you to boot and run it from the ROM. It is really pretty neat product, one anyone interested in Unix operating systems should have.

One last item is why TCJ is here, mainly to make sure you learn enough about system to tell when someone is selling you something you probably would be

better off not buying. That looks like what happened with the C64C you almost bought.Thanks for the letter and best luck in your search. Bill.

Dear Bill,

I read your comment about 4 years engineering/beginners level/mechanics of stepper motors in # 70. I not quite certain exactly what you were looking for but see if this helps - if it doesn't then simply discard it. Suppose we have a motor with a 5.84 inch diameter (2.92 Inch radius) pulley/shaft on which is wound a wire and attached to the bottom is a 20 lbf weight. The motor is turning at 9.0 rpm and thus the 20 lbf weight is being lifted at 2*pi*2.92/12*9 = 13.75ft/sec. Horsepower is defined as 33000 ftlbf/min or 55O ft.lbf/sec = 745.7 Watts or 0.7457kW. Ft.lbf means force applied multiplied by the rate at which the force is moving. An interesting work/power/energy relationship which not too many people seem to be aware of is that which links Joule.seconds to Watts to work/energy. The Joule is not only defined as the work done by a power of one Watt acting for one second i.e. one Watt.second it is also the work done when a force of one Newton acts through a distance of one metre i.e. work of One metre.Newton and not torque of One Newton.metre. Since one foot = 0.3048 metre, one lbf = 0. 45359237 kg and One kg = 9.80665 Newtons (All precise definitions - no rounding off applied) then we can easily work out that One horsepower = 550 ft.lbf/sec = 550*0.3048*0.45359237*9.80665 = 745.70 meter.Newtons/sec = 745.70 Joules/sec = 745.70 watts.

Our weight is being lifted at the rate of 13.75 ft/min with a force of 20 lbf applied at a radius of 2.92 inches. The work being done is 13.75*20 = 275 ft.lbf/min = 275/33000 = 1/120 hp. We, got the 275 from 2*pi*R*N*F = 275 i.e. 2*pi*(R*F)*N where F = force applied (20 lbf) and N = rpm (9). Look at the quantity R*F it is force * radius and is, in this case 20*2.92 = 58.4 lbf.inches = 934.4 oz.ins = 4.87 lbf.ft which is the quantity known as TORQUE i.e. twisting or turning moment. Note that I use lbf.ft and not ft.lbf as did your article in #70. That is how we English distinguish Torque (lbf.ft) from work (ft.lbf) so, our expression for horsepower was 2*pi*R*F*N/33000 - we could also write this as HP = 2*pi*T*N where T is the torque stated in units compatible with feet and pounds (lbf). This tells us then that if we have given horsepower and rpm (N) then only one value of torque will complete the equation i.e. Torque is irrevocably tied to HP and N. That is important - just last month I saw an article in a popular journal which stated the speed (rpm), horsepower and the torque of an engine. I checked their math and their figures were incompatible. Note, given any two of HP, N and T then the third one is irrevocably fixed.

Suppose that we decided to either double the diameter of the pulley/shaft of our 1/12O hp motor OR to increase the weight from 20 lbf to 40 lbf. It is easy to see that if speed (N) remains constant at 9 rpm then we have doubled the horsepower either by doubling the speed at which the weight is rising OR by doubling the force applied at the periphery of the pulley/shaft. Depending upon the type of motor being used it Is highly probable that performing such an operation would either burn out the motor or stall it. What would we do if wanted to apply this motor (With a max of 1/120 hp) to lifting a 40 lbf weight at 9 rpm ? We have already said that if horsepower and rpm are fixed then so is the torque. We could also write that T = 33000*HP/(2*pi*N) = 4.863 lbf.ft and, since T = R*F then R = T/F = 4.863/40 = 0.12158 lbf.ft = 1.46 lbf.in. i.e. we have to halve the diameter of the pulley which decreases the rate of lift from 13.75 ft/min to 6.875 ft/min and maintains power and torque at the correct values. At this point it is probably worth noting what happens if we use a gear box or other reduction such as small and large pulleys etc i.e. if HP is fixed then the product T*N has to remain constant so if we put a fixed HP output through a 10:1 gear box then the rpm (N) would decrease by a factor of 10 hence the torque would increase by a factor of 10 to maintain T*N constant.

A principal difference between an ordinary electric motor and a stepping motor is that if you try to turn the rotor of a non-energized ordinary motor you will feel little or no resistance to turning. If however you try this with the motor energized then it will turn and you would have to apply the full rated torque to it to stop it. If a stepper motor is energized in the unchanged coil patten of the last move it made then it will remain firmly in one stationary position and you would have to apply it's full rated torque before you could move it by hand. This because the motor is manufactured more like a multiple position solenoid than a motor with a pattern of permanent magnets in the rotor which do not exactly match the pattern of coils around the motor stator. Consequently if one changes the pattern of energization of the coils i.e. some with reverse current and others with forward current then the rotor will take up a fixed position. if you change the pattern of energization to the next required pattern then the rotor will attempt to move either clockwise or counterclockwise to a new position and will thus traverse the defined stepping angle. When the rotor is stationary and the stator energized it is being held In a stationary position by the magnetic force between the coils and the permanent magnet rotor. Since this force is being applied radially to the rotor then we can measure it as a function of the actual force and the radius of the rotor i.e. Force*Radius or Torque as previously defined. This force is known as the 'holding torque' and is analogous to the 'Pull-out Torque' in an ordinary motor. Once we change the pattern of coil energization to an acceptable new pattern with a view to moving the rotor by one defined step (Angle) then the rotor is no longer being held in place but is being pulled, with a certain force toward a new position. This is another value of torque i.e. the applied torque. This may be different to the holding torque but, as far as I am aware usually only the least of these two values (Which is usually the applied torque) is quoted for a particular motor.

Suppose then that our motor with the 20 lbf weight and the 5.84 inch dia shaft/pulley i.e. a torque of 4.87 lbf.ft, were a stepper motor and the applied torque of the motor was 4.0 lbf.ft but the holding torque was 5.0 lbf.ft then the motor would

be able to hold the weight OK but would be unable to move it. In the unlikely event that the holding torque was 4.0 lbf and the applied torque 5.0 then the motor would only be able to move the load and not hold it. Which is, of course, highly unlikely.

If we apply AC volts/current to a coil then the rms current and power taken by the coil are a function of both inductance and coil resistance. If however we apply DC to a coil then the final steady coil current is a function of coil resistance only. The effect of the inductance is to limit the rate at which the current can increase i.e. di = dV/dt.L. An electromagnet becomes saturated at a certain level of current (The 'Knee' of the B/H curve) and increasing the current in a saturated coil does not increase the magnetic field hence we make the reasonable assumption that, since stepper motors are DC devices which may be required to hold steady DC coil currents indefinitely then the manufacturer stated coil current will be somewhere near the most efficient value for production of a magnetic field which does not overheat the motor. If you have to apply 5 volts to each of two coils In a stepper motor to produce 1.0 amps in each then if the motor is not moving you are still using 10 watts = 1/74.6 HP and not doing any work then the efficiency is zero on the other hand if the motor is working flat out and continuously switching the 2 amps between various coils then you may make the assumption that you are getting maximum work out of the motor and this will be something less than 10 watts of work (1/74.6 hp) from the motor. By the same token if you knew the torque at which the motor was operating (Which you would do if it was lifting a 20 lbf or other weight) and you knew the RPM (N) then you could work out how much work the motor was doing from HP = 2*pi*T*N/33000. - transforming that to watts by multiplying by 746 would then give you an indication of efficiency.

Best Wishes, I hope you find some of this useful, Bill Brown.

Well Bill, that sure explains it, I just hope others can understand it. What I was getting at in the note was how many text books require many years of hands on calculus just to see what they are doing. What I like about Henry's testing the motor with coins, is the simplicity and fact that any user could do the same test, college degree or not.

Now don't get me wrong, your letter is not college only material, you do a good job of keeping the discussion simple and without tons of math. What is missing is the little tricks and rules of thumb. You gave a few and I appreciate that, but I think what we need in TCJ is a good article on finding, sizing and using motors, since many of our readers want to tinker with them.

So Bill, Thanks for this great start on getting TCJ's readers into understanding how motors are sized and why. I look forward to more from you and others on this topic of interest to our readers. Bill Kibler.

Dear Bill,

I have been wanting to write for some time now. So here it is. I love your magazine, so keep up the good work. Let me start out with a little bit about my computer experience. I bought my first computer in 1988. A Tandy 1000 SX. Since then I have added expansion up to 640 k, a 20 MB hard drive, a 2400 baud modem, a second 5 1/2" disk drive, and a serial card with a mouse. I feel like I am fairly experienced with MS-DOS. My programming skills lack somewhat though. I am interested in learning FORTH. *TCJ*'s coverage of FORTH is great. Well, I also have a Color Computer with 4 k, a TRS-80 Model III with 16 k, and just this weekend I acquired one possibly two TRS-80 model 4 computers with 64 k. In perfect working order. A friend of mine found them at Michigan State University and they were giving them away. What a find! My Model III disks work in the Model 4, but I am in search of Disks and manuals for the Model 4. If anybody has these contact me at the address below. I will probably buy a CP/M upgrade for the Model 4 from A.J. McGlone of the *Z-Letter* some time in the future. I currently own exclusively Tandy computers at the moment, but I am interested in all classic computers. My resources make me expanding my collection a little difficult at the moment. Well, That's all for now.

Mark J. Kingsbury
Battle Creek,MI
E-mail Mark-J_Kingsbury@fcl.glfn.org
Prodigy STHH95A

OK Mark, looks like you got some good finds. Our aim at TCJ is keeping people like you well informed on your new venture down computing memory lane. I tend to push Forth because of the across platform possibilities it offers, not to mention rolling your own options, but we don't want to be pushed or pushing only one language, since most of your machines run Tandy BASIC which worked for many and in fact started the Gates empire. By the way, did you get your problem with the Z-19 terminal fixed? I know you were looking for help with it and hoping one of our reader could help you out. Enjoy! Bill.

This letter got cut from last issue..BDK.
Dear Editor:

Enclosed is my check for $24 for a subscription to *The Computer Journal.*

Yours is one of the last strongholds of real personal computing. I got my start building the SWTPC 6800 kit and now use a PC with 1 Mbyte RAM and a 286. People think that the old days are gone forever, but that does not have to be so. A few months ago I found an old Model-15 Teletype from WW-2 in the garbage and proceeded to put it in a plastic safety cover so you could see the wheels go round. Next I interfaced it to an old Radio Shack Color Computer 3 (using plastic optical fiber) and displayed it at the local historical society show, held yearly at the grammar school. It made noise and attracted a lot of attention. Kids loved it and adults remembered. I have since inherited A lot of other old Teletypes and have figured how to operate them safely using a standard +/- 12 Volt PC power supply. The whole thing was a fun project and resulted in meeting a lot of interesting people.

Since the Color Computer is no longer

made I decided to buy a single board computer to run the TTY. I obtained a free copy of Nuts and Volts magazine (devoted to buying and selling electronic equipment) and saw a computer in my price range (less than $50). It used a 68HC11, which is part of the Motorola family line and made programming easier. I ended up doing extensive hardware and software modifications and the results exceeded my expectations. It is obvious that other Motorola 8 bit processors could be substituted without much effort. These are some of the most easily understood and programmed processors ever made, and Motorola has a whole line of free cross compilers and other software available. You could even substitute a Z8O, since the board uses static RAM and an external terminal. The idea is to have a low cost bare bones board with reasonable software that an experimenter could use without much care about ruining anything. As a matter of fact, I took a copy of the Ron Cain Small C compiler (developed for the 8080) that had been ported to the 6809 and ported it to the 68HC11 without burning too much midnight oil. I assume that compatible forms of Basic and Forth are similarly available. Just about any computer could be used to communicate with the board. I happen to like the PC because virtually all of them have high speed serial data capability, allowing fast data transfer. The board does not need any special device for this, as even the 1.8 Mhz 6809 used in the Color Computer 3 can operate its bit banger serial port at a 57.6 Khz baud rate, although you do need 2 stop bits.

The foregoing makes me wonder why the small computer community has apparently not come up with a "generic" single board computer. Lots of experimenters and software hackers are still around, so why let all that talent go to waste? A basic, flexible design can go a long way. Look at the PC, which is a clone of the much older PDP8/E. IBM did the right job at the right time and ended up generating a mountain of business. Now that the static RAM bottleneck has been removed it is infinitely easier to make a small computer. Why not design a set of "generic training wheels" and keep on having fun?

One reason I like to use the board is that I can run it from my PC through the serial data port with almost no worries about hurting anything. You can also go a long distance with serial data. I have the board across the room on a workbench. No looking behind the PC to figure how to hook up things. You might think that downloading programs and data to the board is slow, but most PCs' can use their serial ports at 57.6 Kbaud. You do need 2 stop bits if you are going to load programs into a bit banger port, but that is still over 5,200 bytes per second. In 15 seconds you could load a 64 Kbyte RAM, and most programs are a lot shorter. Upload is the problem. To avoid problems with PC interrupts the serial port chip should be replaced with one that has a larger buffer. It might be a good idea to look into using the parallel printer port for data transfer.

Yours: Frank Wilson.

Dear Editor:

I know you are very busy, so I hope this latest letter is not too much of a bother. The reason for this one is that Ronald Anderson sent *TCJ* issue #64 as part of some correspondence we have going. You wrote an article entitled "Small-C?", and I noticed in the discussions that the slow speed and size of Ron Cains original version has carried on into the various public domain offerings based on his pioneering work. There may be hope. Years ago a friend sent me a version of Small C for FLEX that had been modified by John Byrns (17-JUN-8O and 14-FEB-82). I think it is public domain, but my friend does not remember where he got the source. In changing it to work on the HC11 I found that it turned out code that was half the size and twice as fast as that from #309 on the C Users Group CD ROM which I also adapted for the HC11. Even though Byrns version has no extra bells and whistles it is darn fast, and most of the speedup seems to be in the code generation portion. I think it may be possible to use his ideas to improve other versions of Small C.

One of the reasons I am so anxious to determine if Mr. Byrns version is public domain is due to an unfortunate experience with a file called MC141.ZIP from a PC bulletin board. It was an excellent small C evidently written from scratch. One of the documents that came with it indicated public domain use was allowed. The compiler And documentation looked very sophisticated and I was suspicious. Further investigation proved the material was NOT public domain and thus I wasted considerable time seeing if it would help us create a better small-C.

Anyway, neither the C Users Group or Mr. Anderson have heard of John Byrns. If I can verify that his code is public domain I will continue to work with it, but on no account do I want to repeat my experience with MC141.ZIP.

Yours: Frank Wilson.

Thanks for the notice and both letters, Frank. I have been trying for the last two years to build a Z180 based ISA compatible (PC BUS) CPU card for the exact idea you mentioned. A simpler and more basic design which could be used for practical learning projects. Main focus was the cheap cards and bare boards available. Since then however Z-World moved their Z180 product to the PC/104 and now ISA Bus. The bad point is they want $219 for it, but then that was why I was having problems building it, the cost would be a bit high for hacking. Since they are very close to our office, I will be talking with Z-World to see how the product is going and if anyone has ported ZCPR to it.

I am well aware of the problem of public domain tags getting put on products when in fact it is just a sample or demo version. I am considering doing my own TCJ-CDROM and would like to put many programs that have no know owners at present, big problem. Most of what I am interested in is BIOS source code which often had copyrights, but each vendor treated completely different as to whether it was free or cost to acquire.Finding owners is also a big problem in Europe where I understand the original writer seldom put their name in it. That is truly public since it contains no information of origin. what is a person to do? Well like you, check it out, seek more input, give up if someone claims rights to it, or

work on it if no one owns up.

What I find annoying is some people still think you can take someone's work, make a few changes and then sell it as your own. We can see by the major law suits about look and feel, that the concept is only valid as long as no one catches you (or they don't have a good attorney). I am not sure, but you gave the impression that even MC141 looked like a version of small-C that had been enhanced and packaged better? To me that is not original work if that is the case, and only original work can be copyrighted and protected by law. So, Thanks Frank for investigation and on going project to find our readers a better Small-C package, we appreciate your efforts! Bill.

Dear Bill,

Since their delivery I have enjoyed the back numbers of *TCJ* along with the regular issues. I want to complete the collection with the back numbers listed below and also to renew my subscription for another year. Your reminder was timely!

Mention of Gidding & Lewis in your 'Computer Corner' of #27 brought back memories of nearly 20 years ago when I worked at the G&L factory in Arbroath, Scotland though not with the machine tool product. Computers were the coming thing then but were still inaccessible to the ordinary engineers. They were vastly interesting but as relevant to our daily work as astronomy i.e. not a lot! So I continued to be a looker-on when our 360-25 was installed. It became clear that it couldn't do anything useful for a mechanical designer because the Cobol writers weren't engineers & tho' willing they were not too accessible either, in their air-conditioned cells. There were cultural hindrances too. People who had traded internationally for decades may yet have done so from the 'sticks' & have toe prejudices that go with such growth. IBM would hold 'seminars' to tell everyone about this machine. Seminaries is where RC priests are trained (tho' basically any one can be taught at a place so called!) Calvinists wouldn't care to attend a seminar, by association. Today our vocabulary & minds are broader, (Ulster excepted) & such thinking would be classed as bigotry. Nevertheless suspicions about IBM lingered.

Then one day an NC Borer, complete with Post Processor program was ordered by Rolls-Royce. We had supplied R&R from spitfire days (arrogant b*rs!) but we were ignorant about poet-processors. Because of my interest I was sent to Fond du Lac (Wisconsin, USA) to find out & to learn what had to be done. I bought back a reel of magnetic tape & McCracken's book on Fortran. The NEL research place near Glasgow was the only place in Scotland with a capable computer at the time, but it used a different (Univac) tape format. However, they had spare capacity, were keen to help & would train our programmer for next to nothing.

The outcome was that R&R were pleased, said it was the first one they'd had that worked straight away, our new non-DP programmer got a better offer, I moved to another endeavor. That was my only brush with machine tools. As time went by the electrical dept. sprouted a subsidiary doing control systems & electronics & had a teletype link with GEIS in USA. I had about an hour hands-on with this & early BASIC & then back to on-looking for years elsewhere till I got a Z80 machine for book keeping & letters in my 4 man special machines design venture.

Another visit to Wisconsin was to Gilman Engineering at Janesville where I spent an intensive 6 months studying their automatic assembly machine product. Your associate RW Anderson (Small System Support #67 p20) may be aware of them in the balancing machine market. Among other things I was introduced to the flow chart as a medium for conveying the mechanical requirements to the control designer. Thence it was an easy step to have the mechanical man draft the ladder diagram which the controls man could asses & specify his components. Solid state relays were the novelty then but computers were not yet trusted because they might lose their bit in a factory & had to be re-programmed every few seconds.

Wishing you a Merry Xmas & Happy New Year! Sincerely HK Fraser.

P.S. I was particularly interested by B Morgan's REL-Style articles in #35, 36. A third article was forseen in the latter, but I don't see it among the contents of the back numbers. When did it happen?

While on assemblers I remember believing that a library file would be reasonably INCLUDED after a main program file. I learned that AD2500's macro assembler would not do this for me. The supplier suggest that I change them round so that the program file was the one INCLUDED. That worked well! is this peculiar? What is normal?

Well Mr. Fraser, I am glad you are enjoying all the back issues I sent you. I never know if people feel the cost was worth it or not. It seems it is for you, even figuring the overseas mailing expenses in the cost. I am working on a better back issue listing that I hope can be in next issue. It will be by topic and author not issues which is the current case.

The G&L factory was pretty neat place to visit and see them actually using their own machines to make new machines with. Of course I went there to learn about the computers that controlled their machines. Seems they got rights from Motorola to build the 6800 chip as a 16 bit TTL CPU on a separate board. Pretty primitive by todays standards, but then some of these machines are still running and doing perfect jobs.

As to INCLUDE files, you are correct, sounds pretty bad to me. I have however run into some pre-macro processors that will not let you do FORWARD references and that might be your problem. Normally includes are listed in the program main file before any code or references to them. If you did that and still got errors, bad assembler! Just because they sell it doesn't mean it is good or maybe you just found the unfixed bug.

Thanks for writing and reading TCJ! Bill Kibler. *End.*

Special Feature

Annual Award

David Jaffe

1994 COMPUTING HERO

By Bill Kibler

On this 20th anniversary of the Altair introduction (Jan 1975), *The Computer Journal* has decided to start recognizing work done in those early days. Most of the people who took an active part back then, are still at it. At *TCJ* we feel some awards are in order for a very select few individuals who are still producing. We call these people the "un-sung heroes" of computing.

The Computing Hero Award

The requirements for this award are rather simple. First you must have been involved in the early days of computing, about fifteen years ago for a minimum. Next, you must still be contributing to computing, preferably into public domain, or in some way that rewards users more than yourself. Lastly you must be somewhat unrecognized for your past and present deeds.

The concept is simple, find those who have been giving their all to computing for nothing but the normal self gratification of a job well done. We think these people deserve to have their name known beyond a few select circles.

David L. Jaffe, *TCJ*'s 1994 Computing Hero

At the Sacramento Forth Meeting in November of 1994, the guest speaker was David L. Jaffe of the Department of Veterans Affairs Medical Center, Rehabilitation Research and Development Center in Palo Alto, CA. I have been watching Dave and his work for many years. While at the talk, I discovered that Dave's history started at the beginning. With a little research, I determined that David L. Jaffe deserves our first Computing Hero Award.

Dave was invited to talk with our small group and explain his work with Ralph (Ralph stands for Robotic **Alph**abet). Ralph is a mechanical fingerspelling hand for people who are deaf and blind. Until this talk I was unaware of just how long Dave has been into computing. He worked with Ward Christensen and tested the first BBS in Chicago where he grew up. He wrote BYE in assembly language back then and found Forth under CP/M. He did this after the first S100 Modem card came out by Hayes (this issue's Center Fold).

Dave got his EE degree in 1970, followed by a Master's in Bio Med Engineering in '73. When the January 1975 issue of *Popular Electronics* gave us the Altair computer, Dave ordered his $400 computer and has been hooked on computing ever since. He told me it "changed his direction" which has been from being an assistant chief of medical equipment in a Chicago VA hospital to his current research work in Palo Alto. He has been doing rehabilitation work for 15 years and using computers for over 20 years.

The BYE program started out in BASIC and he did the assembly language version. Dave was involved in the first mass purchase of the Hayes modem boards so that he and other members of the Chicago computing group could communicate over the phone lines. At one point he got to meet one of the design engineers from Hayes. Certainly some early hands on work for Dave. You can check his work out on the new CPM CDROM by finding the file DCHBYE55.ASM with a comment by Ward that the code had been adapted from the original work of David Jaffe, Jan 1979. With starting credentials like those, that could make Dave the father of all BBS's, and Ward the midwife?

That gives us Dave's past but what about his current work. To me focusing your direction on using computers to help others is choosing direction over money. Dave works on Rehabilitation Research for the VA in Palo Alto, California. To quote theorganizations letter of introduction "the Center is dedicated to bringing science and technology to bear on the problems faced by physically impaired veterans in their pursuit of living independence."

One of Dave's projects (I found Dave's name listed on 14 papers from the center's yearly report) is Ralph (for Robotic **Alph**abet) which is a fourth generation computer-controlled electromechanical fingerspelling hand. "The device offers deaf-blind individuals improved access to computers and communication devices in addition to person-to-person conversations"

The hand moves it's fingers to perform "fingerspelling". The user places their hand over Ralph and feels the fingers move, thus understanding the letter. Normally a live person is used, but the hand's advantage is the connection into computers and telephone systems.

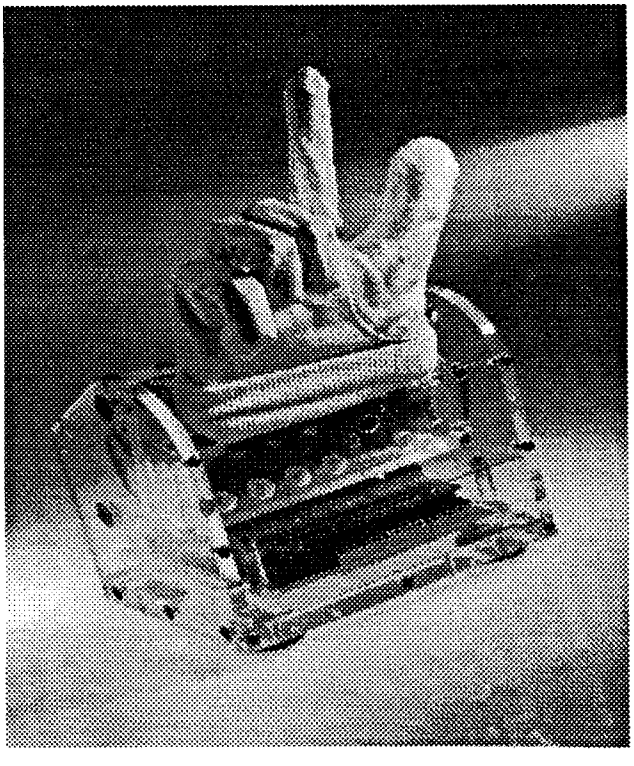
The concept started in 1978 in San Antonio with an all mechanical machine. Being all mechanical it had many limits.

The entire unit was big, noisy, vibrated badly, and broke often.

Dave started on the Ralph project using students from Stanford. Dave is a coach in the Stanford University's Smart Product Design course in the Mechanical Engineering Department. The course ME218, has 2 instructors, 2 teaching assistants, and 32(!) coaches. The object is to have the students develop and package a product or gizmo all in one semester. This type of course has been featured on TV many times, and as Dave said, it is challenging for both the students, coaches, and teachers. That is why he helps, it takes more than one teacher to help all the students get through the project.

In 1988 the student group went to servo motors, a Z80 STDBUS system with an Epson laptop computer for input. The system worked better, but had major problems with mechanics and size. In 1989 Gallaudet University funded similar work with heated metal instead of motors only to have too many failures. Since then, he has moved from STDBUS to a Z-World Z180 computer (about 3 inches square), an I/O board with 9513 PWM drivers (Pulse Width Modulation - a technique for driving motors) on it. This all now fits into the base of the hand.

Fifteen years ago, Dave was introduced to an alternative language, Forth for the Z80. He has been using the same program ever since, and for Ralph. The hand's ROM based Forth software has improved with time, now being able to know what letter was formed "last" and move the fingers so that the next letter is a natural transition from the one currently being done. The laptop has been replaced with a Palmbook (8086 running DOS 5). Model airplane servo motors with feedback drive the linked fingers that have also been greatly improved and simplified. Original finger movement was as much as 1 1/2 inches, now the new mechanical linkage permits the 1/2 inch displacement of a control rod to operate the finger from fully extended to flexed. The software also permits the finger positions to be edited.



RALPH the fingerspelling hand.

The hand is fun to watch, and it has been very interesting to see the design improve over the years. Dave says there are from 20,000 to 40,000 deaf/blind people in the US, of which an unknown percentage are potential users of a commercial version of Ralph. Yet, to date no major money people have come forward to take the project from research to daily use.

From a personal appreciation of what Dave has done, from giving us BYE to the HAND, I want to thank him. There are I am afraid many more programmers and engineers like Dave whose contributions and work go unnoticed. Thus I would like stop ignoring these "heroes" by giving David L. Jaffe, the first of many Computing Hero awards.

If you would like to contact Dave or find out more about Ralph, you can contact him at:

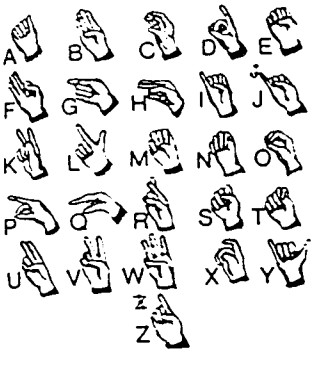
David L. Jaffe
Palo Alto VA Medical Center
3801 Miranda Ave., Mail Stop 153
Palo Alto, CA 94304
415/493-5000 ext 4480
jaffe@roses.stanford.edu

Next year

Since we want to make this an ongoing award, please drop messages to *TCJ* of people you think deserve this honor. We are considering a regular feature that chronicles people like Dave and what they have contributed to your computing enjoyment. Stories, personal recollections, and details are always welcome as either letters or full article.

Bill Kibler, Editor, *The Computer Journal.*

The Manual Alphabet



32-Bit Systems

All Readers

TCP/IP Addressing

Real Computing

By Rick Rodman

House numbers in TCP/IP-land

Viewers of old television programs will certainly recall a notable family which resided at 1313 Mockingbird Lane. Actually, there most probably is no such address. It, like Hazard County of another old TV show, is a fictitious address.

What, you may ask, does this have to do with TCP/IP? Because Tiny-TCP, although a network for small computers, is still *TCP/IP* for small computers, and in the TCP/IP world, every computer needs a unique IP address. In theory, all computers in the world can be joined together and communicate among each other. In practice, of course, things are a little shakier than that.

The Internet has three address classes. People who have obtained addresses get an address of class A, B or C depending on how many machines they say they'll have. People with Class A address have 8 bits assigned, with 24 bits for their 16.7 million computers. People with Class B addresses have 16 bits assigned, with 16 bits for their 65,536 computers. People with Class C addresses have 24 bits assigned, with 8 bits for their 256 computers. In any case, the IP address is 32 bits.

Now here's why the Internet can't actually support 4 billion computers: Most big companies have more than 256 computers, but nowhere near 65,000; and I doubt any company or government has anywhere near 16 million computers. What you're really assigned is an *address space,* and what happens is that you divide it up into *subnets.*

Let's take a realistic example. Suppose you have some machines on a Token-Ring LAN, others on an Ethernet LAN, and others which are running Tiny-TCP and connected through SLIP. For simplicity, we'll suppose that only one machine is connected to both Token-Ring and Ethernet, and all SLIP machines are connected to it also. (I may make some mistakes, so you network gurus be sure to send in your corrections.)

We'll also suppose (for now) that you are using the Class C address 192.9.200.X (more on this later). These numbers separated by dots are in decimal, ranging from 0 to 255. All Class C addresses have 110 in the top 3 bits. So you have an address range of 8 bits or, actually, 1 to 254 (255 is a broadcast address, and 0 means the network itself).

What we'll do is break the 8-bit range into four subnetworks by using the two high bits of the last byte. Thus, 1 to 63 would be Token Ring, 64 to 127 on Ethernet, 128 to 191 on SLIP, and 192 to 254 left unused. This means that your *subnet mask* would be hex C0 or decimal 192, or, in total, 255.255.255.192. The subnet mask is the value which, logically ANDed with the IP address, gives the subnet number. For each machine on a subnet, you assign an address in that subnet's range. One machine is designated as the "gateway" for non-subnet traffic. That machine would have visibility of one or more other subnets.

Notice that message routing has to proceed through these gateways. If a subnet were "broken", with two gateway machines, there is no way to decide which to send the message through. For this reason, if you have multiple machines using SLIP, you should either connect them all to one machine, or make each a subnet unto itself.

You can obtain a network address from Internic, listed below. However, you only need to do this if you're connecting to the real Internet. They'll want to know who you're connecting through, so that national routing tables can be updated. This means that you'll always have to connect through that other person, so you'd better have a pretty permanent relationship with them. This is a rather static design approach for what has become a pretty fluid, dynamic network; we can say the same for the telephone companies with their geographical area codes. Besides, like area codes, the way the network numbers are assigned leads to huge wastes of numbers in some ranges and shortages in others. There are some proposals for kludgy solutions to these problems, such as DHCP, but these are not universally accepted and, in any event, are probably too complex for our little LANs.

For our purposes, as long as we're not connecting to the Internet, we can use any network numbers we like. We do need to follow the subnetworking scheme, however, as it is fundamental to IP routing.

My Ethernet LAN uses the network number mentioned above, 192.9.200.x. This network number is suggested by Internic and RFC 1597 as a number to use for LANs *not* connected to the Internet. Thus, this is a fake address, much like 1313 Mockingbird Lane. It has validity only within the confines of my own LAN. There is potentially a problem if any of these machines should send messages through the gateway machine, through Switched 56, to one of

the client's other machines. However, there is little need for worry.

The reason there's no need for worry is one of the great superiorities of TCP/IP over other network protocols, like Novell's IPX. There is no *broadcasting.*

In NetBIOS (aka NetBEUI) (and I mean the protocol, not the API), machines are assigned names. To "add your name to the net", it's like walking into a room and shouting "I'm John Smith. Does anybody object?" After you wait a few minutes, and nobody objects, you can use the name John Smith. This is a type of *broadcast.* Microsoft's LAN Manager, NT Advanced Server, IBM's LAN Server, and many other packages are built on top of NetBEUI and use this approach.

As long as that room is not connected to any others, that will work fairly well. But now suppose that your room is connected with many other rooms by speakerphone. It could take minutes for your shout to propagate to other rooms where there is already another John Smith. In the middle, some people already know a John Smith, so they don't know who gets John Smith's messages now. Besides, with everybody shouting, you might have to wait a while to make your own announcement, with the speakerphones constantly congested with broadcasts.

IPX uses broadcasts too, except they're from servers out. It can be pictured as a cocktail party in a living room where there are several waiters and waitresses, each shouting his or her name at the top of his lungs every couple of minutes.

TCP/IP does away with broadcasts entirely, except *within* a LAN (a network segment), where a protocol called ARP is used to define the relationship between IP addresses and hardware network addresses. Since there are no broadcasts, you have to know the IP address of a machine to which you're addressing a message. This is no hardship, though, as you assign addresses yourself.

You can picture IP addresses as postal addresses, with subnet numbers identifying streets and the lower part (masked off by the subnet mask) as a house number. The subnet number gets your copy *TCJ* to the right street, and the rest of the address gets it to your house. No shouting is necessary to get mail delivered (usually).

In a larger network you may not need to know another machine's IP address; you can look it up by name in a *name server* (usually referred to as a Domain Name Server or DNS).

Those of you who are setting up Tiny-TCP LANs should begin considering how to structure your subnets. If you use Tilmann Reh's RS-485 bus, the machines will all be on a single subnet; if you use point-to-point SLIP, you may want to connect all of your machines to a machine with lots of serial ports (such as an S-100 machine). Tiny-TCP itself does no routing; I plan to write a simple routing program.

Linux happenings

Linux news is now available using Mosaic, aka WWW, aka HTTP, a graphical interface for the Internet. I haven't experimented with Mosaic yet. It seems that HTTP defines a file format which is something like SGML (Standard Generalized Mark-up Language) used in ISO Open Document Architecture (ODA), but, of course, different. (SGML is a standardized file format for text and graphics which, naturally, nobody uses.) While HTTP is standardized and well-documented, like everything else on the Internet, the people who've written programs for viewing it have taken a nasty commercial turn, and are all shareware with no source code available.

One thing I've always liked about the Internet and its related communities is the freeness of everything. All technical specifications, for example, are free for the asking; software is all in source code form, all free. The Linux world is like that too: You get *all* the source. Not just samples, not just drivers, *all.* As in *everything.*

Here is the HTTP address for Linux news (Washington, DC area):

http://zerosys.tsl.imssys.com/dclinux.html

Littlenet hardware

Tilmann Reh and I have had further discussions of the Littlenet hardware design presented in *TCJ* #69. One important feature is optical isolation of the bus. However, some computers, such as the Scroungemaster, have RS-485 ports already, but without optical isolation. Tilmann writes:

"Those people could:
a) use the non-isolated RS-485. With only one of those at a net, there will be no problem if the YAWT is located nearby. However, there might be a software problem since then the 'FF' code will be sent which isn't the case with my circuit (where the driver gets active *after* the start bit of the 'FF').
"b) use a standard RS-232 instead, and connect to our isolated interface.
"c) isolate the RS-485 in a similar manner like our interface. This is somewhat difficult since they need some direction information which is not delivered by the interface itself (perhaps there are additional handshake lines which might serve that purpose)."

So there you have it. If you decide to bus-connect a lot of RS-485 ports on existing computers, remember that you're sharing signal grounds between them and creating a potentially threatening noise path. I've read about people using interfaces like this to connect to weather sensing boards in their attic; I guess these people never have lightning strikes nearby.

As far as the bus itself, after discussing many, many options, we've settled on using twisted-pair ribbon cable with p-pin mini connectors (DB-9P) connected every so often. This cable usually has a one-inch straight section every foot or so for crimping. This way, you can disconnect computers in the middle without affecting the bus. The bus will be powered by a wall transformer (YAWT) at

Continued on page 16

Special Feature

Beginning Users

The Power Source

Power Supply Basics

by Ken Smyth WA6HDZ

Power supplies are a boring part of a computer, there's no nanoseconds, megahertz's or three-letter acronyms (TLA's) involved; and the latest software may run perfectly fine with one made three years ago. Obviously not a topic you'll see in a mainstream computer magazine that plugs expensive goodies! But your computer stops cold when that power supply dies or misbehaves, so a simple lesson, with a little history, about what's in that box with all the wires coming out is useful.

A power supply is in the computer to do two things: first, to reduce the relatively high voltage AC from the power line to a low DC voltage that can be used by the computer components; and second, to isolate the computer (and computer operator) from any transients or noise on the power line that could interfere with proper operation. The isolation function seems a little trivial for hobbyists, but remember that messing up just a few bits at the wrong time will trash quite a lot of data. (This possibility is very worrisome to financial and accounting users, and the makers of add-on "surge-suppressors" and similar products do quite a good business.) Ideally, a power supply should convert the voltage without losing any energy in the process; or, in other words, be 100% efficient. All but a tiny bit of the AC going into your computer is turned into heat, someplace, and we want that energy to go towards doing something useful instead of being wasted along the way. This isn't just ecology, its practical engineering: energy lost becomes heat, which must be gotten rid of, and the best way to get rid of it is to not produce it at all. Although modern integrated circuits consume less power for a given function than those designed ten years ago, there are more functions on a given area of silicon, and these are running at higher speeds, and as a result computers today require more DC power. This evolution pushes power supplies to become increasingly efficient and continually smaller in size for a given power output.

You'll hear the terms linear and switching used in describing power supply circuitry. Early microcomputers used linear power supplies, the term "linear" referring to a transformer-rectifier-filter circuit topology. A transformer is a device with two (or more) windings made up of wire wrapped around a core of magnetically conducting material, such as iron or steel. The principle of operation of a transformer is very simple: the alternating current (AC) in the primary winding induces an alternating magnetic flux in the core which in turn induces a current in the secondary winding. The two currents will be electrically isolated from one another, and the ratio of the primary and secondary voltages will be equal to the ratio of the number of turns of wire on the two windings. For example: to make a transformer step from 110 volts to 10 volts, the transformer needs a turns ratio of 11:1 from primary to secondary, that is to say, 11 times as many turns on the primary (110 volt) side as there are on the secondary (10 volt) side. Current will be transformed, too, but in the opposite direction. In the example, if we draw 5 amps from the 10 volt output, the input side will draw 5/11 amps from the line. These linear relations give this topology its name! Since a transformer cannot provide any power gain, the product of voltage and current,V times I, is equal for both primary and secondary circuits; in this case, 50 watts. A transformer is the electrical equivalent of a lever, but it will only work with alternating currents. Additional voltages may be obtained by adding more secondary windings, in which case the total primary power is equal to the sum of all secondary powers. All of this assumes 100% efficiency, but small transformers actually do operate in the 80% range, and the BIG ones on power poles can operate at nearly 100%. As the current requirements for a transformer increase, the winding wire size and the size of the core also both increase, which means additional size and weight: two prominent features of linear supplies which are found in many "classic" computers. Except for the voltage regulator, you could change some component values and voltages; swap the silicon rectifiers for vacuum tubes, and find the basic circuit in your 1957 (or '37!)Radio Amateur's Handbook.

The output of the transformer will go to a rectifier to turn the AC into a "pulsing DC" waveform, be shunted by a large capacitor to smooth out the waveform, and finally pass through a regulator to hold the DC output voltage constant over a range of load demands. These output stages, especially the regulator, are where the inefficiencies in linear supplies start to become significant. A rectifier drops at least 0.7 volts, and most linear regulators, almost all of those found in old computer supplies, require at least a 2.5 to 3 volt "headroom" difference between input and output voltage to operate reliably. For a 5 volt/10 amp supply running at maximum rating, this would mean that for 50 (5 x 10) watts output you need to put 82 watts ((3.2 x 10) + 50) into the rectifier, and will consume 100 watts off the AC line. This is only 50% efficient, but the overall approach is straightforward, and can be built

cheaply with very generic parts. The low cost of parts makes linear power supplies still the most cost effective choice for many low power applications. Those small "AC Adapter" black cubes that plug into wall sockets are good examples of bare-bones linear supplies.

S100 bus computers used a variation on linear design: rather than have a large regulator to handle the maximum current that a box filled with cards might need, the main power supply provided unregulated DC at nominal levels of 8 and 16 volts, positive and negative. Each plug-in card then had its own small regulators on board to feed the circuitry on that card alone. These computers were designed before fast CMOS logic was available: the circuits used TTL family logic IC's and NMOS memories and often required two or three amps of +5 VDC per card, along with other voltages. TTL IC's have a relatively constant current drain regardless of clock speed so even at 2 MHz a machine needed a lot of current. (The CMOS logic prevalent today draws current roughly proportional to the clock rate, so a device on a 2 MHz clock will draw about a tenth the current of the same device on a 20 MHz clock.) Cards were also quite expensive, so you were not likely to stuff your box with (gasp!) 64K of memory unless you were quite well financed. Putting the regulators on the cards was a cheap way for the system manufacturers to get around designing costly high current regulators, and also allowed the heat generated by the linear regulation scheme be spread more or less around the cabinet rather than be concentrated in one spot, probably a good idea; but it meant that considerably more of this heat was generated than would be with a different scheme. The cards themselves almost always used the "three-terminal" fixed regulator IC's (LM340-xx or uA78xx series) which are good for about one amp output each if, a big if, mounted with a heat sink or other means to keep the temperature within safe limits. These regulators had a built in "thermal shutdown" function which would turn the output "off" when the regulator got too hot, resulting in one of the classic "sometimes it works, sometimes it doesn't" problems with S100 systems. Cards requiring more current had more regulators, so you could easily run out of space for regulators on a card before you ran out of space for functional IC's. Meaning, of course, that you needed more cards in your computer to do anything! A single packed S100 card could require 25 watts all by itself, with 10 watts of that being burned off by the linear regulators; so a simple computer by today's standards with a CPU, disk controller, parallel and serial ports, ROM card and four 16K memory cards could draw 200 watts, not including the disk drive itself.

For S100 computers to give way to the "all-on-one-board" (examples: Xerox 820, Kaypro II) variety, power supplies had to get smaller and more efficient, so designers looked to "switching" technology. Switching supplies achieve high efficiency, low weight and small size at the expense of circuit complexity by using a transistor to turn the input current on and off at a rate much faster than the 60 Hz line frequency so that energy can be stored as magnetic flux in a relatively small inductor. The charge and discharge of this inductor is controlled by regulator circuitry to give the correct output voltage under a predetermined range of load current requirements. A switching supply running off the AC power line usually has some sort of transformer to provide isolation, but due to the high switching frequency this is a much smaller and lighter part than the 60 Hz transformer needed in a linear supply, and it may also function as the switching inductor. Switching supplies may also be designed to run off of straight DC which makes them the obvious choice for battery powered equipment. This approach is very "manufacturable" in large quantities because the complex analog circuitry that controls all of this can be placed on one or two IC's. All the switching and inductive discharge creates more electrical noise than a linear supply. This is not a problem for computers or other digital circuitry but it means that extra filtering is needed when switching supplies are used for critical analog applications, negating many of the size and cost advantages. The added circuit complexity makes switchers less economical for low power applications, but the improved efficiency and smaller size compared to linear supplies has made them the predominant technology for medium and high power applications in computers. As practical switching speeds increase with better transistors and control schemes, the magnetic components needed for a given power level shrink and so the size of these supplies gets smaller. The reduction in size of supply components and the increase in DC current requirements for PC and Macintosh style computers have roughly balanced themselves out, so the 250 watt supply in a Pentium machine fits into the same or less physical space as that 63 watter in the old IBM-PC. This simplifies replacement and upgrading. The improving technology is also constantly making older parts obsolete, so finding a replacement for that ten-year-old IC or transistor that went bad can often be a problem. With new supplies selling for around thirty dollars, its often more practical to discard a defective one rather than pay someone to fix it.

PC clone supplies are often spec'ed in terms of watts, like "150 watt supply" or "220 watt supply". This is a handy way to express the total output power available if all the various voltage outputs (for PC clones: +5,-5, +12, and -12 volts) are delivering their maximum rated current, but it doesn't say anything about how much AC line power it actually consumes to produce that output, or how that power is distributed between the output voltages. Modern logic IC's used in today's PC's are designed to run off of + 5 VDC supplies exclusively, with a few also requiring +12 VDC, but earlier devices often required -5 and -12 volts. Special interface cards for controlling or monitoring external devices and disk or tape drives may also require more current on different voltage lines. All of this complicates the process of replacing a power supply on an old/weird computer, so in those cases; it is best to ignore the "watts" label and make sure that the new supply provides at least as much current on all voltage outputs as the old one did. Replacing the supply in a PC clone or Macintosh computer is mostly a matter of being certain that the new supply fits in the old box and has at least the same "wattage", fortunately the

power supply case sizes for PC's are standardized to the point where this is not a big concern. Replacing a supply in an old or non-standard computer is more of a problem. In these cases, look for a new supply that has the same voltage outputs, and the same or higher maximum current on each voltage. If you've added a lot of things on to your computer, allow extra current for them; many early computers had very little margin left in the hardware design by the time the accounting department got through with it!

Testing a suspect supply requires a large resistor to simulate the load of the computer, and a voltmeter. (Good voltmeters can be had for about $30 to $50 at Radio Shack if you don't have one on hand. For this use, the exact type isn't important.) The load resistor is important since many switching supplies require a certain load on the "primary" (one providing the most current) output for the regulator to work properly: without a minimal load a 5 volt output may go up to 15 volts or more! Use Ohm's law to pick a resistor that will draw about 20-40% of the rated current on the 5 volt output, and not overheat with the power you are dissipating. Remember the formulas: R=V÷I and P=VxI. To draw 5 amps at 5 volts, for example, you will need a 1 ohm resistor rated at 25 watts. (If you don't have a power resistor available, household incandescent light bulbs will work. A 100 watt bulb is around 10 ohms when cold. I haven't tried it, but a non-halogen auto headlamp should work as a good power supply load. Just remember that light bulbs greatly increase in resistance as they heat up!) Connect the load across the output, and plug the thing in. If you hear a loud buzzing or humming sound, unplug it IMMEDIATELY and check the connections again. Measure the voltage across the resistor terminals, and between the other voltage outputs and the common ("ground") terminal. If the outputs are all within 10% of the rated voltage, your supply is probably good. Switching supplies often have a small glass fuse buried inside; if the supply is totally dead (fan not running) that's worth checking first. Replacement fuses are also a Radio Shack item, just be sure to get the same amperage rating as the old one. If the supply "went down in flames" (you'll know what I mean when you open one of these!) its time for a new one. Switching supplies are not easy to repair, for reasons explained previously, but sometimes the problem is something obvious. Linear supplies, on the other hand, are pretty easy to diagnose if you have some knowledge of how they work.

For more information, you can look up the application notes in the Motorola, National, or TI handbooks on voltage regulator IC's. Radio Shack has a small book titled Building Power Supplies which also contains good basic information, as well as circuits for building small supplies using Radio Shack parts. Current editions (not the 1957) of the aforementioned Radio Amatuer's Handbook have chapters on power supply theory and construction; check out your local public library or ham radio store for this one. Knowing a little about how a power supply works can save you a little time, frustration, and sanity; especially when its 10:30 PM and things suddenly go pfft.

Real Computing

Continued from page 13

one end, supplying unregulated power; each interface will have a small regulator (78L05).

The pinout will tentatively be like this:

Signal:	DB-9 pin:	Wire number:	
GND	1	1	
GND	6	2	
		Note: Pins 1 & 2 are a pair	
GND	2	3	3 & 4,
VCC	7	4	5 & 6 etc.
A	3	5	
B	8	6	
GND	4	7	
VCC	9	8	
GND	5	9	
		10 N/C	

Of course, you can still use SLIP using point-to-point links if you like. TinyTCP will not see any difference. Using the bus, you will receive messages that are intended for other machines; these are easy to ignore. Any hex FF byte which may come in between messages is easy to ignore.

Next time

Sun used to have a slogan, "The Network is the Computer." How do you exploit synergies of multiple computers working together? We'll examine some approaches for keeping all of our tireless workers busy. Plus, some news relating to JPEG, the International JPEG Group, and the unnecessary evil of software patents.

Where to call or write

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

IP addresses: hostmaster@internic.net
Network Solutions, InterNIC Registration Services
505 Huntmar Park Drive, Herndon VA 22070. Fax: +1 703 742 4811

Mr. Kaypro

By Charles B. Stafford

Regular Feature

Kaypro Support

Product Options

In The Meantime...

Much has happened since last we met, the price of new processors has been dropping steadily, electronics manufacturing has increased its migration back to the U.S., and the American Populace has spoken. This is still being written on a K-28, a model only manufactured here at Bullmoose Ironworks & Woodbutchers. Judging by the mail, there are several new readers who are still in the dark about Kaypro models and modification possibilities and difficulties. We will therefore pause in our continuing series of transmogrification construction projects and address the subject of what can be done, cost effectiveness and difficulty.

While Back at the Ranch...

As of last issue, the Personality/Decoder Board project was for the most part finished. Since then BI&W has built two more and it gets easier each time. Models have been done with both direct plug-in and ribbon cable connections, and perform equally well. For those who do not require a hard drive, but would like the option of multiple high capacity floppy drives, the decoder for the *MicroCornucopia* Rom is planned for the near future, in two versions, one on the mother-board and one outboard like the last project. BI&W would also like to hear from those who are either in progress, or who have finished the last project, especially if you have suggestions on how to make it easier. A side note, the prototype board and all the components for the Personality/Decoder board project were procured from HSC Electronics here in Sacramento. There must be other suppliers, but if there is enough demand, BI&W will package and ship "project parts kits".

When a Kaypro is not a Kaypro...

When the Kaypro was originally conceived, it was intended for technicians, who, it was thought, would take it into the "field" and would most probably want to modify it for their own special purposes. Thus it was born with the aluminum case, which has become a hallmark of sorts. There have been, over the years, basically three models; the "II/IV", a two floppy configuration; the "10", a hard drive version and the "16/16II", an MS-DOS (gasp) variation. Within these basic groups, there have several variations, mostly designated by year and/or the letter "x". All of the with the exception of the very early "KayComp" and the very early"K-II"s use a standard IEC power cord, and all except the "Robie" and the "K-4x" use standard double sided double density diskettes. The K-IIs however only wrote on one side of the diskette.

The early 2 floppy models, usually designated as 83s ran at 2.5 MHz, while the later ones (84s) run at 4.0MHz, thus suggesting a possible hardware "upgrade". The fourth model issued,(after the KayComp, the K-II, and the K-IV,) was the K-10, which had a 10 Mb hard drive, but with the exception of the hard drive interface, essentially the same motherboard, and processor, thus suggesting another hardware "upgrade". Wonder of wonders, this same model also included rudimentary "graphics"(perhaps another hardware "upgrade" ?) Then came the age of "Standardization" and several attempts at the "Universal" motherboard using which, any model could be built. Most of the 84 series is based on this concept, which explains the unused outlines and solder pads in many of your machines. There were also several attempts at a "universal" BIOS, resulting in CP/M 2.2d,f,2 versions of g,h, and u. There is an easy way to "standardize" if you have multiple machines of the same year, however.

More on that later.——

Meanwhile back at the firmware ranch, things were not standing still. Several very ingenious and talented people had discovered what they considered shortcomings in the original BIOS and were busily concocting, producing, and marketing new firmware (i.e. new EPROMs). Among these were Advent Products, *MicroCornucopia,* and Barry Cole to mention but a few. They issued "new and improved" versions of both the "monitor rom" and the "character rom". Perhaps the first was the *MicroCornucopia* character rom for the early K-IIs. The early computers, Kaypro included, had character sets thar lacked "true descenders", you know, those little tails on the lower case "y", "p", & "g". As issued, the entire character was "above the line" and this new rom fixed that. The Kaypro "as issued" also had a Greek character set in the character rom for scientific purposes, and the next new rom, as "screen blanking" became popular, substituted blank spaces which were called, to become a quick easy way to handle that little chore.

As you can see, the possibilities are limited only by your own ingenuity, and/or your pocketbook and are generally di-

vided into three categories; hardware, firmware, and software.

Hardware:

Reset button and brightness control

The most basic and easiest changes are to relocate the "reset" button and the brightness control to the front panel for easy access. Neither change requires any real electronics expertise, exceptional manual dexterity, or significant amount of money. The main ingredients are 1. a way to make a 3/8" hole, 2. a small amount of small stranded insulated wire (for the brightness control), 3. some duct tape to capture the metal shavings, and 4. a modicum of care.

K-II to K-IV conversion

The 83 K-IIs "as issued" stored data on single sided doudle density diskettes (190kb nominally 200kb hence the II designation), while the K-IV, essentially the same machine, used double sided double density diskettes (390kb nominally 400kb, hence the IV). It didn't take long to figure out that with diskettes at $1.00 each, a II to IV conversion would pay for itself in short order, if it could be done easily. Again *MicroCornucopia* to the rescue. This conversion has been further refined and is discussed at length with detailed instructions in Issue 63 of *TCJ*.

Again, no real expertise is needed, just a few IC sockets, some wire or Radio Shack test clips, and a K-IV monitor rom or suitable substitute (*MicroCornupia*, TurboRom, Barry Cole, etc).

Quad density drives

There are always those who can't get enough, and for them Quad density drives were invented. They mount in the same space as a 390kb half height drive and have a 794kb capacity. They also use about 1/3rd the power that the original 390kb full-height do, and four of the will fit into the same space as two of the originals, increasing the on-line storage capacity by a factor of four to 3.2mb and decreasing the power demand by 1/3rd. If it sounds like a great modification, that's because it is. Physical installation is easy, just measure carefully, use the duct tape, and make 8 small holes in the drive enclosure. Then add two more connectors to your drive ribbon cable (they're available from Radio Shack, if you have nowhere else to turn) and use 2 "Y" cables for the power. YOU WILL NEED A NEW MONITOR ROM AND A DECODER BOARD OR MODIFICATION to use these drives, however. The original BIOS in the monitor rom does not know about Quad density and there are only two drive select lines implemented in the Kaypro design. The new rom takes care of the former, and the decoder board or modification takes care of the latter.

Hard-drive Conversion

When the K-10 was issued with a 10mb hard drive (more power!), the more competitive folks with IIs and IVs started looking for ways to keep up with the Jonses, without throwing away their previous investment. Fortunately, there were several respondents to this need. Among them were MicroSphere in Oregon, Advent in southern California, SMT in Texas to mention just a few. One solution involved an outboard drive and enclosure with separate power supply that connected to the parallel port and allowed your printer to be connected as well. The Advent and MicroSphere solutions involve a "daughter" board which plugs in between the Z-80 and its original socket. These solutions allow you to keep the machine "portable" (luggable ?) as a single unit. The MicroSphere Winchester Connection, and the Advent Host Interface board are still currently available in limited quantities. Both require use of a Western Digital 1002-05 or 1002-HDO hard drive controller, also available in limited quantities. The Advent has a Real Time Clock option, the MicroSphere does not. Both allow booting from the hard drive if the monitor rom has that capability. The MicroSphere comes with drivers that allow booting from a floppy using the stock Kaypro rom, installing the drivers, and then accessing the hard drive. The Advent Host Interface board requires use of the Advent TurboRom. Installation is straight forward, the mother-board is removed to allow physical mounting of the controller to the side of the drive housing. A "Y" cable is used to split the power from one of the floppy drives for the controller, and another is used to power the hard drive itself. The hard drive is usually mounted in the space left empty when the original full-height floppies are removed and half-height drives installed.

Speed-ups

As other machines entered the market, and tinkerers became familiar with the inside of the Kaypro, there was a cry for "More Speed". Both Legacy and Advent responded with add-in boards and so did folks at MicroCornucopia. The products from Advent and Legacy, while elegant, were pricey. The modification designed by *MicroCornucopia*, in several variations, was fairly easy and cost practically nothing. It appeared in 4.0MHz, 5.0MHZ, and 7.0MHZ versions, and the 5.0MHZ has been re-engineered for ease of construction and installation and was published in *TCJ* issue 61.

External Monitors

When demonstrations of software and hardware modifications are conducted at User's Group meetings, or other gatherings, the Kaypro's screen size and location become very inconvenient. The ideal solution would be a large external monitor. There were commercial solutions, not now available. *MicroCornucopia* came through, however, and their solution is being examined for future re-engineering and publication.

Firmware:

This is an interesting term, used to describe software captured in permanent or semi-permanent memory (i.e. read-only memory, erasable read-only memory, or some variation thereof, includes GALs and PALs, etc. for you purists). Currently available options include the MicroConucopia Pro-8 Moni-

tor Rom, Pro-884, Pro-884 Max, and the Advent TurboRom in '83 and '84 versions.

MicroCornucopia

Pro-8 Monitor Rom

As you might guess, this rom allows use of Quad density drives, as well as blinking block cursor, user-definable screen dump character, selectable slow or fast step rate for each drive, automatically figures out what kind of drive you're using, ignores nulls on the command line, allows use of 1-4 drives of 191k, 390k, 784k in any combination. Use of 3 or 4 drives requires the 4-drive decoder board or modification.

Pro-884 Monitor Rom

All of the above, but for the '84 machines.

Pro-884 Max Monitor Rom

All of the above, for '84 machines, with ZCPR1 in rom to allow warm boots without a bootable disk mounted.

Advent TurboRom

This is the only one of these four that is "hard drive aware". It will find and boot from a hard drive if there is one installed. It will also allow cursor configuration, blinking or steady, line (varying widths) or block, user selectable screen-dump character, keyboard type ahead buffer and keyclick disable/enable, current hour and minute display on the 25th line if there is a clock installed, supports the Kaypro clock as well as others, drive deselect timing method and interval selection, automatic execution of preselected programs on boot (such as NZCOM, see software, below), selectable boot drive other than a:, and use of 1-4 drives of any capacity from 191k through 784k in any combination as well as a hard drive. NOTE use of 3 or 4 floppy drives requires the Advent Personality/Decoder board. (Construction project in the previous two issues of TCJ).

A note on standardization

All of the '84 Kaypros (except the Robie and 4x) will run CP/M 2.2f with the TurboBios when the TurboRom is installed. This is an easy way to standardize operating systems if you have multiple Kaypros.

All of these roms are compatible with both the MicroSphere and Advent hard drive installations, but only the TurboRom will boot from either installation. All of these roms allow up to 4 floppy drives of various capacities, and all will run at 4.0MHz or 5.0MHz.

Selection of firmware will depend on previous selections of hardware. If all you want is increased floppy capacity, the "biggest bang for the buck", is to use 2 or 3 quad density drives, 1 double density drive (for data interchange capability), one of the *MicroCornucopia* roms and the 4-drive decoder board or modification. On the other hand, if you have or want a hard drive installed, the Advent TurboRom is the way to go.

Software:

On the operating system front, unlike Bill Gates' minions Digital Research was firmly in control of CP/M 2.2 (courtesy of our copyright laws) and emphatically wasn't doing anything in the way of improvements. Fortunately, hackers being hackers, a fellow named Richard Conn came up with ZCPR (Z-80 Command Processor Replacement) and released the source code into the "Public Domain" which meant anyone could have it and use it for free. The first version was ZCPR, later dubbed ZCPR1, when ZCPR2 was released. ZCPR made the "user" command in CP/M obselete, moving between user areas was accomplished by issuing a command in the form "du:" return, where "d" is the drive letter, and "u" is the user number such as "b12:" return. ZCPR2 added additional embedded commands, and continued to evolve until, courtesy of Joe Wright and Jay Sage, the current version is NZCOM. The best part is that although previously the user had to edit source code and complie and overlay the CCP, it is now self-installing, and a lead-pipe cinch to customize.

NZCOM is available from Sage Systems East whose advertisement is in this issue and most others at a nominal cost and is without a doubt the most cost effective, best single upgrade available.

For those of you who are interested, the BI&W K-28 runs at 5.0MHz, sports 2 784k drives, 1 390k drive, and a 20mb hard drive, and runs NZCOM with named directories on boot. It also has provision for an external monitor for club demonstrations and an amber CRT.

Preview of Coming Attractions

As previously mentioned, next issue will resume the construction projects. In future issues, you will see both '83 and '64 external monitor connections and the *MicroCornucopia* decoder board and alternative mother-board modification. Another planned project is "The Beginners Guide to Trouble-shooting the Kaypro", which will probably be published one chapter at a time.

end..

WANTED

TCJ Needs an FTP site with
1 Gig or more space to
collect OLD BIOS source files
for possible CD-ROM.
Accessing same file space by regular
BBS is also very dersirable!
If you have the facilities and
would like to help continue
the computer restoration of
older systems, please contact:

Bill Kibler
Editor
The Computer Journal
PO Box 535
Lincoln CA 95648
B.Kibler@GEnie.geis.com

Regular Feature

68xx/68xxx Support

6809 Assembly & Flex

Small System Support

By Ronald W. Anderson

My Progress in C

A couple of times ago I mentioned that I was beginning to become more comfortable with C, and able to get my programs to work much more quickly, having avoided the really dumb errors a beginner makes in C. What it took was not a translation of a program in another language, but starting from scratch on a significant program (presently 31 pages of source listing plus some from scratch font bitmap libraries). Starting from scratch you feel free to experiment more. I've learned most of the features of C including the use of Structures and Unions, and further, I feel comfortable using them. I don't have to run to a reference book to look up the syntax of every little thing I want to do.

Assembler Part 3

This time I would like to use a few more of the instructions of the 6809. It is about time we got to look at a loop and some branching instructions. I thought perhaps we could allow the user to input a number (of course as ascii digits from the keyboard) and output a binary representation to the screen. I.e. input 100 and the output will be 0000 0000 0110 0100, Hexadecimal 0064, which is the same as decimal 100. 64 + 32 + 4.

The algorithm for this is somewhat as follows:

```
RESULT = $0000
CH = GETCHAR
WHILE CH IS A VALID ASCII 0 THROUGH 9 DO
BEGIN
        CH = CH AND $0F (BITWISE AND)
        MULTIPLY RESULT BY 10
        ADD CH TO RESULT
        CH = GETCHAR
END

PRINT CRLF
FOR N = 1 TO 16
BEGIN
        IF RESULT AND $8000 NOT ZERO PUTCHAR ASCII 1
        ELSE PUTCHAR ASCII 0
        ARITHMETIC SHIFT RESULT 1 BIT POSITION LEFT
END
PRINT CRLF
```

We won't check (at least for our first try) to see that the input number is less than 32768. We'll rely on the user to do that for now.

Here is the program source listing with comments, followed by the assembler output listing. First a comment describing the program

```
* PROGRAM TO INPUT AN ASCII NUMBER LESS THAN 32767
* AND CONVERT TO BINARY.
```

An assembler directive to put the name ASCTOBIN on each page if we use the page option of the assembler.

```
NAM     ASCTOBIN
```

Here is a new assembler directive. FDB means Form Double Byte. We could use RMB 2 here to reserve a 16 bit variable storage space, but FDB allows us to INITIALIZE the value in NUMBER, in this case to zero. It saves us three instructions that would be required to store zero at that location (CLRA, CLRB, STD NUMBER).

```
NUMBER FDB 0
TEMP    RMB 1
```

Another new Assembler directive is FCC (Form Constant Character(s)). A character string may be delimited by quotes or any other punctuation character not included in the string. I frequently use slash / or backslash \. The 4 is the string terminator used by FLEX. The assembler allows a string of bytes outside of the delimiters. Frequently one would use $0D, $0A, $04 to add a crlf to the string.

```
PROMPT  FCC "INPUT AN INTEGER < 32767 ",4

* FLEX EQUATES

GETCHR  EQU $CD15
PUTCHR  EQU $CD18
PSTRNG  EQU $CD1E
PCRLF   EQU $CD24
WARMS   EQU $CD03
```

Pstrng wants X pointing at the string in memory. It outputs a CRLF and then the string.

```
START   LDX  #PROMPT
        JSR  PSTRNG
```

Now we start a loop to get characters from the user.

```
LOOP    JSR    GETCHR
        CMPA   #'0
```

if the code is less than the ascii zero code $30, or if it is greater

than the ascii 9 code $39, it is not a valid digit and we are done inputting the number to be converted.

```
BMI    DONE  if it is less than zero we are done
CMPA  #'9
BGT    DONE  if it is greater than 9 we are done
```

If it was a valid digit we got past the branch instructions. Now we "AND off" the high order nybble and what is left is the binary code for 0 through 9.

```
ANDA  #$0F
```

Tuck it away for later.

```
STA   TEMP
```

This seems like a funny place to multiply by ten since we have just gotten the first digit and NUMBER must be 0 at this point, but it makes the loop regular, and it belongs here for successive input digits. MULTEN is our first example of a subroutine. It multiplies the value of NUMBER by ten and stores the result back in NUMBER.

```
BSR   MULTEN
```

When I first wrote this test program I got some nonsense results so I put in a couple of lines to output the value of NUMBER each time through the loop. I quickly discovered that I had forgotten the ANDA #$0F instruction above. I left the instructions in place and commented them out so you could see the technique.

```
* THE FOLLOWING TWO LINES WERE USED TO SEE HOW THE
* MULTEN SUBROUTINE HANDLED SUCCESSIVE DIGITS
* JSR PCRLF
* BSR OUTBIN
```

Now we get our digit that we tucked away before the subroutine call. This is a good place to point out the necessity of "saving registers" sometimes before a subroutine call. You will notice that MULTEN uses both the A and B accumulators for the multiply operation so anything left in those registers would be lost.

```
LDB   TEMP
```

Temp is always a small number 0 thru 9 so it fits the B accumulator. Before we add it to NUMBER, however, we have to clear A so we don't add garbage to it. NUMBER is a 16 bit number so we must do a 16 bit add and using D is the easiest way to acomplish that.

```
CLRA
ADDD  NUMBER
STD   NUMBER
```

Now we have the first use of a loop in our assembler programs so far. We branch back to the label LOOP unconditionally. We get out of this loop when a non digit is detected, at which point we jump to the instruction past BRA LOOP

```
BRA   LOOP
```

Here is the destination of the branch out of the get input loop. This starts the last phase of the program. We have input digits and in the process converted them to a binary number. Now we output the number as a string of 0's and 1's. We do this by a BSR OUTBIN subroutine. When we return from the output routine we return to FLEX.

```
DONE   BSR   OUTBIN
       JMP   WARMS
```

Here are the two subroutines we used. Previously we had used only JSR's to FLEX routines. This time we have written two of our own. We have the binary representation of our input number in the variable NUMBER. Here we analyze it one bit at a time starting at the highest order, and output an ascii 1 or 0 depending on the value of the bit. This code seems a little redundant. We have coded it in line essentially duplicating the first loop which outputs the high order byte of NUMBER to output the second byte. It would be possible at the cost of increased complexity to keep track of which byte we are outputting, and to run through the same loop twice. The point of this program is not the ultimate compactness, but understandability.

```
* OUTPUT BINARY NUMBER SUBROUTINE
```

We're going to use X for a counter to tell us when we've output 8 bits from the high order byte, so we preload it with the value 8

```
OUTBIN  LDX   #$0008
```

ACCB is an 8 bit register. When we load it with NUMBER we get only the high order byte. We would have to load D to get the whole number. We'll use ACCB for this so we can use ACCA to output an ascii 0 or 1.

```
        LDB   NUMBER HIGH ORDER BYTE ONLY
```

The BITB instruction does a mental "AND" of the imediate value with the value in B and sets the appropriate flags in the condition code register, i.e. zero and negative if applicable. I this case we simply want to test the bit for non-zero since if it is not zero it has to be 1. The first time through the loop we are testing the highest order bit of NUMBER.

```
LOOP2   BITB  #$80
```

If it is not a zero we want to output a 1 so we branch to label OUT1

```
        BNE   OUT1
```

If it was a zero we didn't branch so we LDA #'0 and output it, skipping the code to load a '1 into ACCA.

```
        LDA   #'0
        BRA   SHIFT
OUT1    LDA   #'1
```

If we branched to OUT1 and did the LDA #'1 we simply "fall through" to this point. If we loaded a '0 we branch to this point.

```
SHIFT   JSR  PUTCHR
```

Now we've output the character that represents a bit of NUMBER. We shift the binary representation of NUMBER left one place, so the second highest bit is now in the high order position.

```
        ASLB  #1
```

Decrement our loop count in X, and go around again if X hasn't reached zero. We'll talk more about the LEAX instruction later. It literally means "Load Effective Address". The argument -1,X means to load X with one less than it's present value, or to decrement it by 1. The 6800 had the instruction DEX which the 6809 assembler would accept, but it would generate the code for LEAX -1,X. This instruction is more flexible than DEX because we can LEAX -5,X or increment it with LEAX 5,X just as well.

```
        LEAX  -1,X
        BNE   LOOP2
```

After we have output 8 bits, we fall through to here and first print a space to separate the two bytes of the number.

```
        LDA  #$20 SPACE
        JSR   PUTCHR
```

The assembler allows us to do some limited arithmetic with our operands. In this case we LDB with the byte AFTER the label NUMBER, the low order byte of number.

```
LDB NUMBER+1 LOW ORDER BYTE
```

Now we do the same loop as above.

```
        LDX  #$0008
LOOP3   BITB  #$80
        BNE  OUT2
        LDA  #'0
        BRA  SHIFT1
OUT2    LDA  #'1
SHIFT1  JSR   PUTCHR
        ASLB  #1
        LEAX  -1,X
        BNE  LOOP3
        JSR  PCRLF
        RTS
```

As I said before, this is redundant code and not very efficient, but easy to understand. Later we'll talk about what I call "telescoping" the code a bit. To do so we have to have another variable to tell us whether we are going through the loop for the first or second time, since at the end of the loop we do different things depending on the pass.

Now comes the difficult one. The code is small but the operation is a bit complex. This is essentially an integer multiply routine hard coded so that one of the operands is decimal ten, binary 1010. First we move NUMBER into ACCD and shift it left twice. Recall that there is no ASLD instrution. It may look like we are shifting in the wrong order, but if you study the instruction set you will see that this works. ASLB shifts the low order byte of NUMBER left one position. On an ASL instruction the leftmost bit overflows or rotates into the carry bit of the processor. ROLA (ROtate Left A) picks up the carry bit and shifts it into it's LOWEST order bit. The ASLA instruction does NOT do this, but simply shifts a zero into the low order bit. To use ASLA rather than ROLB would be an error and the routine wouldn't work properly. Anyway, the combination ASLB ROLA shifts D one place to the left, effectively multiplying NUMBER by 2. We do it again, which multiplies NUMBER by 4. Then we ADD NUMBER to D again, resulting in 5 times number. Finally we shift the whole thing left once more, resulting in 10 times NUMBER, and store NUMBER back where we got it.

```
* MULTIPLY BY 10 SUBROUTINE

MULTEN LDD   NUMBER
        ASLB              MULT BY 2
        ROLA
        ASLB              X4
        ROLA
        ADDD  NUMBER      X5
        ASLB              X10
        ROLA
        STD   NUMBER
        RTS

        END   START
```

Notice as I said above, that MULTEN wipes out the contents of both the A and the B accumulators. It was necessary to save the contents of B before calling this subroutine, and to restore them after the return.

Neither of these subroutines is called more than once (except during my debug session). You may then well ask why we would set them apart as subroutines. Actually we do that for debug purposes. MULTEN is rather self standing and if we had a problem with it (I did) we could look at it's result back in NUMBER after each call to it. I used that technique and quickly found my dumb error. (Murphy's law number 234, "There is no such thing as a smart error".)

The output routine worked first try so it didn't need debug, but it is a nice self contained module. A better question than why these are subroutines would be to ask why the first part of the program isn't a subroutine "GETCON" for Get values and Convert them. We could even separate those two functions by getting the ASCII value string into memory and then converting it to binary. That would be less efficient but perhaps even easier to understand.

Let's take a second pass at the program:

```
* PROGRAM TO INPUT AN ASCII NUMBER LESS THAN 32767
* AND CONVERT TO BINARY.

NAM     ASCTOBIN

NUMBER FDB 0
TEMP    RMB 1
COUNT   FCB 0

PROMPT  FCC "INPUT AN INTEGER < 32767 ",4
```

```
* FLEX EQUATES

GETCHR EQU  $CD15
PUTCHR EQU  $CD18
PSTRNG EQU  $CD1E
PCRLF  EQU  $CD24
WARMS  EQU  $CD03

* THIS IS THE "MAIN PROGRAM"

START  LDX  #PROMPT
       JSR  PSTRNG
       BSR  GETCON
       BSR  OUTBIN
       JMP  WARMS

* GETCON SUBROUTINE, GETS ASCII STRING NUMBER AND
* CONVERTS TO BINARY STORED IN NUMBER

GETCON EQU *
LOOP   JSR    GETCHR    RETURNS CHAR in ACCA
       CMPA   #'0
       BMI    DONE      IF < '0 IT IS NOT A DIGIT
       CMPA   #'9
       BGT    DONE      IF > '9 IT IS NOT A DIGIT
       ANDA   #$0F      IT'S A DIGIT, MAKE IT THE BINARY
                        VALUE OF DIGIT
       STA    TEMP      SAVE IT
       BSR    MULTEN    MULTIPLY RESULT BY
                        10 (0 ON FIRST PASS)
       LDB    TEMP      ADD DIGIT TO NUMBER
       CLRA
       ADDD   NUMBER
       STD    NUMBER    STORE BACK IN NUMBER
       BRA    LOOP      GO  AGAIN UNTIL NOT A DIGIT
DONE   RTS

* OUTPUT BINARY NUMBER SUBROUTINE

OUTBIN LDX    #$0008
       TST    COUNT     PRESET TO 0
       BNE    PASS2
       LDB    NUMBER    HIGH ORDER BYTE ONLY IF PASS 1
       BRA    LOOP2
PASS2  LDB    NUMBER+1
LOOP2  BITB   #$80      TEST HIGH ORDER BIT
       BNE    OUT1      OUTPUT EITHER 1 OR 0
       LDA    #'0
       BRA    SHIFT
OUT1   LDA    #'1
SHIFT  JSR    PUTCHR
       ASLB   #1
       LEAX   -1,X
       BNE    LOOP2
       TST    COUNT
       BNE    DONOUT
       LDA    #$20      SPACE
       JSR    PUTCHR
       INC    COUNT
       BRA    OUTBIN
DONOUT JSR    PCRLF
       RTS

* MULTIPLY BY 10 SUBROUTINE

MULTEN LDD   NUMBER
       ASLB              MULT BY 2
       ROLA
       ASLB              X4
       ROLA
       ADDD  NUMBER      X5
       ASLB              X10
       ROLA
       STD   NUMBER
       RTS

       END   START
```

All I've done here is to make GETCON a subroutine as mentioned above, making the main program five lines long. It separates the functions a bit more and makes the program more modular. I decided to "telescope" OUTBIN by using the variable COUNT to determine which byte of NUMBER to load into ACCB and then to use it to detmine when we're done outputting. It works identically to the first version. Making GETCON a subroutine added a BSR and an RTS instruction, and with all the added code to handle COUNT in OUTBIN, removing the redundant code made the program three bytes shorter than the original, at 130 bytes total. If you study OUTBIN in the new version you will certainly agree that it is a lot harder to understand than the first version. The program flow is contorted.

Just in case you haven't realized it by now, the subroutine GETCON is basically doing what INDEC did in the previous part of this. That is, INDEC grabbed a number as an ASCII string on the command line of our ADD program and converted it to binary for us. This program prompts for the number to be input and GETCON does just what INDEC did in the way of conversion. I mentioned last time that programs tend to be shorter if you can make use of operating system calls. We could extend this present program or use all but OUTBIN and make it a loop to input and sum numbers, in which case it would be pretty much what we did last time but using our own supplied routines rather than FLEX calls. Part of my reason for doing this is to show you that there is nothing difficult or magic about the built-in FLEX calls. In fact most of them are subroutines that were needed by the operating system itself. The authors were kind enough to allow us access to their code and to document that access for us in the programmer's guide.

This time I won't waste space with the assembler output listing. if you are following this, you can type it in and assemble it for yourself.

What have we added to our list of useful instructions? First we have the two new assembler directives FCC (Form Constant Character), FDB (Form Double Byte) and it's smaller version used for COUNT in the revision of the program FCB (Form Constant Byte). These are useful for assembling a constant value or a string value in a program as we have used them here. FCB and FDB support a list of bytes or words (16 bit values) separated by commas, or a single value as we have used them.

We've used the X register as a counter for a loop.

We've introduced loops by means of branch instructions. We need to discuss the branch instructions a bit. If you have the folding Motorola instruction set card that SWTPc supplied with their computers you can look at it and find out which condition flags are set as a result of each instruction. A com-

parison such as CMPA #'9 sets the condition code flags. If the comparison is equal, i.e. if ACCA contained the code for ascii 9, the zero flag is set. If they are not equal, and ACCA contained a value smaller than '9, the minus flag is set. If ACCA value was greater than '9 neither the minus flag nor the zero flag is set.

The Motorola card indicates which flags are set and which don't change as a result of each instruction. LEAX -1,X for example sets the zero flag when X contains zero (or our code above wouldn't work).

The various branch instructions test the condition code flags to decide what to do. BNE branches if the zero flag is not set. BEQ branches if the zero flag IS set. BGT branches if the comparison result is greater than zero, i.e. register number was larger than that to which it was compared. BLT branches Less Than. BLE branches less than or equal. BGE branches greater than or equal. BLO and BHI branch if the value is lower or higher respectively treating the value as UNSIGNED. I have a hard time remembering which instructions treat values as signed and which as unsigned so I'll include a table here.

Signed Branch Instructions

BGT	Branch Greater Than
BGE	Branch Greater or Equal
BEQ	Branch if EQual
BLE	Branch if Less or Equal
BLT	Branch if Less Than
BNE	Branch if Not Equal

Unsigned Branch Instructions

BHI	Branch if HIgher
BHS	Branch if Higher or Same
BEQ	(works same for signed and unsigned)
BLS	Branch if Lower or Same
BLO	Branch if LOwer than
BNE	(works same for signed and unsigned)

We've noted that we can use arithmetic with operands (LDA NUMBER+1). There is another operand supported by the assembler. I used it to define the label GETCON at the same address as the label LOOP. The operand "*" means literally "this address". LABEL EQU * assigns the label to the present program counter location without incrementing it, so the label at the next line has the same value. In the case of the above, GETCON is a sensible name for the subroutine, and LOOP is more meaningful within the routine.

We've used the ASL and the ROL instructions. If you have the Motorola card, these are diagrammed there indicating the presence of the carry bit in the picture for both instructions. There is an LSL instruction (Logical Shift Left) which does exactly the same thing as the ASL (Arithmetic Shift Left). There are the analogous Right Shift instructions ASR and LSR which DON'T do the same things. LSR shifts a zero into the high order bit of the operand while ASR propagates the leftmost bit to the right. That is, it can divide a signed integer by 2 preserving the negative sign, and it is valuable for that operation.

This example is undoubtedly far from the absolute optimum program in terms of speed or size. However it is quite small, you will have to admit, occupying only one disk sector with about half of it to spare. As long as it converts numbers faster than you can type them, it is acceptably fast.

I probably ought to mention something that is obvious once you understand it and not obvious otherwise. This assembler does not support what are known as local labels. That is every label in the program must be unique or the assembler will complain "MULTIPLY DEFINED SYMBOL". I used "LOOP" so the second time I had to use "LOOP2" (or LOOPA or whatever to make it unique). This assembler also limits labels to six characters so they necessarily become cryptic at times.

If there is anything particularly puzzling about any of this program I'd be happy to reply to questions directly or in a later column, or more likely, both. I'll answer your questions and collect all that might be of general interest and answer them all in a later column.

Flash

I've just found a good reference that outlines the use of a standard PC printer port for other purposes. It turns out that there is an 8 bit parallel port that is used to output to the printer, but it can also be used as an input port. Then there is a 4 bit output port (5th bit used to enable/disable the interrupt for the port), and a 4 bit input port.

The book is "Interfacing to the IBM Personal Computer" by Lewis C Eggbrecht, published by Howard W. Sams & Company, (C) 1990. The printer port portion starts on page 229. It is followed by a couple pages on interfacing with the Game Port. Instructions are explicit enough so you can make use of these ports easily.

At work I've successfully connected an LCD display controller to the printer port with enough hardware left over to scan an 8 by 4 membrane switch array. This replaces a $200 "industrial" parallel port board with a $12 "multi-I/O card" that has two serial ports and a game port as well.

This has just been done, so I am not quite used to the idea of such an inexpensive I/O. I will be coming up with some projects using this port. One idea I have is "Let's Build a Programmable Logic Controller". The Allen Bradley version is called a PLC. If you are not familiar with them, these devices are used in machine control systems to replace a lot of mechanical relays. Inputs read switch positions, logic levels from other devices such as solid state proximity switches, etc. They perform logic based on the states of the inputs and decide which output signals (relays) to turn on or off on the basis of the logic. For example, you want to run a motor using a "momentary" start button and stop button. The controller sees

I finish Ron's work on the last column of my Computer Corner, since I want you to see my comments on a PLC projects.

TCJ Center Fold

Special Feature

All Users

Hayes 80-103A Modem

For this issue of *The Computer Journal*, I felt it appropriate to feature the Hayes S-100 Modem card. Since David Jaffe our Computer Hero of the year started by turning this card into a remote system through BYE, it seems only fitting that we should show you what he had to start with.

As a reader you must remember that the idea that a normal person could own a modem to talk with their computer to other computers was a bit of a new thing. We are talking 1977, just when S-100 was almost the only show in town. I need also point out that legally you were suppose to RENT a DAA or Data Access Arrangement which was the phone companies hardware interface to their equipment. It seems that MA-Bell was worried you might blow up their master equipment with your computer system.

Reading the manual for this card is quite a trip down memory lane. As a broadcast technician of the day, I often interfaced to these old DAA's when doing remote broadcasts. Now days everyone has FCC approved transformer coupled systems that are a part of the modem.

The S-100 card can be interfaced as I/O or memory. The I/O is preferred as the memory used up 1024 bytes of addressing. The interface is rather software simple with only four registers to deal with. The main device is the TI 6011 UART (or TR1602, AY5-1013, S1883) and a Motorola MC14412P modem chip. 300 Baud is the maximum speed.

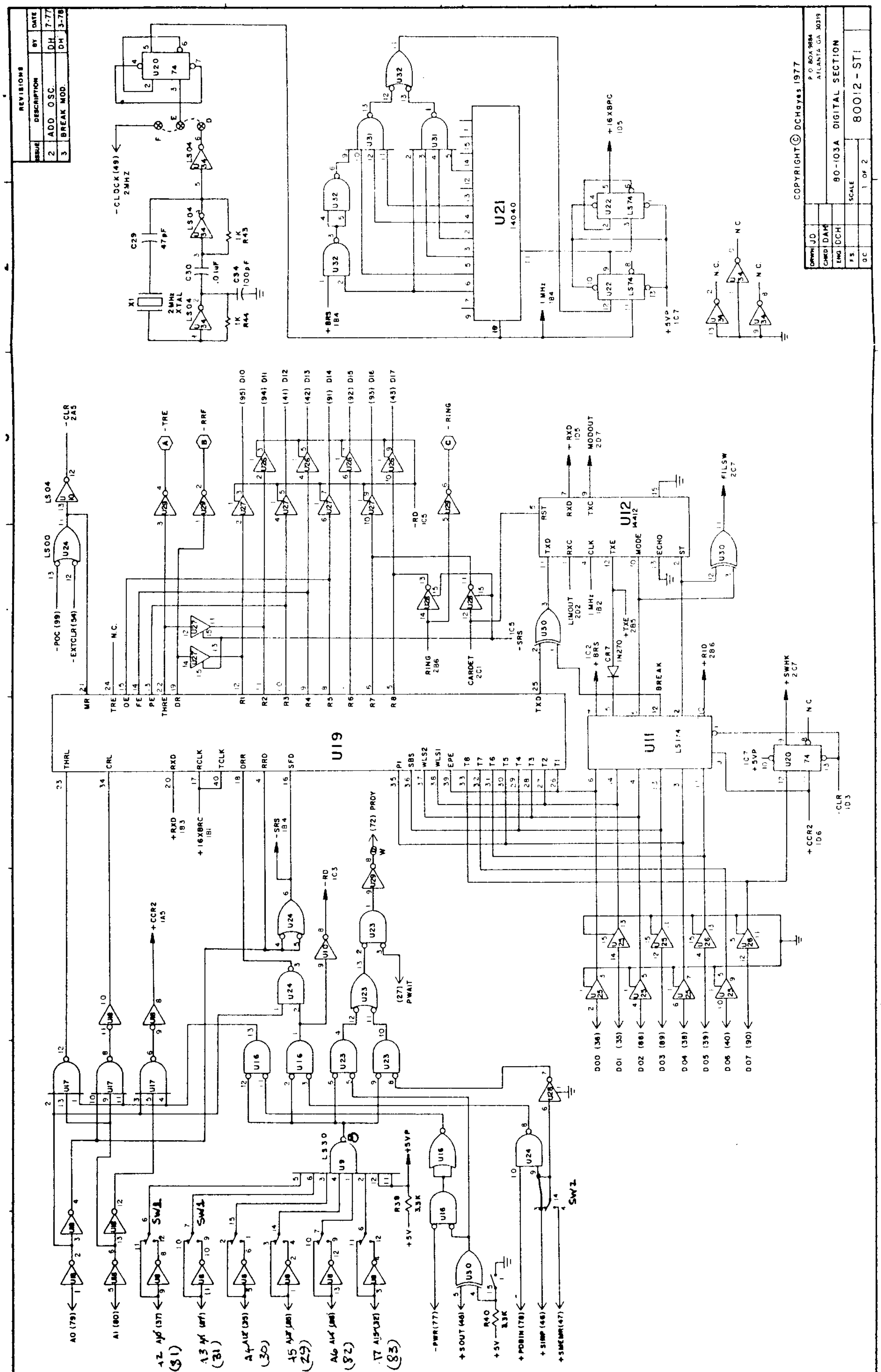
The major work horse in this unit is the analog separation system. The system is composed of filters (resistors and capacitors) and analog amps. Since what you get on the phone line is an audio signal composed of a mark and space tone separated by 200Hz, the analog devices must be able to see a difference between these two tones. The seeing of one tone says it is a mark or one, the other tone a space or zero. For duplex operation (sending and receiving at the same time) two sets of frequencies are used (1070 space and 1270 mark for originating modem, with 2025 space and 2225HZ mark for the answering modem.) These conversions are still going on today, but using similar filters all on one chip, and in cases of the very high baud rates, digital filtering using a DSP system (Digital Signal Processing).

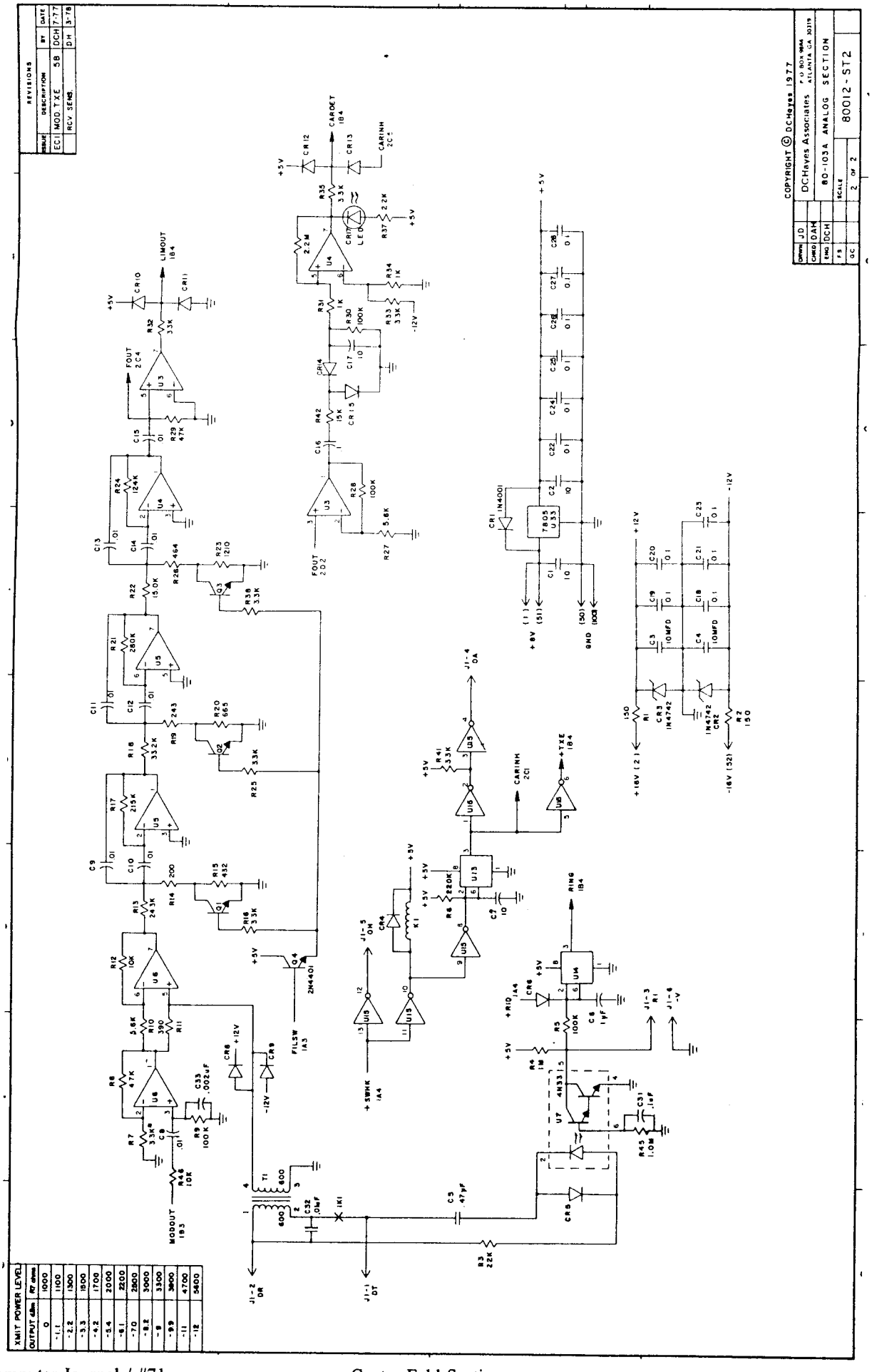
You can see and say that it all started here, both for modeming in general and the Hayes Modems in specific. Since the use of modems was new, Hayes provided sample programing examples to help you get started. The example in assembly was a complete terminal program in 1024 bytes for use on your Altair system (also provided in HEX format for directly entering the program without an assembler). The assembly language program is in 8080 mnemonics, but any processor could have been used, since nothing on the card is CPU dependent.

The basic use of the card, involves setting the control ports for the Baud rate, number of characters, number of stop bits. Once done you load data into the transmit buffer, monitor the control or status register till the character has been sent, then start again by inputting the next character. To receive the character we monitor the status port or register till a flag says a character is ready to be removed. The read from the data port clears the flag and we again wait till the next character is received. This information was provided in the programmable register chart of the manual. These steps are the same today, whether doing 300 Baud with the old Hayes modem, or 19200 Baud on the latest modems available.

Real efficient programmers learned about interrupt handling when they tried to use the optional interrupt control signals of the board. In the above, our program simply loops and checks for either empty transmit or full receive registers. Not finding either causes it to skip over the needed steps, only to recheck the register on the next pass. Interrupt handling says, we perform some other operation in the foreground (such as processing your last command) while in the background nothing happens till an actual character is received. Then the processor stops what it is doing, gets the character, sticks it in a buffer, flags that the buffer has a new character, and returns the processor to the previous task as if nothing had happened. Interrupts can get tricky and few early systems used them, but this product would support them if you wanted to try.

Many of the early companies provided incentives for users to start using their product. Dave Jaffe got a special treat from Hayes when he went to visit them. They sent an engineer out to the airport to meet him. This was due to Dave's work on a group buy of cards for the Chicago users group. The object was getting your hardware to be the main product supported by software, in this case Christensen's Modem program and Dave's BYE. As we know now, these people did for Modems what Popular Electronics did for the Altair and S-100 systems, created a whole new industry.


U19
U21
14040
U12
U11
LS174
COPYRIGHT © DCHayes 1977
80-103A
DIGITAL SECTION
80012-ST1

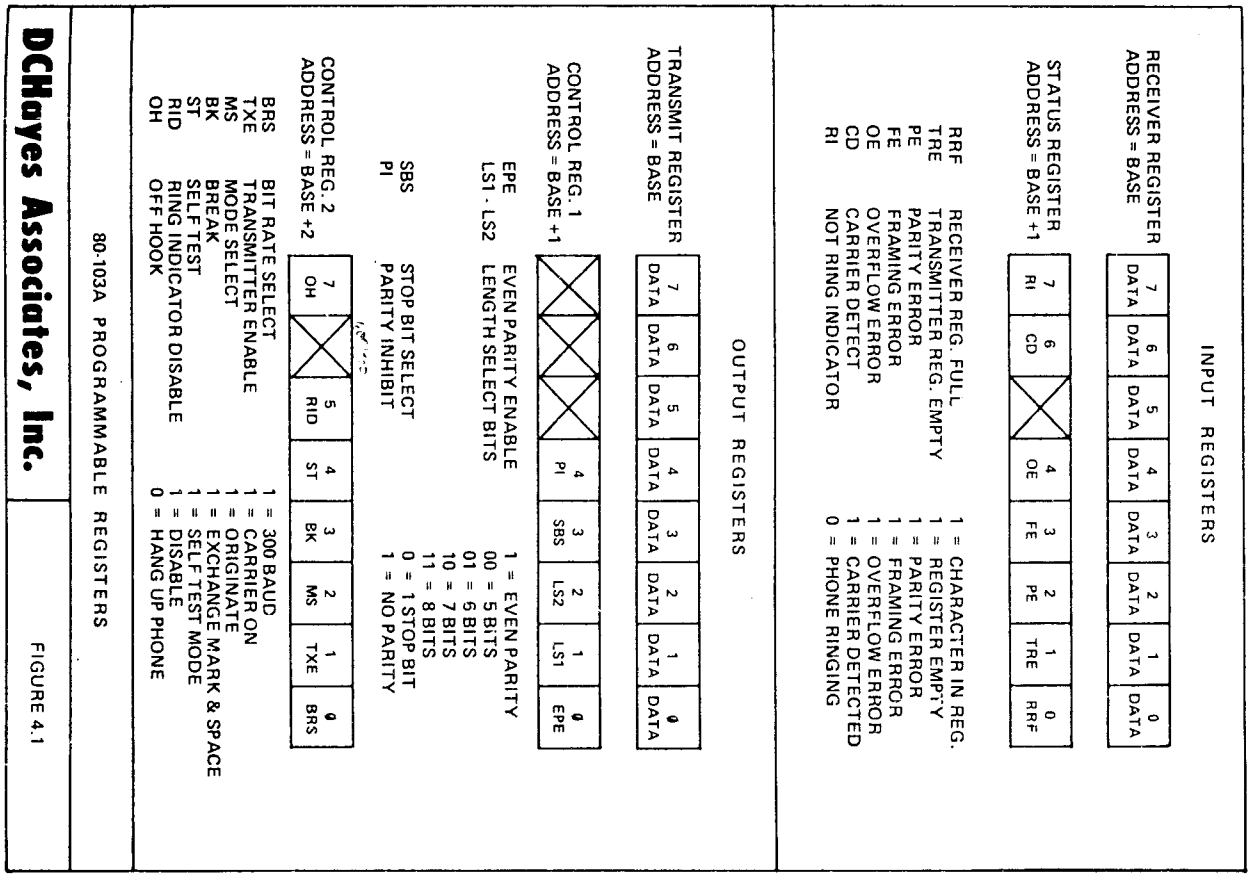

REVISIONS
ISSUE
DESCRIPTION
BY
DATE
EC1 MOD TXE 58 DCH 7-77
RCV SENS DH 3-78
COPYRIGHT © DCHayes 1977
DCHayes Associates P.O. BOX 1844 ATLANTA GA 30319
80-103A ANALOG SECTION
80012-ST2
2 OF 2
XMIT POWER LEVEL
OUTPUT dBm
Rf ohms
0 1000
-1.1 1100
-2.2 1300
-3.3 1500
-4.2 1700
-5.4 2000
-6.1 2200
-7.0 2800
-8.2 3000
-9 3300
-9.9 3600
-11 4700
-12 5400
+5V
+12V
-12V
LIMOUT 184
FOUT 2C4
FOUT 2D2
CARDET 184
CARINH 2C1
RING 184
+TXE 184
MODOUT 183
FILSW 1A3
+SWHK 1A4
J1-1 DT
J1-2 DR
J1-3 R1
J1-4 -V
J1-4 0A
J1-5 0M
7805 U33
+8V (1) (51)
GND (50) (100)
+16V (2)
-16V (52)
2N4401
IN4001
IN4742
4N33
LED

PORT 00	SWI F	SW2 C	PORT 80	SWI F	SW2 8
04	B	C	84	B	8
08	7	C	88	7	8
0C	3	C	8C	3	8
10	E	C	90	E	8
14	A	C	94	A	8
18	6	C	98	6	8
1C	2	C	9C	2	8
20	D	C	A0	D	8
24	9	C	A4	9	8
28	5	C	A8	5	8
2C	1	C	AC	1	8
30	C	C	B0	C	8
34	8	C	B4	8	8
38	4	C	B8	4	8
3C	0	C	BC	0	8
40	F	4	C0	F	0
44	B	4	C4	B	0
48	7	4	C8	7	0
4C	3	4	CC	3	0
50	E	4	D0	E	0
54	A	4	D4	A	0
58	6	4	D8	6	0
5C	2	4	DC	2	0
60	D	4	E0	D	0
64	9	4	E4	9	0
68	5	4	E8	5	0
6C	1	4	EC	1	0
70	C	4	F0	C	0
74	8	4	F4	8	0
78	4	4	F8	4	0
7C	0	4	FC	0	0

DCHayes TABLE 3.5 I/O MAPPED LOCATIONS

ADR 0000	SWI F	SW2 F	ADR 8000	SWI F	SW2 B
0400	B	F	8400	B	B
0800	7	F	8800	7	B
0C00	3	F	8C00	3	B
1000	E	F	9000	E	B
1400	A	F	9400	A	B
1800	6	F	9800	6	B
1C00	2	F	9C00	2	B
2000	D	F	A000	D	B
2400	9	F	A400	9	B
2800	5	F	A800	5	B
2C00	1	F	AC00	1	B
3000	C	F	B000	C	B
3400	8	F	B400	8	B
3800	4	F	B800	4	B
3C00	0	F	BC00	0	B
4000	F	7	C000	F	3
4400	B	7	C400	B	3
4800	7	7	C800	7	3
4C00	3	7	CC00	3	3
5000	E	7	D000	E	3
5400	A	7	D400	A	3
5800	6	7	D800	6	3
5C00	2	7	DC00	2	3
6000	D	7	E000	D	3
6400	9	7	E400	9	3
6800	5	7	E800	5	3
6C00	1	7	EC00	1	3
7000	C	7	F000	C	3
7400	8	7	F400	8	3
7800	4	7	F800	4	3
7C00	0	7	FC00	0	3

DCHayes TABLE 3.6 MEMORY MAPPED LOCATIONS



DCHayes Associates, Inc. 80-103A PROGRAMMABLE REGISTERS FIGURE 4.1

CONNECTING IDE DRIVES

by Tilmann Reh

Special Feature

Intermediate Users

Part 5: GENERIC IDE

Generic Z80 IDE Interface

(plus a few words about the LittleNet interface)

Discussions about connecting IDE devices like hard disks or eventually CD-ROM have grown during the last months. Seemingly we triggered something with the articles about the interface basics and my interface board for the ECB-Bus. Another actual development was the single-chip IDE interface by Claude Palm (of Palmtech in Australia) who also offered an S-100 board draft containing his chip.

This in turn started a discussion between *TCJ* columnist "Dr. S-100" (Herb Johnson), Johnathan Taylor from England (with whom I discussed Wayne Sung's interface version for the QX-10 before), and me. Our main theme was how to reduce the total cost of an IDE interface so it will be of more general interest than the relatively expensive S-100 board.

I finally made a circuit and PCB draft for an interface which will directly connect to a Z80 processor. This has two great advantages: First, the board is very small, thus PCB prices will be much lower than for an S-100 board. Second, the direct connection to the Z80 processor socket opens the way to ALL Z80 computers, not only those based on the S-100 bus (or ECB-Bus, like my interface described earlier in *TCJ*). So we might get a larger volume, further reducing PCB costs. Of course, it also has a disadvantageous side: the mechanical construction is left to the end user.

However, it has to be made clear that this is only a draft yet. We are now trying to determine if there is enough interest in this generic interface to build a prototype and produce a run after it really works. But before we go further on this, here are some technical details:

The new interface (I call it GIDE, for Generic IDE) consists of a GAL chip (Generic Array Logic, a smaller programmable logic device) and a few TTLs. Its size is about 60 x 70 mm (2.4 x 2.8 in), including all connectors. It plugs directly into the Z80 socket via two pin strips, or onto a cable which plugs into the Z80 socket. The processor itself will normally be plugged in the appropriate socket on the interface board. The interface is I/O mapped, with user selectable base address (in increments of 10h). The IDE drive will be connected via flat cable. This cable and of course the hard disk drive will not be included with the interface.

Quoting Herb (from a message he placed in the nets): "It would be encouraging to all involved with this design to know who would buy it, what they would require for support, and how much they would pay for it. Clearly, this is not a very "commercial" venture: there are too few Z80 systems around, and few Z80 or CP/M vendors for "reselling" for a commercial effort.

But, to avoid losing money and to get a reasonable quantity produced, we need to set a fair and acceptable price. Too cheap a price will make it unreasonable to make; too large, unreasonable to buy. Please use your honest judgment and suggest what YOU would spend, and what you would expect."

Meanwhile, Herb and Johnathan started lists of interested people who responded to their messages in the nets. We also discussed how to handle further support like board testing (for those who would like a kit) and driver software (example routines, test programs etc.). But before we spend more time and money on this topic, we really should know if there is enough interest to justify all those expenses. So all you who are interested, please contact one of us (see addresses below) with the above requested information. We really would be glad to make this board! If there is interest, I could also offer to describe this new interface in another article here in *TCJ*.

For those who are already active in getting used and/or cheap hard disks, here is an advice: I strongly recommend Conner drives, since those drives have the lowest power consumption and also generate the least noise. Some other makers like Quantum and Sony are comparable in noise emissions. Seagate drives are horrible in both terms, and additionally have slightly different timing specifications which sometimes may lead to problems. So if you have the choice, try to get Conner drives.

Actual state of the LittleNet development:

Since my circuit draft for the isolated RS-485 interface converter was printed in *TCJ* #69 (in the column of Rick Rodman), we further discussed some technical details of this interface. Soon I will finish a PCB draft which probably will be single-sided, so the PCB will be easy to make in home cellars. The parts costs will also be very low. We will surely inform you about further results and also print the PCB artwork in *TCJ*. If any of you have some suggestions or questions, please contact Rick or me as soon as possible, so we can consider your thoughts in PCB layout and/or software details.

Contacts:

Tilmann Reh, Am Rueckelchen 5a, 57078 Siegen, Germany
InterNet: tilmann.reh@hrz.uni-siegen.d400.de
Fax (at work): +49 271 484520
Herbert R. Johnson, CN5256 #105, Princeton, NJ 08543, USA
InterNet: hjohnson@pluto.njcc.com
Voice/FAX +1 609 771 1503 (8am-11pm EDT)
Johnathan Taylor, UK
Internet: jet@centron.com, Fidonet: 2:2501/307.9
Rick Rodman, USA
InterNet: rickr@aib.com

Regular Feature

Intermediate

Compupro 8/16

Dr. S-100

By Herb R. Johnson

"Dr. S-100's Winter column" by Herb Johnson (c) Dec 1994
Internet: hjohnson@pluto.njcc.com

Introduction

Hope you all had a good holiday, and the best for 1995. It's "back to basics" for me after noodling around with IDE stuff. My next columns will be pure S-100...or IEEE-696 if you insist. But just to finish off the subject....

Networking for IDE

In recent issues of *TCJ* I discussed an IDE interface for the S-100 bus, as designed by Claude Palm of Palmtech, and I included his design announcement. Unfortunately, the costs of the Palmtech PAL (programmed logic device) and the S-100 card brought the projected priced well above $100. I received letters and messages of interest, but not enough to justify the costs of producing the printed circuit cards. Those still interested in the Palmtech product should contact Mr. Palm as he can provide the chip and other information.

Following that effort, most of my time in December was in Internet correspondence with Tilmann Reh, who has written a series of articles on his Z280 card and its IDE interface. As I received only a small response to the Palmtech S-100 IDE interface, I hoped to encourage Tilmann to modify his interface design to accept Z80 control and data signals. As it turns out, others were also encouraging him in this direction. His report in this issue will announce a daughter card which is designed to plug into the Z80 chip socket to provide an IDE interface. I encourage all my S-100 readers to read and consider his announcement. Even those readers with 8085, 8088, or other S-100 processors should be encouraged to respond, as Tilmann's design can be adapted to those processors.

Once again, a show of interest and a willingness to pay a good price for such a product will encourage its small production. As the Tilmann GIDE is smaller and cheaper it should stimulate a good response. Contact myself or Tilmann without delay.

Compupro 8/16

I recently acquired a Compupro S-100 system. Well, actually ANOTHER Compupro...but this one has some unique hardware and software, and it represents one of the classic S-100 and Compupro systems of the early 1980's. The Compupro 8/16 was in part a reaction to the IBM PC and in fact generally superior to it. Although the PC's - thanks to the clone makers - overwhelmed the market, it was a real race for a few years. After I've written a few articles about other people's computers and IDE interfaces, I wanted to get "back to basics" in my column, and this "new" ten-year-old system will be the basis of my next series.

A few months ago I got a call from **Jim Briggs** of Mt Laurel, NJ. He had read some of my work here and there on S-100 systems, and asked if I would be interested in yet another system. It was the oft-repeated story: he was a software developer in the CP/M days, then had moved on to MS-DOS, but never gave up his old system (to support customers, or just because it worked so well!). After sitting idle for some years, he needed the space...NOW!. I usually show some level of interest at this point, depending mostly on the sheer WEIGHT and volume of stuff, and what it will cost to ship. So I asked what he had...and he caught my interest.

Jim described his system as "a Compupro 8/16, with two 64K memory cards and one 128K memory card". So far, this is a modest system with a dual processor card, namely the 8085 and the 8088, either of which can run. (Zenith fans will recognize this as the same configuration in the Z-100 (or Z-121) system which, in fact, is a "clone" of the Compupro card.) The memory cards as described would provide plenty of memory for running CP/M 80 or CP/M 86. And the 8088 processor runs at 6 MHz, faster than the original PC at 4.77, so there!

Drives and I/O

Jim went on to note two 8-inch double-sided drives and a Disk 1 controller. I was a little disappointed by the controller: the Disk 1A is more recent and more desirable, but the Disk 1 is certainly a good controller for 8-inch drives. Double-sided double density 8-inch disks, like the smaller 5-1/4 inch and 3-1/2 disks on IBM-PC's, will hold over a megabyte of programs and data, and transfer it at comparable datarates. In the CP/M environment, that will hold a lot more programs than in the MS-DOS world! Nonetheless, this was pretty typical of most Compupro systems. In fact, I don't remember any 5-1/4 inch drives in use on any Compupro 8/16 system I've heard of.

The I/O on Jim's system was also typical: The System Support 1 card is often used as the console interface. It includes parallel ports, timers, the serial port,

and a socket for a arithmetic chip, the AM9511. Unlike the 8087 coprocessor for the 8088, this is more of a "slave processor chip" that you feed data and computations to, and receive results from, under control of a program. It never was a cheap chip, usually above $100, and I didn't think to ask if he used it. There was also an Interfacer 4, which is Compupro's 4-port serial card using 8051 chips. These are software programmable UART's, as opposed to the hardware strap-selectable features of the older 8050 UART.

Graphics?

So far, this Compupro system was kinda typical. But Jim had more to say. He proceeded to tell me about the TWO graphics cards installed on this system and THAT peaked my interest! He said both cards supported color, and one was particularly good with graphics (although not to today's VGA level).

Now, for those of us who remember the early days of the introduction of the IBM PC, you will remember the original PC display was either monochrome with graphics characters; or color with what amounted to a color TV. IBM called the former MGA, and the latter CGA. All this was terribly expensive, a few thousands of dollars. As IBM had yet to dominate the MS-DOS or CP/M market, and thanks to the BIOS interface in both operating systems that separated hardware from software, programs had to run in very "plain" fashion to support ANY output device, or had to be configurable to any device by the user. These were the days of "percent compatibility". It was acceptable for a time to be "60%" or "%80 compatible" and in fact to run quite a lot of software.

What about Jim's cards? One was called **Microsprite**, and it supported a standard color video (TV) monitor which he would include. The other, from **Illuminated Technologies**, used a special digital RGB monitor, which he ALSO had available. AND...he had an old Canon color printer as well! "But it probably needs a new ink cartridge", he said.

"Do I want it?"

Even with all the above, I hesitated. I describe that I couldn't offer much for the system as these things come and go often enough. Jim knew they were not worth much in general, but more to the point it was "either give it away or throw it away" NOW! I asked when I could schedule to pick it up, and I was surprised to hear he could DROP IT OFF at my door step! He was THAT eager to pass it along, (presumably before my wife found out and canceled the whole deal). How could I refuse. I DID have the presence of mind to refuse the NEC Spinwriter which, while a great full-character impact printer and workhorse, is very heavy and less convenient to use than an HP Laserjet. Likewise, I refused the Televideo terminal: I do have some discretion!

In a few hours, as Jim lives only a half-hour drive from me, he was at my doorstep. All the equipment was in original boxes, clean as new. We chatted about his work of the era and he went back home, his former system stacked in my basement. A large paper bag of data books, a box of manuals, another box of 8-inch diskettes, and other piles of equipment including a small Epson printer he forgot to mention. With the cold of winter still a few months away, I delayed playing with it: it wasn't going anywhere.

Well....I DID set up the color printer. It's a Canon 1080A printer, with a single cartridge for green, yellow, cyan, and black. A few days of calls found Canon as a source for cartridges for about $20 each, and $12/roll for its special paper for best color transfer.

Next issue

I'll talk about how I brought up the system and made the usual minor repairs that any system needs after sitting in a box for five years or so. And, you'll hear about the little surprises I got when I looked at the cards. Graphics will probably be an article in itself, but not immediately. Correspondence will resume in the next issue!

Calls for support

I have docs for the Compupro stuff. Anyone out there with paper, cartridges, or software for the Canon 1080A? Or software I can test the 9511 math processor on (yes, it had one!).

WANTED

TCJ needs your embedded story or project!

Our readers are waiting to hear from you about how you developed that embedded project using 8051 or 6805. Used Forth, "C", BASIC, or assembler any language is fine, just tell us what happened, how you did it,and how it ended up. No project too small!

Send those Ideas to:

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

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

Send your items to:

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

Regular Feature

Editorial Comment

Building 8048 Systems

8048 Emulator

By J. G. Owens

THE STORY OF AN 8048 ETC. EMULATOR

The name "MON48" I borrow from the amazing MON80, I believe it was called, authored by the inscrutable Jeff Moscowitz, to whom I was frequently unforgivably rude in my ill-spent early middle years. (Moscowitz is only slightly better-known these days for Pascal Z, the star software product of Ithaca Intersystems wherein we toiled in those ancient days.) MON80 was a little 8080 machine-language monitor that, if I recall correctly, had the then-amazing property of being able to run at any location! Since then I've been naming various feverish hacks "MON" this and "MON" that in honor of whatwas probably the first piece of interesting software I encountered in my life.

INTRODUCTION

This is the story of the various components of an 8048 emulator I built about 8 years ago, and used now and then since I wrote it. The components of this system are:

1. A 25-or-so-IC 8048 emulation card.
2. The source for the software that runs in that card.
3. The source for an IBM-style PC program that "hosts" the emulator card through an RS-232 link. (Turbo C 2.0)
4. Free Free Free MA48 8048 macro assembler and linker and source.

Also required to do useful work is an 8048 EPROM burner. I don't know who sells those these days or how much they cost. Mine is 10 years old and is no longer available.

All four elements will be covered in this document (the assembler's mostly in another ZIP file), and so much more. It is a long and convoluted story; it began long ago and far away (oh well at least a small distance) on a two-diskette Kaypro IV computer. Designing/building the emulator was fun, exciting, and extremely complicated; I had to buy a better scope just to debug it (my NRI 6 mHz job just didn't do any good, so I graudated to a Hitachi 40! mHz job). The emulator still runs today!

HOW DID THE 8048 EMULATOR START?

I now remember why I built this in the first place. I was convinced my employer needed a low-rent proprietary bar-code reader, and I proceeded to build such a thing — i.e. to read simple bar codes that could be printed cheaply. I developed and tested the system with a record-player (obsolete technology itself now!) which I'd use to move the test bar codes before my sensor. I believe somewhere in the middle of this I became convinced that I should have an emulator, and somehow that led to the 8048 emulator. I know the bar code project existed; whether I ever actually got to the point where I debugged it with the resulting emulator — who can say?....

DO YOU REALLY NEED THIS?

The short answer is NO.

I use this emulator to build things when I want to; as recently as November 1993 I built a simple-minded little MIDI (Musical Instrument Digital Interface) gadget: it emitted a do-nothing MIDI code every 2 seconds or so, so a Casio keyboard I'm fond-of wouldn't keep putting itself to sleep.

But if I didn't have this support equipment already-built, I'd probably find something else, i.e. like the Basic Stamp I've seen advertised. (But note that, for instance, emitting MIDI is something you almost *have* to do in assembly language, Basic Stamp or not, and/or use an external UART, which of course is what we're usually trying to avoid.)

SO WHY ARE YOU READING THIS?

There are two plausible reasons for distributing this material I can think of:

1. Existing 8048 "orphan" projects may get resurrected. This isn't as obscure as it might appear; the 8048 is *everywhere*, or at least relatives and descendants are. Kaypro keyboards had them I think, and of course the IBM keyboard started that way....

2. You're curious about emulation in general. Heaven knows, there's precious little data on the topic, and this may be useful in that regard. But see "The Death of Emulation" below.

ONE REALLY NICE THING ABOUT THE 8048

P10..P17 and P20..P27 are what Intel calls "quasi-bidirectional" ports; later they became ashamed of the practice and moved on to better things, but the way they work is you'd output a 1 to a bit (they come-up at reset that way), and then you could input from that port! Momentarily while switching to high, a relatively low impedance of around 5K ohm is switched in, but normally a high output is held high through 50K ohms — which is still enough to pull-up most TTL inputs, which essentially float high

unless they're pulled low. But also easy to drive low; you see, this was a "cheap" way to get bidirectionality.

But additionally, as inputs they had the handy feature of being pulled-up. This works wonderfully for things like keyboard scanning; normally you have to add resistors to pull-up the inputs in such a circuit, but the 8048 thoughtfully does it for you! One is normally not in a hurry with keyboard scanning anyway, and I've found the built-in pull-ups perfectly useful, for instance, for my AGO (American Guild of Organists) standard 32-note pedal board MIDI gadget. In hand-wired projects, which of course is what I'm doing, those resistors can be very annoying.

There. Is that a feature, or is that a feature?

THE DEATH OF EMULATION

As I explained in one of my incredibly excellent letters to Electrical Engineering Times, all emulation technology depends on the fundamental difference between current microprocessor technology and current *digital* technology: the idea is to "fake out" a microprocessor chip by running rings around it with much faster TTL chips or whatever.

The 8048 emulator described here definitely does that; it uses average TTL (or HCT or etc.) technology to trap various events the 8048 target does in its normal processing. *If* the TTL/HCT technology wasn't *fast enough* to catch those events, *it wouldn't work*.

So you see children, today the microprocessors are becoming the fastest most-current technology; they aren't step-children anymore. So the old trap-em and catch-em techniques won't work. To compensate, some modern processors are developed with built-in debugging features which can be accessed with appropriately-knowledgeable software. And I approve — but the problem that I suspect will assert itself anyway is that manufacturers will refuse to build-in utterly-most-recent-hi-tech features into the debug suite, because normally they're trying to hide them from others. Indeed, in the old days, manufacturers were typically hostile to emulator manufacturers for the same reason.

THE 8048 SINGLE-CHIP MICRO-COMPUTER

The 8048 is a very common — i.e., readily available, cheap, and therefore obsolete — single-chip microcomputer originally released by Intel but now available from many sources. It was/is? typically used as a micro-controller to provide intelligence for keyboards (IBM PC keyboards are based on this architecture), printers, microwave ovens, etc. The 8748 — a 2K, EPROM version of the part — goes for as low as $8 retail.

8048 VERSIONS

	EPROM	ROM	RAM(bytes)
8048:		1K	64
8049:		2k	128
8748:	1K		64
8749:	2K		128
8035:	no eprom/rom		64
8039:	no eprom/rom		128

Of course these were versions available around 1981....

Developing programs for the 8048 family, even with MA48 or some other assembler, isn't much fun without development equipment like the 8048 emulator described here. Also, an 8748/49 EPROM burner is necessary — unless you plan to have someone manufacture the things in quantity and burn the ROMs on 8048/49 parts; I don't know how you go about doing that.

But it's probably feasible to write simple programs for the 8748 without debugging equipment like the 8048 emulator described herein, providing intelligence for small quantity applications with a start-up cost equal to an appropriate EPROM burner. (Honestly, I haven't done it; I used the 8048 emulator described here on a few projects. I'm not sure I'd like to try an 8048 project without an emulator.)

The ROMless 8035 and 8039 parts are intended to be used with external ROM. This probably makes no sense; the external memory bus uses-up many of the parts pins, and the 8048 family's already so obsolete and quirky that the only excusable reason for using it is to get the microcomputer-on-a-chip feature and all the pins. I believe you can actually use the emulator to debug such designs, simply by not using the I/O pins that are normally devoted to the bus; but I'm not sure you'd want to.

DOCUMENTATION

Documentation for the 8048 should start with the Intel book, which of course is almost certainly unavailable. Computer and Ham shows are the logical place to look for such material.

FEATURES

Here's some 8748 features as hyped on page 6-8, "MCS-48 Family of Single Chip Microcomputers User's Manual", Intel, 1981:

1. 8-bit CPU, ROM, RAM, I/O in single package
2. Single 5V Supply
3. 2.5 usec and 5.0 usec cycle versions. All instructions 1 or 2 cycles.
4. Over 90 instructions: 70% Single Byte
5. 1K x 8 ROM/EPROM [8749 2k]
   64 x 8 RAM [8749 128]
   27 I/O Lines
6. Inteval Timer/Event Counter
7. Easily Expandable Memory and I/O
8. Compatible with 8080/8085 Series Peripherals
9. Single Level Interrupt

I don't even know what #9 refers to; I'm pretty sure there's more than one interrupt source. #7 may still be true; they're probably referring to the 8243 expansion gadget, two of which are actually used in the emulator.

SOME NOTES ON 8048 SOFTWARE, INSTRUCTIONS

I'm not going to try and explain the 8048 here, but suffice it to say it's an extremely simple-minded 8-bit processor; one "virtue" of working with the part is to see just how awful things can be, i.e. when you find yourself getting annoyed at segment registers or something. The processor, for instance, has

no subtract instruction!

And then there are the "page-relative" conditional jmp instructions. You can "jc destination" to somewhere within the current 256-byte page, i.e. any address which differs from the current one only in the last 2 hex digits; too bad you don't find out if that's valid or not until *after* you assemble and link! You can't imagine how annoying it is....

There's another stupid jump mode that allows execution outside the basic 2K space built into the instruction set, the SEL MB1 / MB1 instructions, but with any luck you'll never encounter that one. However, the vast and amazing emulator software itself uses the instruction, but apparently only in emulation, which makes sense I suppose (i.e., to *emulate* it).

There is a little set of low-memory vectors:

```
;VECTORS:
        org     0
        jmp     begin
        org     3
        jmp     dointerrupt
        org     7
        jmp     dotimer
```

Those I believe are all the vectors; as indicated, the external interrupt pin vectors to location 3, and the timer to location 7.

I own a lovely 8 x 10 plastic card which is your basic cheat-sheet for the 8048 etc. instruction set: "8048 & RELATIVES" Copyright 1981, Micro Logic Corp., POB 174, Hackensack, NJ 07602, and for all I know they're still around somewhere. You'd still need an Intel book or something like it to design a hardware system, but the cheat sheet *may* be adequate for writing or modifying software.

THE EMULATION CARD

FILES: DB8039.A "IBM" - ASCII schematic.

The document DB8039.A is the schematic of the 8048 emulator card. This document was originally created in *WordStar* on a *Kaypro IV*. It was a heavily-hacked WordStar into which I had insinuated a set of graphics characters.

DB8039.A is a translation of that file, using the more-or-less standard IBM graphics characters. If your printer can print characters like this "E" cross character, then it may be able to print the schematic. (If you can't *see* the cross character, then probably the rest of this file looks a little strange, and you need a different text editor/viewer: I use the excellent shareware QEDIT program; the EDIT that came with DOS 5.0 works too.)

Please forgive me for the OR and AND gates in the ASCII-old schematic; but fixing them would be difficult, and I believe the rendition contains adequate information.

DB8039.A is the basic source document for the hardware; I will comment on it here, but note that it's filled with its own comments, and even timing diagrams, parts lists including where I bought the parts, and who knows what else. I've mostly resisted the temptation to edit the DB8039.A material much beyond converting it to the pseudo-ASCII form, because it constitutes a record of work-in-progress while the emulator was built, and I figure whatever's in there may be useful. (Note that the translation process apparently is less-than-perfect, an an occasional odd character may be left-in, although I'm getting rid of them as I notice.)

I can't swear that my existing emulator precisely corresponds to this schematic; however it *definitely* is the only working document that *ever* existed for this emulator, and it is what I wired it from.

OVERVIEW AND HISTORY OF DESIGN

Originally the design was to be a "simulator": U1, an 8039, would contain the MONITOR program to control the system, and U22, another 8039, would simulate a proposed TARGET design. I just figured the designs would be extended-bus designs, because I didn't see any other way to download the test code — that is, without an extended bus connected to RAM, how would I get the test code into the chip?

Remember those terms: the MONITOR is the controlling 8039; it always runs the same program, the monitor program. The TARGET is the test 8039, which executes out of the RAM in U19 and U21.

So I built all that, apparently, including break-point, single-step, and a mechanism for forcing the target 8039 to do what I told it now and then ("FORCE INSTRUCTIONS INTO TARGET" in the schematic), and then apparently it occurred to me it was easy enough to convert the result into an emulator, so I added U30, the EMULATOR socket, into which is plugged a big cable, the other end of which gets plugged into a real target card I'd've concocted. I really don't remember how I did it, but perhaps we'll find-out as we go along....

(The big cable is a part probably available from Digikey; you gotta make sure you get a 1-to-1, some reverse the pins and things... I actually have the *parts list* for this thing still, and it says <6" 40-PIN R8136-6-ND 4.65> which was the Digikey part number and price in 1985. I didn't include this stuff because I assume most of it's obsolete.)

PARTS PLACEMENT

I mounted most analog components on headers, which is feasible in a digital circuit; that is the meaning of things like "H1" nearby U1 and U11; it's just documenting where X1, C1, C3, C2, and R4 are mounted. And I just walked over and looked at the emulator, and they *are* mounted there.

Next issue we will continue on with "Limitations of Emulation", schematics, and later still software and more. BDK.

DIVMOD and More

by Walter Rothenkolber

Special Feature

Intermediate Users

Two Shorts from Walter

Pitfalls in Signed Integer Divide & Modulo
by Walter J. Rottenkolber

If you are like me, most of the integer divide and modulo in my programs use positive integers. As a result I hadn't paid much attention to what happens when you use signed integers. While working on developing some Forth mixed integer operators, I learned that Forth-83 uses floored instead of the symmetric integer division common to other languages. (Other Forths, eg. figForth, use symmetric integer division).

Floored division rounds toward negative infinity. As a result, zero applies only to a positive quotient. The sign of the quotient follows a rule similar to that of multiplication, ie. positive if the signs of the dividend and divisor are the same, and negative if not the same. The sign of the remainder follows that of the divisor. The modulus cycles through the same values as the dividend passes from a positive to negative value.

In symmetric division, the quotient is rounded toward zero. This leads to a discontinuity in which there are both positive and negative zero quotients. The remainder inverts its cycling as the dividend passes from a positive to negative value. The sign of the quotient follows the same rule as in floored division, but the remainder tracks the sign of the dividend.

To compare the two integer division methods, I wrote test programs in Forth-83 (Laxen & Perry, v2.1.0), Basic-80 (Microsoft, v5.21), Turbo Pascal (Borland, v3.01a), and C/80 (Software Toolworks, v3.10). These cycle a simple integer divide and modulo routine through the zero point. There are two lists per language, one with a positive divisor and the other with a negative divisor.

The calculation is:
dividend /MOD divisor = quotient modulo
and the check for accuracy is:
dividend = (quotient * divisor) + modulo.

Comparing the lists for Forth-83 and Basic will give you a good idea of the differences between floored and symmetric division. It's obvious that symmetric division has extra zeros as the dividend slides from positive to negative. Any device using the quotient for control would have a 'bobble' at that point. Also, the modulus inverts so that any device using it would, in effect, work backwards after passing the zero point.

Since Forth-83 is widely used in embedded systems, a smooth and consistent transition through the zero point was considered valuable in programming such things as plotters and robot arms.

Knuth makes a distinction between floored and ceilinged conversion of real numbers to integers. A floored number is rounded to the greatest integer less than or equal to the number. A ceilinged number is the least integer greater than or equal to the number. Forth division is consistently floored. Symmetric division is floored above zero, and ceilinged below zero.

Wirth distinguishes between Eulerian (or symmetric) arithmetic and modulus (or congruent) arithmetic. In symmetric arithmetic, a given dividend and divisor will result in the same quotient and remainder (ignoring the sign).

Ada, the official language of the Department of Defence (DOD), is designed for both general and embedded system programming. It has both a remainder (REM) and modulo (MOD) function. REM returns the symmetric value, while MOD returns a floored value.

In a recent development, the newly approved ANS-Forth standard has both floored and symmetric integer division primitives, though it favors floored division.

Both Forth and Ada recognize the difference between the arithmetic remainder and the mathematical modulus operator.

For positive divisors, you can convert a Remainder (symmetric MOD) to a (floored) Modulus as follows:

```
(a)  k := m REM n ;  if k < 0 then k := k + n ;
(b)  k := (((m REM n) + n) REM n) ;
```

Or the Modulus to a Remainder:

```
(a)  k := m MOD n ;  if m < 0 then
if k <> 0 then k := k - n ;
```

I left the discussion of the Turbo Pascal and C/80 data until last because each has a serious bug in the Modulus function. Compare the two with that of Basic. The bug shows up when a negative dividend or divisor (or both) are used. I've marked

the erroneous MOD operations with an asterix (*). As you can see, the absolute values of the remainder are okay but the sign may be inaccurate. Since most Modulo functions are with positive integers, you might not have stumbled onto this problem. The integer Divide function appears intact.

You can rely on the MOD function of Turbo Pascal only when using positive integers. With C/80, all non-zero MOD's with a negative divisor are erroneous. I find this disappointing because both compilers are mature products that I like.

In C/80, the error is in the c.div function of CLIBRARY.ASM. However, any attempt at a fix that increased the size of the library resulted in code that compiled, but crashed when run. (Why?).

When translating programs from other languages to Forth-83 (or vis versa), you should be aware that not all signed integer divisions are the same. Moreover, even mature programs can have subtle bugs. To avoid surprises, tests of computed data should be compared to expected values. A lesson most programmers soon learn is to distrust and verify.

References:

Robert Berkey:"Positive-Divisor Floored Division", Forth Dimension, Vol. XII, Num. 1, May/June 1990, p. 14.

Donald E. Knuth:"The Art of Computer Programming", Vol. 1, 2nd Ed., Addison-Wesley, 1973, p. 37.

Robert L. Smith:"Signed Integer Division", Dr. Dobb's Journal, Num. 83, Sept. 1983, p. 86-88.

Niklaus Wirth:"Algorithms & Data Structures", Prentice-Hall, 1986, p.25.

=== THE END ===

Integer Divide and Modulo

Forth-83 Laxen & Perry

```
6 /MOD  3 =  2  0      6 /MOD -3 = -2  0
5 /MOD  3 =  1  2      5 /MOD -3 = -2 -1
4 /MOD  3 =  1  1      4 /MOD -3 = -2 -2
3 /MOD  3 =  1  0      3 /MOD -3 = -1  0
2 /MOD  3 =  0  2      2 /MOD -3 = -1 -1
1 /MOD  3 =  0  1      1 /MOD -3 = -1 -2
0 /MOD  3 =  0  0      0 /MOD -3 =  0  0
-1 /MOD  3 = -1  2     -1 /MOD -3 =  0 -1
-2 /MOD  3 = -1  1     -2 /MOD -3 =  0 -2
-3 /MOD  3 = -1  0     -3 /MOD -3 =  1  0
-4 /MOD  3 = -2  2     -4 /MOD -3 =  1 -1
-5 /MOD  3 = -2  1     -5 /MOD -3 =  1 -2
-6 /MOD  3 = -2  0     -6 /MOD -3 =  2  0
```

MBasic Microsoft

```
6 /MOD  3 =  2  0      6 /MOD -3 = -2  0
5 /MOD  3 =  1  2      5 /MOD -3 = -1  2
4 /MOD  3 =  1  1      4 /MOD -3 = -1  1
3 /MOD  3 =  1  0      3 /MOD -3 = -1  0
2 /MOD  3 =  0  2      2 /MOD -3 =  0  2
1 /MOD  3 =  0  1      1 /MOD -3 =  0  1
0 /MOD  3 =  0  0      0 /MOD -3 =  0  0
-1 /MOD  3 =  0 -1     -1 /MOD -3 =  0 -1
-2 /MOD  3 =  0 -2     -2 /MOD -3 =  0 -2
-3 /MOD  3 = -1  0     -3 /MOD -3 =  1  0
-4 /MOD  3 = -1 -1     -4 /MOD -3 =  1 -1
-5 /MOD  3 = -1 -2     -5 /MOD -3 =  1 -2
-6 /MOD  3 = -2  0     -6 /MOD -3 =  2  0
```

Turbo Pascal Borland

```
6 /MOD  3 =  2  0      6 /MOD -3 = -2  0
5 /MOD  3 =  1  2      5 /MOD -3 = -1 -2 *
4 /MOD  3 =  1  1      4 /MOD -3 = -1 -1 *
3 /MOD  3 =  1  0      3 /MOD -3 = -1  0
2 /MOD  3 =  0  2      2 /MOD -3 =  0  2
1 /MOD  3 =  0  1      1 /MOD -3 =  0  1
0 /MOD  3 =  0  0      0 /MOD -3 =  0  0
-1 /MOD  3 =  0  1 *   -1 /MOD -3 =  0  1 *
-2 /MOD  3 =  0  2 *   -2 /MOD -3 =  0  2 *
-3 /MOD  3 = -1  0     -3 /MOD -3 =  1  0
-4 /MOD  3 = -1 -1     -4 /MOD -3 =  1  1 *
-5 /MOD  3 = -1 -2     -5 /MOD -3 =  1  2 *
-6 /MOD  3 = -2  0     -6 /MOD -3 =  2  0
```

C/80 Software Toolworks

```
6 /MOD  3 =  2  0      6 /MOD -3 = -2  0
5 /MOD  3 =  1  2      5 /MOD -3 = -1 -2 *
4 /MOD  3 =  1  1      4 /MOD -3 = -1 -1 *
3 /MOD  3 =  1  0      3 /MOD -3 = -1  0
2 /MOD  3 =  0  2      2 /MOD -3 =  0 -2 *
1 /MOD  3 =  0  1      1 /MOD -3 =  0 -1 *
0 /MOD  3 =  0  0      0 /MOD -3 =  0  0
-1 /MOD  3 =  0 -1     -1 /MOD -3 =  0  1 *
-2 /MOD  3 =  0 -2     -2 /MOD -3 =  0  2 *
-3 /MOD  3 = -1  0     -3 /MOD -3 =  1  0
-4 /MOD  3 = -1 -1     -4 /MOD -3 =  1  1 *
-5 /MOD  3 = -1 -2     -5 /MOD -3 =  1  2 *
-6 /MOD  3 = -2  0     -6 /MOD -3 =  2  0
```

== END ==

Source Code for Test List
of Integer Divide and Modulo

Forth-83 Laxen & Perry

```
\ Forth Integer Divide and Modulo  WJR21JUN94
: .FMOD  ( dividend divisor)
2DUP SWAP 3 .R ." /MOD"
3 .R ." = " /MOD  3 .R  3 .R ;
: FMOD2
-6 6 DO  CR  I 3 .FMOD ."    "
I -3 .FMOD  -1 +LOOP ;
```

Basic Microsoft

```
10  REM  Test of Integer Divide and Modulo
20  B=3 : D=-3
30  PRINT
40  FOR A = 6 TO -6 STEP -1
50  PRINT A" /MOD "B" = "A\B A MOD B;
60  PRINT "    ";
70  PRINT A" /MOD "D" = "A\D A MOD D
80  NEXT A
90  END
```

Turbo Pascal Borland

```
PROGRAM  Tmod2 ;
VAR
a, b, c : Integer ;
BEGIN
b := 3 ;
c := -3 ;
FOR a := 6 DOWNTO -6  DO
BEGIN
Write(a,' /MOD ',b,' = ', a div b,'  ', a mod b) ;
Write('    ');
Writeln(a,' /MOD ',c,' = ', a div c,'  ', a mod c) ;
END
END.
```

C/80 Software Toolworks

```
/* test /mod list  */
#include "printf.c"
main()
{
static int a, b = 3, d = -3 ;
for (a = 6; a >= -6; —a )
{
printf("%d /mod %d = %d %d", a, b, a / b, a % b);
printf("    ");
printf("%d /mod %d = %d %d\n", a, d, a / d, a % d);
}
==== END ===
```

Accessing the Kaypro Keyboard
by Walter J. Rottenkolber

There are times when the standard keys are not the ones you need. Two common methods of remapping the keyboard are to use the configuration option in CP/M, or to use a high memory key capture program.

There is a third. Tap into the keyboard directly. If you roll your own programs, portability is not an issue, and you eliminate the need for multiple copies of CP/M, or the hassle of reloading other programs when you reboot.

I use the direct key method in my Forth screen editor as it allows me to remap the keyboard, including the cursor and keypad keys, to whatever edit functions I desire. When out of the editor and in the Forth interpreter, the standard CP/M functions apply.

The Kaypro keyboard connects to the computer via a serial line. It uses channel B of the same serial chip that outputs the modem port (channel A). Routines similar to those used to access the modem port will work for the keyboard. Since key entry is slow, a simple polling method is more than adequate.

Small systems commonly use two methods to access hardware. The first is to connect the hardware directly to the address lines and treat the port as a memory read/write. The 6502 and 6809 CPU's can only use this method. In the days of 16KByte DRAM, computers based on these CPU's used only 48 KBytes for RAM. The upper 16 KBytes of address space were reserved for accessing ROM code and hardware ports. Only later, when 64 KByte DRAM chips became popular, did bank switching allow 16 KBytes of RAM to be available in this location.

The 8080 and Z80 chips can be used this way too, and the Kaypro does just that with the lower 16 KBytes of memory to deal with ROM and Video routines. But, these CPU's also have an internal bankswitch that allows you to separate hardware ports from main RAM. The IN and OUT assembly instructions select this alternate pathway. Only the lower eight address lines are used for port access, so only 256 addresses are available, more than enough for most small systems. To avoid confusion, port addresses are often called port numbers.

High level languages adapted to 8080 and Z80 computers have keywords for the I/O Ports.

Forth83 uses PC@ (p# - b) to fetch data from a port, and PC! (b p#) to output data, where p#= 8-bit port#, and b= data byte. The comments in the parenthesis are the stack picture, which shows the before and after state of the data stack. To fetch data, for example, the Port# is first placed on the data stack, and after PC@ runs, it leaves the data-byte on the stack. Forth83 also uses P@ and P! to deal with the 16-bit data access allowed in the IBMpc's. (Adding to the confusion, fig-Forth uses P@ and P! to access 8-bit data).

Microsoft's MBasic does port access with: b= INP (p#), and OUT p#,b. In both, 'b' is a variable to which the data is assigned.

Borland's TurboPascal uses assignment to access ports: b := Port[p#] for input, and Port[p#] := b for output.

The Kaypro keyboard has two port addresses: 05 for Data, and 07 for Control/Status. Though Control/Status has several reg-

isters, only #0 (the default) is required. Receive Ready (data buffer has data) is indicated by bit0 of the Status byte being set(1). This bit can be isolated from the other bits by ANDing the byte with 01. Tranmit Ready (data buffer empty) is indicated by bit2 set(1). This is isolated by ANDing the Status byte with 04.

Screen 11 shows Forth Words to directly access the keyboard. The Words in the angle brackets are the primitives. For DKEY to be useful, it must wait for data to appear in the input buffer. So <DKEY?> is placed in a BEGIN UNTIL loop which exits only when <DKEY?> returns TRUE, indicating that new data exists for <DKEY> to fetch.

The Kaypro II floppy disk drive motor off delay is determined by a timing loop in the BIOS routine for CP/M Console status, KEY?. The direct key status, DKEY?, bypasses this and so requires DRV-OFF to turn off drive after DDLY time. Code for the delay routine is in screen 10. To reset the delay count when a drive is turned on requires tapping into the Read/Write routines in Forth.

A simpler way is shown in screen 12. VKEY is a hybrid that combines the CP/M Console Status, KEY?, with the direct data fetch, <DKEY>.

Direct key input will enable you to reconfigure the cursor and keypad keys with a simple CASE statement. All of these keys have the high bit set (80 hex or higher). A list of the key codes is in Table 1. A branch statement will separate them from the standard ASCII codes.

Some early Kaypro keyboards have extra keyswitches covered by the case top, but they will need keycaps replaced and other work to use. I find that just recycling the standard keys is good enough.

As an aside, the drive off routine can be handy if you write a program that does floppy disk access, but doesn't check console status often. The drives would just stay on. The Kaypro uses a parallel port I/O chip as a system bit controller, at port# 1C. Bit#7 turns the drives OFF when it is Set(1). The mask for bit#7 is 40h (64d). The bits in the data byte determine the status and control for a number of functions. When changing a control bit, it is important not to tamper with the other bits. That is why in DRIVE-OFF, the byte is first read in, an OR with the mask sets only bit#7, and the modified byte then written back to the port. Checking drive status, DRV-ON?, is similar in method to <DKEY?>.

Screen 13 shows what to do if all you need is the indication (flag) of a keypress. The DUP is needed in Forth to provide one flag for the IF branch, and another for output. You need to fetch and drop the keyed data. If you don't, the data from a keypress will remain in the input buffer of the ZSIO only to emerge later in the program as corrupt data.

Although most attention gets focused on getting data from the keyboard, you can also send data TO the keyboard. The Kaypro has its speaker in the keyboard, and it is activated by sending a code-byte to the keyboard. This byte selects Bells of two different durations (long and short), and the presence of the key- click. The short bell sounds about half the time of the long. The pitch is preset and cannot be changed.

Screen 14 has the Words to control the Bell. <KEMIT?> and KEMIT? check status for ready to send. It does not tell you the status of the Bell, ie. whether it's sounding. <KEMIT> and KEMIT send the Bell-code to the keyboard. The choices are limited and are listed in the screen. Don't use other values, as they will behave erratically and may even return meaningless data.

In a program you might consider separating the codes into a Beep- code (0, 2, 4), and a Click-code (0=on and 8=off), then adding the two to output as the Bell-code. The bell-code must be sent each time the bell is to be sounded. The Click status remains set until changed by a new bell-code.

```
\ Screen 10
\ Disk Drive Motor OFF — Kaypro II            WJR06JUL94
VARIABLE DDLY      10000 DDLY ! \ @ 7 secs., 1333= 1 sec.
VARIABLE DTIME    DDLY @ DTIME !
HEX
: DRIVE-OFF  1C PC@ 40 OR 1C PC! ; \ Set bit 7 in controlport
: DRV-ON?    1C PC@ 40 AND 0= ;   \ Drv-on if bit 7 = 0.
DECIMAL
: SETDOFFDLY   DDLY @  DTIME ! ; \ Place in blk R/W routine
: DRV-OFF   DRV-ON? IF -1 DTIME +! DTIME @ 0= IF
DRIVE-OFF  DDLY @ DTIME ! THEN THEN ;

\ Screen 11
: <DKEY?>  ( - f)  7 PC@ 1 AND 0<> ; \ Status Kaypro II key-port
: <DKEY>   ( - b)  5 PC@ ; \ Read Kaypro II key-port direct
: DKEY?    ( - f)  <DKEY?> ;
: DKEY     ( - b)  BEGIN  DRV-OFF <DKEY?> UNTIL <DKEY> ;

\ Screen 12
: VKEY?  ( - f)  KEY? ;  \ Use CP/M drive off.
: VKEY   ( - b)  BEGIN KEY? UNTIL <DKEY> ;

\ Screen 13
: KEYPRESS?  ( - f)  VKEY? DUP IF VKEY DROP THEN ;

\ Screen 14
: <KEMIT?>  ( - f)      7 PC@ 4 AND 0<> ;
: <KEMIT>   ( bell-code)  5 PC! ;
: KEMIT?    ( - f)      <KEMIT?> ;
: KEMIT     ( bell-code)  BEGIN <KEMIT?> UNTIL <KEMIT> ;
\S
Code     Bell   Click
0        No     ON
2        Short  ON
4        Long   ON
8        No     OFF
10       Short  OFF
12       Long   OFF


Function Keys (Kaypro II Keypad/Cursor Keys)

Keypad
Code  Key    Code  Key
B1 = 0       B2 = .
C0 = 1       C1 = 2
C2 = 3       C3 = Enter
D0 = 4       D1 = 5
D2 = 6       D3 = ,
E1 = 7       E2 = 8
E3 = 9       E4 = -

Cursor Keys
F1 = Up cursor    F2 = Down cursor
F3 = Left cursor  F4 = Right cursor

Table 1.

=== END ===
```

Moving Forth

by Brad Rodriguez

Special Feature

Intermediate Users

Part 7: 8051 Kernel

MOVING FORTH
Part 7: CamelForth for the 8051
by Brad Rodriguez

Under the prodding of Our Esteemed Editor, I present CamelForth for the 8051. CamelForth for the 6809 will follow soon! This 8051 Forth occupies about 6K of program memory. Alas, the full source listing would take 16 pages of *TCJ*, so this article includes only the significantly changed portions of the kernel. These should illustrate how the high-level code is modified for the 8051 assembler, and for subroutine threading. The full source code is available in the Forth Roundtable on GEnie as file CAMEL51.ZIP, and the freeware 8051 assembler as file A51.ZIP. But first...

Z80 ERRATA

In the file CAMEL80H.AZM, the definition of DO is given as

```
['] xdo ,BRANCH  . . .
```

It should be

```
['] xdo ,XT  . . .
```

This is of no consequence on the Z80 (where ,BRANCH and ,XT are identical), but it became embarassingly obvious on the 8051.

Also, in the words S" and (S"), the word ALIGN should really be ALIGNED. On the Z80 — and the 8051 — both are no-ops, so this mistake didn't make itself evident.

8051 CAMELFORTH MODEL

In issue #60 I summarized the design decisions for an 8051 Forth. To recap: the 8051's retarded memory addressing practically demands the use of subroutine threading. This means the hardware stack (in the 8051 register file) is the Return Stack. The Parameter Stack (a.k.a. Data Stack) is in 256 bytes of external RAM, using R0 as the stack pointer. Since that article, I've discovered that it's better to keep the Top Of Stack item (TOS) in DPTR than in R3:R2. Thus:

reg adrs	8051 name	Forth usage
0	R0	low byte of PSP (Parameter Stack Pointer)
	1-5	R1-R5 scratch registers for Forth
6-7	R6-R7	loop index
8		high byte of PSP and UP (also output on P2)
9-7Fh		119 bytes of return stack (more on 8052s!)
81h	SP	low byte of RSP (Return Stack Pointer)
82-83h	DPTR	Top-Of-Stack item
E0,F0h	A,B	scratch registers for Forth

This incorporates an idea from Charles Curley [CUR93]. On a register-rich machine like the 8051, we can keep the innermost loop index in registers. This makes LOOP and +LOOP much faster. DO must still push two values on the Return Stack: the *old* loop index, and the *new* loop limit! UNLOOP must of course restore the loop index from the Return Stack — kudos to the ANSI team for making UNLOOP a distinct word! Note that R6:R7 are *not* the topmost Return Stack item, merely the innermost loop index.

Port 2 (P2) contains the high byte of the Parameter Stack Pointer (allowing R0 to address external memory), which is also the high byte of the User Pointer — the low byte of UP is assumed to be 00. I learned the hard way that P2 can't be read while executing from external ROM, so I keep a copy of the P2 byte in register 8.

I have a novel implementation of BRANCH and ?BRANCH. Since the 8051 model is subroutine-threaded, high-level Forth is compiled as true machine code. So BRANCH can be implemented with an SJMP (or AJMP or LJMP) instruction. ?BRANCH can be implemented with a JZ instruction, *if* the zero/nonzero status of the top-of-stack is put in the accumulator (A register). The subroutine ZEROSENSE does this. So, BRANCH and ?BRANCH become

```
BRANCH:     SJMP dest
?BRANCH:    LCALL ZEROSENSE  JZ dest
```

Similar routines LOOPSENSE and PLUSLOOPSENSE allow a JZ instruction to be used for LOOP and +LOOP. For these, a call to UNLOOP must appear after the JZ, to clean up the Return Stack when the program "falls out" of the loop.

In the assembly language source file I have manually replaced the sequence

```
LCALL word  RET
```

with the shorter and faster

```
LJMP word
```

in many places [CUR93]. This works as long as "word" isn't a return-stack operator (such as R> or >R). LCALL and LJMP have also been replaced with ACALL and AJMP where pos-

sible. The CamelForth compiler does *not* attempt these optimizations.

I wrote the 8051 kernel to use "Intel" byte order (low byte first). Then I discovered that the address compiled into an LJMP or LCALL is stored *high* byte first. Rather than rewrite the entire kernel, I included a byte-swap in those words which compile LCALLs: COMPILE, !CF and ,CF (all in the Dependency word set).

Listing 1 gives the 8051 assembly language "primitives", and Listing 2 gives the Dependency word set.

HARVARD ARCHITECTURES

The 8051 uses a "Harvard" architecture: program and data are kept in separate memories. In embedded systems, these are typically ROM and RAM, respectively. ANS Forth is the first Forth standard to address the restrictions of a Harvard architecture. Briefly, ANS Forth says that a) application programs can only access Data memory, and b) all of the operators used to access memory and build data structures must operate in Data space. (Ref. section 3.3.3 of the ANS document [ANS94].) This includes the Forth words

@ ! C@ C! DP HERE ALLOT , C,
COUNT TYPE WORD (S") S" CMOVE
Yet the Forth compiler still needs to access Program space (also called Code or Instruction space). And Forth needs to maintain a dictionary pointer for Program space as well as Data space. So I've added these new words (shown in Listing 3):

I@ I! IC@ IC! IDP IHERE IALLOT I, IC,
ICOUNT ITYPE IWORD (IS") IS" D->I I->D

The "I" prefix stands for "Instruction" (since "P" and "C" have other meanings in Forth). ICOUNT and ITYPE are needed to display strings which have been compiled into ROM. IWORD copies the string left by WORD from Data space to Code space — needed to build Forth word headers and ROMmed strings. D->I and I->D are equivalents of CMOVE, which copy to and from Code space.

VARIABLEs must have addresses in Data space. So they can't use the traditional practice of putting the data immediately after the Code field. Instead, the *Data space address* of the data is stored after the Code field. In essence, a VARIABLE is a CONSTANT whose value is the Data space address. (Note that the traditional CONSTANT is still valid.)

CREATEd words, and words built with CREATE...DOES>, must work the same way. Here's how they look in Program space:

```
CODE word:    ...header... 8051 machine code
high-level:   ...header... 8051 machine code
CONSTANT:     ...header... LCALL-DOCON  value
VARIABLE:     ...header... LCALL-DOCON  Data-adrs
CREATEd:      ...header... LCALL-DOCON  Data-adrs
```

Note that CONSTANT must replace the value stored by CREATE, and : must "un-allot" both this value and the LCALL DOCON.

S" presents special problems. Strings defined with S" ("text literals") must reside in Data space, where they can be used by such words as TYPE and EVALUATE. But we expect those strings to be part of a definition, and to exist in ROM in a ROM forth environment. We could store the string in Program space, and copy it to HERE when referenced, but the ANS document does not allow text literals to exist in this "transient" storage region (ref. sections 3.3.3.4 and 3.3.3.6 [ANS93]). Also, if WORD returns its string at HERE — as in CamelForth — text literals must not alter this transient region.

My solution is to have S" *store* the string in Code space, but permanently reserve space for it in Data space, and copy it from Code to Data when referenced. ANS Forth does not yet fully address the problems of Harvard processors; something like C's "initialized data" region may eventually be required.

Since ." strings can never be accessed by the programmer, they *can* be stored in Code space, using the words (IS") and IS". (These are the "old" (S") and S".) This adds two words to the kernel, but saves quite a bit of Data space. I plan to move the string-literal words into either the Dependency word set, or a new "Harvard" word set.

WRITING TO PROGRAM SPACE

The 8051 can't actually write to Program memory. There's no hardware signal for this, and no machine instruction. Under these circumstances, the CamelForth *interpreter* will work, but new words can't be compiled. You can get around this by causing some memory to appear in *both* Program and Data space. Figure 1 shows the modification to my board, an MCB8031 from Blue Ridge Micros (2505 Plymouth Road, Johnson City, TN, 37601, USA, telephone 615-335-6696, fax 615-929-3164). U1A and U1B create a new read strobe which is active for *either* a Program or Data fetch. EPROM is selected only when A15 is low (lower 32K), and RAM when A15 is high (upper 32K). You still can't write to EPROM, of course, but you *can* execute programs out of RAM! One disadvantage: this makes @ and I@ equivalent, so it's not immediately obvious if the wrong one was used somewhere.

NEXT ISSUE...

These modifications to the CamelForth high-level code are intended to be portable to *either* Harvard or non-Harvard ("von Neumann") machines. For the latter, the new Program-space words are simply equated to their Data-space equivalents, e.g. on the Z80,

```
IFETCH EQU FETCH
```

```
ISTORE EQU STORE
ITYPE EQU TYPE
  etc.
```

In the next installment I shall modify the *8051* source code to work on the 6809...thus approaching a truly portable model by successive approximation.

REFERENCES

[ANS93] dpANS-6 draft proposed American National Standard for Information Systems - Programming Languages - Forth, June 30, 1993. "It is distributed solely for the purpose of review and comment and should not be used as a design document. It is inappropriate to claim compatibility with this draft standard." Nevertheless, for the last 16 months it's all we've had to go by.

[CUR93] Curley, Charles, Optimization Considerations, *Forth Dimensions* XIV:5 (Jan/Feb 1993), pp. 6-12.


```
            .command -ai    ; output Intel hex format
;
;================================================
; CamelForth for the Intel 8051
; (c) 1994 Bradford J. Rodriguez
; Permission is granted to freely copy, modify,
; and distribute this program for personal or
; educational use.  Commercial inquiries should
; be directed to the author at 221 King St. E.,
; #32, Hamilton, Ontario L8N 1B5 Canada
;
; CAMEL51.ASM: Code Primitives
;           Source code is for the A51 assembler.
;           Forth words are documented as follows:
;x          NAME    stack — stack       description
;           where x=C for ANS Forth Core words, X for
;           ANS Extensions, Z for internal or private
words.
;
; Subroutine-Threaded Forth model for Intel 8051
; 16 bit cell, 8 bit char, 8 bit (byte) adrs unit
; split Code & Data spaces
;           8051 PC = Forth IP  Interpreter Pointer
;           SP =      RSP Return Stack Pointer low
;           RSP high byte = 0
;           R0 =      PSP Parameter Stack Ptr low
;           PSP high = UP
;           reg 08 = P2 =  UP  User area Pointer high
;           (and PSP high), UP low = 0
;           DPTR =    TOS (top Param. Stack item)
; A,B,R1-R5 =         temporaries
;           (no W register is defined)
;           R6,R7 =   loop index
; reg 09-7F =         return stack
;
;================================================
; REVISION HISTORY
; v1.0  alpha test version, 12 Dec 94
;
;================================================
; Forth linkage
            .set link,0
            .equ IMMED,1    ; flag for Immediate word

; 8051 EQUATES
            .equ dr2,h'02   ; r2-r5 accessed as
            .equ dr3,h'03   ; direct registers;
            .equ dr4,h'04   ; required for PUSH and
            .equ dr5,h'05   ; POP instructions.
            .equ dr6,h'06   ; Assumes register bank 0
            .equ dr7,h'07   ; is selected.
            .equ UP,h'08

; FORTH MEMORY MAP EQUATES
; Memory map:
;           regs 8-7Fh  Return stack, 120 bytes, grows up
;           0000h       Forth kernel
;           8000h       Forth dictionary (user RAM)
;           UAREA=FE00h User area, 128 bytes
;           UAREA+80h Parameter stack,
;                       128B, grows down
;           UAREA+100h  HOLD area,
;                       40 bytes, grows down
;           UAREA+128h  PAD buffer, 88 bytes
;           UAREA+180h  Terminal Input Buffer,
;                       128 bytes
; See also the definitions of U0, S0, and R0
; in the "system variables & constants" area.
; A task w/o terminal input requires 200h bytes.
; Double all except TIB and PAD for 32-bit CPUs.

; Initial RAM & ROM pointers for CamelForth.
            .equ romdict,h'0e000  ; where new code goes
            .equ ramdict,h'0f000  ; where data goes
            .equ UPHI,h'FE        ; Uarea at FE00 hex

; RESET AND INTERRUPT VECTORS
;==================
            .org 0
            ljmp reset
            ljmp ie0
ie0:        reti
            .rs 4
            ljmp tf0
tf0:        reti
            .rs 4
            ljmp ie1
ie1:        reti
            .rs 4
            ljmp tf1
tf1:        reti
            .rs 4
            ljmp riti
riti:       reti
            .rs 4
            ljmp tf2
tf2:        reti

reset:      mov ie,#0       ; disable all irpts
            mov pcon,#0     ; T1 baudrate not doubled
            mov tmod,#h'20  ; T1 mode 2, T0 mode 0
            mov th1,#h'fd   ; 9600 baud @ 11.0592
MHz
            setb tcon.6     ; enable timer 1
            mov scon,#h'52  ; UART mode 1 (8-bit)

            mov UP,#UPHI
            mov p2,UP       ; user area at FE00,
            mov r0,#h'ff    ; param stack at FEFF
            mov sp,#h'8     ; ret stack at bottom
            ljmp COLD       ; enter Forth interpreter

; SERIAL I/O
;===================================
;C EMIT      c —         output character to console
            .drw link
            .set link,*+1
            .db  0,4,"EMIT"
EMIT:       jnb scon.1,EMIT ; await Tx interrupt flag
            clr scon.1      ; clear flag
            mov sbuf,dpl    ; output TOS char to UART
            ajmp poptos     ; pop new TOS

;C KEY       — c         get character from keyboard
            .drw link
            .set link,*+1
            .db  0,3,"KEY"
KEY:        jnb scon.0,KEY  ; await Rx interrupt flag
            clr scon.0
            dec r0          ; push old TOS
            mov a,dph
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            mov dpl,sbuf    ; get new char in TOS
            mov dph,#0
            ret

; ?KEY      — f         return true if char waiting
            .drw link
            .set link,*+1
            .db  0,4,"?KEY"
QUERYKEY:   dec r0          ; push old TOS
            mov a,dph
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            mov a,scon      ; get rx flag in carry
            rrc a
            ajmp cyprop     ; propagate that thru TOS

; INTERPRETER LOGIC
;==============================

; NEXT and ENTER are not needed for Subroutine
; Threading. EXIT may be used in high level code.

;C EXIT      —           exit a colon definition
            .drw link
            .set link,*+1
            .db  0,4,"EXIT"
EXIT:       dec sp      ; discard ret adrs in caller
            dec sp
            ret         ; return to caller's caller

;Z LIT       — x         fetch inline literal to stack
            .drw link
            .set link,*+1
            .db  0,3,"LIT"
LIT:        dec r0      ; push old TOS
            mov a,dph
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            pop dph     ; get return address
            pop dpl
            movx a,@dptr    ; get literal low byte
            inc dptr
            mov r2,a
            movx a,@dptr    ; get literal high byte
            inc dptr
            push dpl    ; restore updated ret adr
            push dph
            mov dph,a   ; put literal in TOS
            mov dpl,r2
            ret

;C EXECUTE  i*x xt — j*x     execute Forth word
;C          at 'xt'
            .drw link
            .set link,*+1
            .db  0,7,"EXECUTE"
EXECUTE:    push dpl    ; push addr onto r.stack,
            push dph    ; then pop new TOS->DPTR
                        ; 'ret' in poptos will then execute
                        ; desired word; its 'ret' will return
                        ; to EXECUTE's caller.
            ajmp poptos

; DEFINING WORDS
;================================

;C VARIABLE  —           define a Forth VARIABLE
;           CREATE CELL ALLOT ;
; Action of ROMable variable is that of CONSTANT;
; the constant holds the RAM address.
            .drw link
            .set link,*+1
            .db  0,8,"VARIABLE"
VARIABLE:   lcall CREATE
            acall CELL
            ljmp ALLOT

;C CONSTANT —            define a Forth constant
;CREATE CELL NEGATE IALLOT  I,  Harvard model
;           DOES> (machine code fragment)
; Note that the constant is stored in Code space.
            .drw link
            .set link,*+1
            .db  0,8,"CONSTANT"
CONSTANT:   lcall CREATE
            lcall CELL
            lcall NEGATE
            lcall IALLOT
            lcall ICOMMA
            lcall XDOES
; DOCON, code action of CONSTANT,
; entered by CALL DOCON
docon:      ; — x       exec action of constant
dovar:      ; — a-addr  exec action of ROMable var
docreate:   ; — a-addr  exec action of Harv.CREATE
            dec r0          ; push old TOS
            mov a,dph
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            pop dph         ; get addr of param field
            pop dpl         ;(in Code memory!)
```

```
            ajmp IFETCH     ; go fetch its contents

;Z USER     n —         define user variable 'n'
;           CONSTANT DOES> (machine code fragment)
; Note that this version allows a full 16-bit
; offset from the user pointer.
            .drw link
            .set link,*+1
            .db 0,4,"USER"
USER:       acall CONSTANT
            lcall XDOES
; DOUSER, code action of USER,
; entered by CALL DOUSER
douser: acall pushtos   ; push old TOS
            pop dph     ; get addr of param field
            pop dpl     ;       (in Code memory!)
            acall IFETCH    ; go fetch its contents
            add a,UP    ; add UP:00 to offset
            mov dph,a   ; NB. IFETCH leaves A=DPH
            ret

; DOCREATE's action is for a table in RAM.
; DOROM is the code action for a table in ROM;
; it returns the address of the parameter field.
; Entered by CALL DOROM
dorom:      acall pushtos   ; push old TOS
            pop dph     ; param field adrs -> TOS
            pop dpl
            ret

; DODOES, code action of DOES> clause
; (internal code fragment, not a Forth word)
; entered by  LCALL fragment
;             address of data
;             ...
;             fragment: LCALL DODOES
;             high-level thread
; Enters high-level thread with address of
; data on top of stack.  HARVARD MODEL: the data
; (in Data space) does NOT follow LCALL fragment
; (in Code space); instead, the address of the
; data is appended after LCALL fragment.
dodoes:     ; — a-addr  support routine for DOES>
            dec r0      ; push old TOS
            mov a,dph
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            pop dr5     ; addr of DOES> clause
            pop dr4     ;       Forth code
            pop dph     ; addr of defined word's
            pop dpl     ;       Param. field
            push dr4    ; restore Forth code addr
            push dr5
            ajmp IFETCH     ; fetch adrs from P.field
                            ;  & go do the DOES> code

; STACK OPERATIONS
; ================================

;C DUP      x — x x     duplicate top of stack
            .drw link
            .set link,*+1
            .db 0,3,"DUP"
DUP:
pushtos:    dec r0      ; push hi byte of TOS
            mov a,dph
            movx @r0,a
            dec r0      ; push lo byte of TOS
            mov a,dpl
            movx @r0,a
            ret

;C ?DUP     x — 0 | x x  DUP if nonzero
            .drw link
            .set link,*+1
            .db 0,4,"?DUP"
QDUP:       mov a,dph
            orl a,dpl
            jnz pushtos
            ret

;C DROP     x —         drop top of stack
            .drw link
            .set link,*+1
            .db 0,4,"DROP"
DROP:
poptos:     movx a,@r0      ; pop lo byte -> TOS
            mov dpl,a
            inc r0
            movx a,@r0      ; pop hi byte -> TOS
            mov dph,a
            inc r0
            ret

;C SWAP     x1 x2 — x2 x1   swap top two items
            .drw link
            .set link,*+1
            .db 0,4,"SWAP"
SWOP:       movx a,@r0      ; pop lo byte -> X
            mov r2,a
            inc r0
            movx a,@r0      ; pop hi byte -> X
            mov r3,a
            ; inc r0
            ; dec r0       ; push hi byte of TOS
halfover:   mov a,dph
            movx @r0,a
            dec r0          ; push lo byte of TOS
            mov a,dpl
            movx @r0,a
            mov dph,r3      ; old 2nd item -> TOS
            mov dpl,r2
            ret

;C OVER     x1 x2 — x1 x2 x1  per stack diagram
            .drw link
            .set link,*+1
            .db 0,4,"OVER"
OVER:       movx a,@r0      ; pop lo byte -> X
            mov r2,a
            inc r0
            movx a,@r0      ; pop hi byte -> X
            mov r3,a
            dec r0          ; restore stack pointer
            dec r0          ; predecrement for 'halfover'
            sjmp halfover   ; push TOS, then copy X to
TOS

;C ROT      x1 x2 x3 — x2 x3 x1 per stack diagram
            .drw link
            .set link,*+1
            .db 0,3,"ROT"
ROT:        ; x3 is in TOS
            movx a,@r0      ; pop x2 -> r5:r4
            mov r4,a
            inc r0
            movx a,@r0
            mov r5,a
            inc r0
            movx a,@r0      ; pop x1 -> r3:r2
            mov r2,a
            inc r0
            movx a,@r0
            mov r3,a
            ; inc r0
            ; dec r0       ; push x2
            mov a,r5
            movx @r0,a
            dec r0
            mov a,r4
            movx @r0,a
            dec r0          ; predecr. for 'halfover'
            sjmp halfover   ; push x3 (TOS), then
                            ; copy x1 to TOS

;C >R       x —         R: — x      push to return stack
            .drw link
            .set link,*+1
            .db 0,2,">R"
TOR:        pop dr3         ; save ret addr in r3:r2
            pop dr2
            push dpl        ; push lo byte*
            push dph        ; push hi byte*
            push dr2        ; restore ret addr
            push dr3
            sjmp poptos     ; pop new TOS
;* NB. stored lo:hi in regs because SP increments

;C R>       —  x        R: x —  pop from return stack
            .drw link
            .set link,*+1
            .db 0,2,"R>"
RFROM:      dec r0          ; push old TOS
            mov a,dph
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            pop dr3         ; save ret addr in r3:r2
            pop dr2
            pop dph         ; pop hi byte
            pop dpl         ; pop lo byte
            push dr2        ; restore return address
            push dr3
            ret

;C R@       — x  R: x — x  fetch from rtn stack
            .drw link
            .set link,*+1
            .db 0,2,"R@"
RFETCH:     dec r0          ; push old TOS
            mov a,dph
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            mov r1,sp       ; get copy of SP
            dec r1          ; skip return address
            dec r1
            mov dph,@r1     ; fetch 2nd return stack item
            dec r1
            mov dpl,@r1
            ret

;Z SP@      — a-addr   get data stack pointer
            .drw link
            .set link,*+1
            .db 0,3,"SP@"
SPFETCH:    dec r0          ; push old TOS
            mov a,dph
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            mov dph,UP      ; 16-bit pointer P2:R0
            mov dpl,r0
            ret

;Z SP!      a-addr —    set data stack pointer
; Note: only the low 8 bits are affected!
            .drw link
            .set link,*+1
            .db 0,3,"SP!"
SPSTORE:    mov r0,dpl      ; set stack pointer
            ajmp poptos     ; get new TOS

;Z RP@      — a-addr   get return stack pointer
            .drw link
            .set link,*+1
            .db 0,3,"RP@"
RPFETCH:    dec r0          ; push old TOS
            mov a,dph
            movx @r0,a
            dec r0
            mov a,dpl
            movx @r0,a
            mov dph,#0      ; 16-bit pointer 00:SP
            mov dpl,sp
            ret

;Z RP!      a-addr —    set return stack pointer
; Note: only the low 8 bits are significant!
            .drw link
            .set link,*+1
            .db 0,3,"RP!"
RPSTORE:    pop dr3         ; save ret addr in r3:r2
            pop dr2
            mov sp,dpl      ; set new stack pointer
            push dr2        ; restore ret addr
            push dr3
            ajmp poptos     ; get new TOS

;X NIP      x1 x2 — x2  per stack diagram
            .drw link
            .set link,*+1
            .db 0,3,"NIP"
NIP:        acall SWOP
            ajmp DROP

;X TUCK     x1 x2 — x2 x1 x2  per stack diagram
            .drw link
            .set link,*+1
            .db 0,4,"TUCK"
TUCK:       acall SWOP
            ajmp OVER

; MEMORY OPERATIONS
; ================================

;C !        x a-addr — store cell in Data mem
; Byte order is lo,hi.
            .drw link
            .set link,*+1
            .db 0,1,"!"
STORE:  movx a,@r0          ; low byte of X
            inc r0
            movx @dptr,a
            inc dptr
            movx a,@r0      ; high byte of X
            inc r0
            movx @dptr,a
            ajmp poptos     ; pop new TOS

;C C!       c c-addr — store char in Data mem
            .drw link
            .set link,*+1
```

```
            .db  0,2,"C!"
CSTORE:     movx a,@r0      ; low byte is char
            inc r0
            movx @dptr,a
            inc r0          ; skip high byte
            ajmp poptos     ; pop new TOS

;C @        a-addr — x  fetch cell from Data mem
; Byte order is lo,hi.
            .drw link
            .set link,*+1
            .db  0,1,"@"
FETCH:      movx a,@dptr    ; low byte
            mov r2,a        ; ..temporary stash
            inc dptr
            movx a,@dptr    ; high byte
            mov dpl,r2      ; copy to TOS (DPTR)
            mov dph,a
            ret

;C C@       c-addr — c  fetch char from Data mem
            .drw link
            .set link,*+1
            .db  0,2,"C@"
CFETCH:     movx a,@dptr
            mov dpl,a
            mov dph,#0
            ret

;C I!       x a-addr — store cell in Code mem
; On 8051, the only way to store to Code memory
; is to have it also appear in Data space.
; So, I! is identical to I, and IC! to C!.
            .drw link
            .set link,*+1
            .db  0,2,"I!"
ISTORE:     sjmp STORE

;C IC!      c c-addr — store char in Code mem
            .drw link
            .set link,*+1
            .db  0,3,"IC!"
ICSTORE: sjmp CSTORE

;Z I@       a-addr — x  fetch cell from Code mem
; Byte order is lo,hi.
            .drw link
            .set link,*+1
            .db  0,2,"I@"
IFETCH:     clr a
            movc a,@a+dptr  ; low byte
            mov r2,a        ; ..temporary stash
            mov a,#1
            movc a,@a+dptr  ; high byte
            mov dpl,r2      ; copy to TOS (DPTR)
            mov dph,a
            ret
; Note! USER expects IFETCH to leave A=DPH!

;Z IC@      a-addr — x  fetch char from Code mem
            .drw link
            .set link,*+1
            .db  0,2,"IC@"
ICFETCH:    clr a
            movc a,@a+dptr  ; low byte
            mov dpl,a
            mov dph,#0
            ret

; ARITHMETIC AND LOGICAL OPERATIONS
; =============

;C +        n1/u1 n2/u2 — n3/u3  add n1+n2
            .drw link
            .set link,*+1
            .db  0,1,"+"
PLUS:       movx a,@r0      ; low byte
            inc r0
            add a,dpl
            mov dpl,a
            movx a,@r0      ; high byte
            inc r0
            addc a,dph
            mov dph,a
            ret

;Z M+       d n — d     add single to double
            .drw link
            .set link,*+1
            .db  0,2,"M+"
MPLUS:      movx a,@r0      ; pop d.high -> r3:r2
            mov r2,a
            inc r0
            movx a,@r0
            mov r3,a
            inc r0
            movx a,@r0      ; d.low, low byte
            add a,dpl
            movx @r0,a
            inc r0
            movx a,@r0      ; d.low, high byte
            addc a,dph
            movx @r0,a
            dec r0
            clr a
            addc a,r2       ; d.high, low byte
            mov dpl,a
            clr a
            addc a,r3       ; d.high, high byte
            mov dph,a
            ret

;C -        n1/u1 n2/u2 — n3/u3  subtract n1-n2
            .drw link
            .set link,*+1
            .db  0,1,"-"
MINUS:      movx a,@r0      ; low byte
            inc r0
            clr c
            subb a,dpl
            mov dpl,a
            movx a,@r0      ; high byte
            inc r0
            subb a,dph
            mov dph,a
            ret

;C AND      x1 x2 — x3 logical AND
            .drw link
            .set link,*+1
            .db  0,3,"AND"
AND:        movx a,@r0      ; low byte
            inc r0
            anl a,dpl
            mov dpl,a
            movx a,@r0      ; high byte
            inc r0
            anl a,dph
            mov dph,a
            ret

;C OR       x1 x2 — x3 logical OR
            .drw link
            .set link,*+1
            .db  0,2,"OR"
OR:         movx a,@r0      ; low byte
            inc r0
            orl a,dpl
            mov dpl,a
            movx a,@r0      ; high byte
            inc r0
            orl a,dph
            mov dph,a
            ret

;C XOR      x1 x2 — x3 logical XOR
            .drw link
            .set link,*+1
            .db  0,3,"XOR"
XOR:        movx a,@r0      ; low byte
            inc r0
            xrl a,dpl
            mov dpl,a
            movx a,@r0      ; high byte
            inc r0
            xrl a,dph
            mov dph,a
            ret

;C INVERT   x1 — x2     bitwise inversion
            .drw link
            .set link,*+1
            .db  0,6,"INVERT"
INVERT:     xrl dpl,#h'ff
            xrl dph,#h'ff
            ret

;C NEGATE   x1 — x2     two's complement
            .drw link
            .set link,*+1
            .db  0,6,"NEGATE"
NEGATE:     xrl dpl,#h'ff
            xrl dph,#h'ff
            inc dptr
            ret

;C 1+       n1/u1 — n2/u2     add 1 to TOS
            .drw link
            .set link,*+1
            .db  0,2,"1+"
ONEPLUS:    inc dptr
            ret

;C 1-       n1/u1 — n2/u2     subtract 1 from TOS
            .drw link
            .set link,*+1
            .db  0,2,"1-"
ONEMINUS: mov a,dpl
            jnz dphok
            dec dph         ; if dpl=0, decr. affects dph
dphok:      dec dpl
            ret

;Z ><       x1 — x2     swap bytes (not ANSI)
            .drw link
            .set link,*+1
            .db  0,2,"><"
swapbytes:  mov a,dpl
            mov dpl,dph
            mov dph,a
            ret

;C 2*       x1 — x2     arithmetic left shift
            .drw link
            .set link,*+1
            .db  0,2,"2*"
TWOSTAR:    mov a,dpl       ; lo byte, left shift
            add a,dpl
            mov dpl,a
            mov a,dph       ; hi byte, left rot w/cy
            rlc a
            mov dph,a
            ret

;C 2/       x1 — x2     arithmetic right shift
            .drw link
            .set link,*+1
            .db  0,2,"2/"
TWOSLASH: mov a,dph       ; get msb of TOS into cy
            rlc a
            mov a,dph       ; high byte, right rotate
            rrc a
            mov dph,a
            mov a,dpl       ; low byte, right rotate
            rrc a
            mov dpl,a
            ret

;C LSHIFT   x1 u — x2   logical left shift
            .drw link
            .set link,*+1
            .db  0,6,"LSHIFT"
LSHIFT:     mov r4,dpl      ; r4 = loop counter
            movx a,@r0      ; pop x1 -> DPTR
            mov dpl,a
            inc r0
            movx a,@r0
            mov dph,a
            inc r0
            inc r4          ; test for r4=0 case
            sjmp lshtest
lshloop:    mov a,dpl       ; shift left
            add a,dpl
            mov dpl,a
            mov a,dph
            rlc a
            mov dph,a
lshtest:    djnz r4,lshloop
            ret

;C RSHIFT   x1 u — x2   logical right shift
            .drw link
            .set link,*+1
            .db  0,6,"RSHIFT"
RSHIFT:     mov r4,dpl      ; r4 = loop counter
            movx a,@r0      ; pop x1 -> DPTR
            mov dpl,a
            inc r0
            movx a,@r0
            mov dph,a
            inc r0
            inc r4          ; test for r4=0 case
            sjmp rshtest
rshloop:    clr c           ; clear carry
            mov a,dph       ; shift right
            rrc a
            mov dph,a
            mov a,dpl
            rrc a
            mov dpl,a
rshtest:    djnz r4,rshloop
            ret

;C +!       n/u a-addr —    add cell to Data mem
            .drw link
            .set link,*+1
            .db  0,2,"+!"
PLUSSTORE: movx a,@r0     ; low byte of n
```

```
            inc r0
            mov r2,a
            movx a,@dptr    ; low byte of memory
            add a,r2
            movx @dptr,a
            inc dptr
            movx a,@r0      ; high byte of n
            inc r0
            mov r2,a
            movx a,@dptr    ; high byte of memory
            addc a,r2
            movx @dptr,a
            ajmp poptos     ; pop new TOS

; COMPARISON OPERATIONS
; =====================

;X <>       x1 x2 — flag    test not equal
            .drw link
            .set link,*+1
            .db 0,2,"<>"
NOTEQUAL: acall EQUAL
            sjmp ZEROEQUAL

;C 0=       n/u — flag  return true if TOS=0
            .drw link
            .set link,*+1
            .db 0,2,"0="
ZEROEQUAL: mov a,dph
zequ1:      orl a,dpl       ; A = z or nz, per DPTR
            clr c
            subb a,#1       ; cy set if A was 0
cyprop: subb a,acc          ; -1 if A was 0, else 0
            mov dph,a
            mov dpl,a
            ret             ; NB! A=0 iff TOS=0

;C 0<       n — flag    true if TOS negative
            .drw link
            .set link,*+1
            .db 0,2,"0<"
ZEROLESS: mov a,dph
            rlc a           ; cy set if A negative
            sjmp cyprop     ; propagate cy thru TOS

;C =        x1 x2 — flag    test x1=x2
            .drw link
            .set link,*+1
            .db 0,1,"="
EQUAL:      acall MINUS     ; x1-x2 in TOS, A=DPH
            sjmp zequ1

;C <        n1 n2 — flag    test n1<n2, signed
            .drw link
            .set link,*+1
            .db 0,1,"<"
LESS:       acall MINUS     ; n1-n2 in TOS, A=DPH,
                            ; CY and OV valid
; if result negative (MSB=1) & not OV, n1<n2
; neg. & OV => n1 +ve, n2 -ve, result -ve, n1>n2
; if result positive (MSB=0) & not OV, n1>=n2
; pos. & OV => n1 -ve, n2 +ve, result +ve, n1<n2
; thus OV reverses the sense of the sign bit
            jnb psw.2,msbok ; jump if overflow clear
            cpl a           ; OV set: invert msb
msbok:      rlc a           ; put msb (sign) in cy
            sjmp cyprop     ; & propagate thru TOS

;C >        n1 n2 — flag    test n1>n2, signed
            .drw link
            .set link,*+1
            .db 0,1,">"
GREATER:  acall SWOP
            sjmp LESS

;C U<       u1 u2 — flag    test n1<n2, unsigned
            .drw link
            .set link,*+1
            .db 0,2,"U<"
ULESS:      acall MINUS     ; TOS=u1-u2, cy set if u1<u2
            sjmp cyprop     ; propagate cy thru TOS

;X U>       u1 u2 — flag    test u1>u2, unsigned
            .drw link
            .set link,*+1
            .db 0,2,"U>"
UGREATER: acall SWOP
            sjmp ULESS

; LOOP AND BRANCH OPERATIONS
; ==========================

; branch and ?branch are done with sjmp and jz,
; respectively, using the following routines
; which leave a value in A.  Typical use:
;           lcall zerosense; jz destadr
;           lcall loopsense, jz destadr, lcall unloop
; LEAVE may exit loop by branching ^—here

            .drw link
            .set link,*+1
            .db 0,7,"?BRANCH"
qbranch:
zerosense:              ; n — leave zero in A if TOS=0
            movx a,@r0      ; new TOS in a:r2
            mov r2,a
            inc r0
            movx a,@r0
            inc r0
            xch a,dph       ; DPH=new TOS hi, A=old DPH
            orl a,dpl       ; A=0 if old TOS was zero
            mov dpl,r2      ; new TOS lo in DPL
            ret

; LOOP and +LOOP are done with jz, using the
; following routines which leave a value in A.
; If the loop terminates, (index crosses 8000h),
; a nonzero value is left in A.  A=0 to loop.
; Typical use:
;           lcall loopsense, jz destadr, lcall unloop
; LEAVE may exit loop by branching ^-here
; The topmost loop index is in regs r7:r6.

            .drw link
            .set link,*+1
            .db 0,6,"(LOOP)"
xloop:
loopsense:              ; —  leave 0 in A if 'loop'
            mov a,r6    ; add 1 to loop index
            add a,#1    ; ...leaves OV flag set if
            mov r6,a    ; loop terminates
            mov a,r7
            addc a,#0
            mov r7,a
            jb psw.2,termloop ; jump if OV set
takeloop:   clr a       ; OV clear, make A zero
            ret         ;       to take loop branch

            .drw link
            .set link,*+1
            .db 0,7,"(+LOOP)"
xplusloop:
plusloopsense:          ; n — leave 0 in A if '+loop'
            mov a,r6    ; add TOS to loop index
            add a,dpl   ; ...leaves OV flag set if
            mov r6,a    ; loop terminates
            mov a,r7
            addc a,dph
            mov r7,a
            movx a,@r0  ; pop new TOS, OV unaffected
            mov dpl,a
            inc r0
            movx a,@r0
            mov dph,a
            inc r0
            jnb psw.2,takeloop ; jump if OV clear
termloop:   clr a       ; OV set, make A nonzero
            cpl a       ; to force loop termination
            ret
```

We will continue with the listing in the next issue, otherwise the full source is on GEnie. BDK.

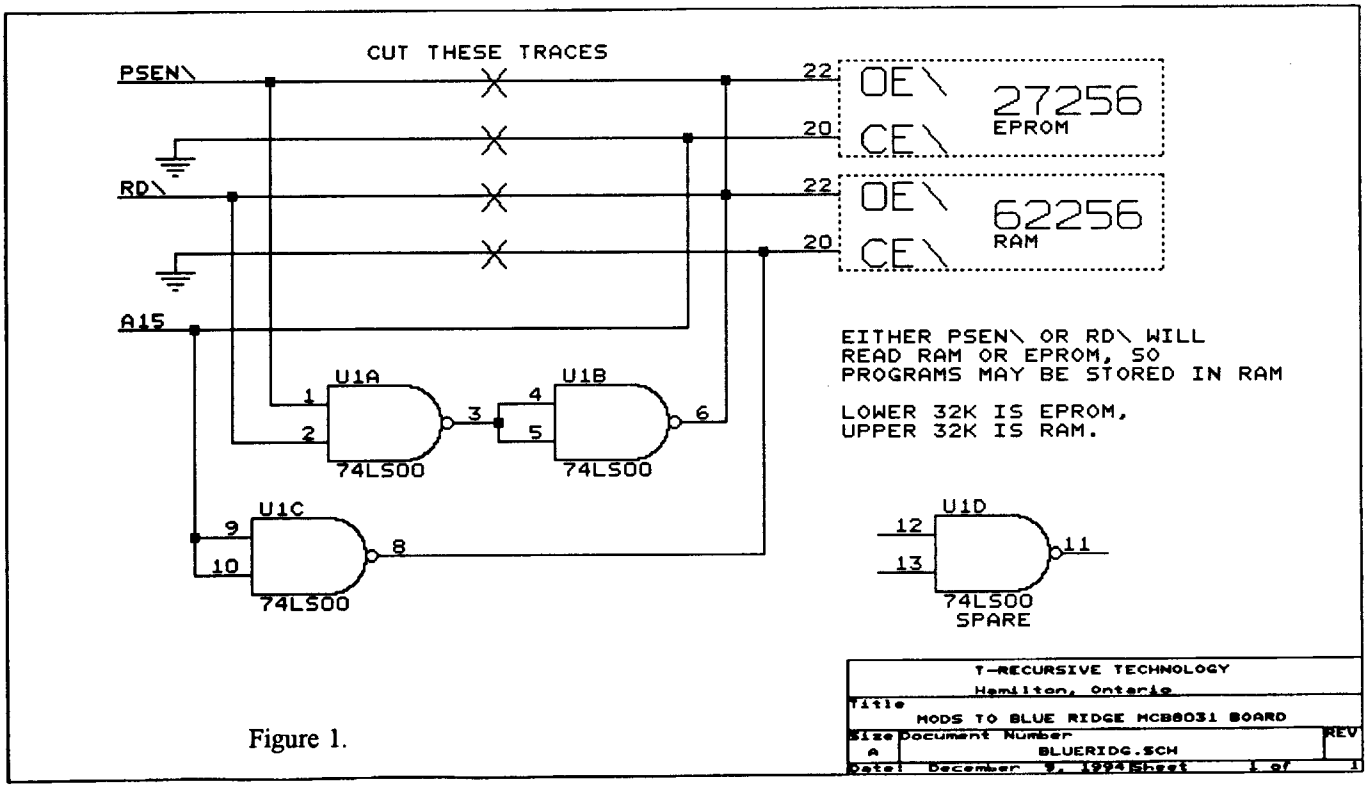


Figure 1.

SUPPORT GROUPS FOR THE CLASSICS

Regular Feature

Group Reviews

On November 18th, 1994, the Forth Interest Group (FIG) had their annual Forth Day meeting in the San Francisco Bay Area. At one time this had been an international event, but the last few years the organization has down played the meeting, and thus only a few local people attend.

The last several Forth Day's as did this one, started at 10AM with a list of speakers. George Nichols of Silicon Composers started the talks by updating the group on his Formula 1 (Indy Racing car class) computer. The computer is used to monitor and control various parameters in the racing car. He is using his SC32, a 32 bit Forth engine. It is called the "Formula Data Logger 32 (tm)". It runs SC32 a 32 bit version of F83.

Next up was Robert Kylberg of Mosaic Industries with a report on their QED board. It uses a 68HC11 with up to 8Megs of RAM. He explained their page switch concept and why so much RAM was needed. The reasoning was a complete system with all math functions on board (floating point and all). Their idea was not to collect data for later analysis, but do all the analysis on line with 160us Floating point transcendentals.

Albert Mitchell of AMR brought his AMR166LC single board system. AMR makes a full line of embedded systems, and the 166 is their latest 16 bit product. They use a tethered Forth with a small kernel on the board, and all other functions on a PC running their version of FPC. AMR tested many options in the 16 bit world before deciding to settle on the Siemens 80C166 CPU. It provides better speed and many instructions that are Forth like. It is suppose to be 3 times faster than the 68332.

Dr. Ting stepped up next and talked about the MuP21. This is Chuck Moore's latest single chip CPU that he has designed on his own CAD system. A more detailed review follows later.

Talks continued with John Carpenter explaining how Post Script Forth engines can be done. Leonard Morgenstern reviewed his latest work on "Brute Forth" search algorithms. Tom Zimmer talked about his latest work and gave out sample disks of it. The "it" is a Windows version of FPC, the "why" being a port of a current product to Windows NT. It currently has 5000 words!

Next was my favorite presenter David Jaffe and his talking hand. He came to our local club meeting the following month and you need to read our 1994 Computer Hero article. Another favorite was Chuck Moore who presented some hardware talk about his MuP21. Chuck explained his design style or lack there of. It seems he has decided that documenting his work locks him into a design strategy that might not always be the best, so he does it "free form" so to speak.

John Rible presented information on the OPEN BOOT project and how it will use Forth programmers. The Open Boot project is the basis behind the new PCI interface in PC's. The plug in board has a Forth in ROM that contains all the diagnostic and boot software for the system to bring up and install the card. SUN's have been doing this for sometime now and it works so well that the PC manufacturers decided they needed some way for users to easily install accessory cards, PCI is it. For more information get IEEE draft report P1275.

At this point Lunch was held and then we started with more talks. It seemed the meeting was a little sleepy after lunch or maybe it was the great food that slowed everyone down. My notes say that Skip Carter talked about ANSI Standard Forth Library project. He indicated that a ANSI group is getting standard libraries together and many have been collected to date.
Wil Baden talked about macro processing in Forth and his ThisForth program. Andrew McKewan talked about some reasons to consider C, especially the C built Forth's, like Wil Baden's ThisForth, Dirk Zoeller's PFE or TIMBRE by Rob Chapman. Another Forth version presented was LibForth which contains a built in B-Tree sort of words in the dictionaries. The sort gives very fast and more detailed definitions all without documentation.

Glen Haydon concluded the talks with some comments of Forth philosophy. Some of his points were; Keep it short and simple, use a forth appropriate to the task, get beginners started with simple versions. From this point we left the meeting and went to dinner with Chuck Moore as feature speaker.

Chuck has always presented a "Fireside Chat" that gets one thinking. In fact this chat was just that, about thinking and using the right and left sides of the brain. Most consider the right side of the brain the side that does creative work. Chuck pointed out that real programmers use the right side of their brain, that being good programs come from creative activities.

Chuck moved on to discuss the idea of what computing should and can do for us. He moved past the normal ideas and centered more on the concept of thinking machines. Can a machine

actually think for itself? Chuck didn't answer that, but he did give everyone many interesting ideas to consider and ponder over. If you have never attended one of Chuck's fireside chats, my notes probably have little impact on you. For those of us lucky enough to sit and listen, he has never failed to stir the thinking juices and then some of those present. I recommend him highly.

MPU21

I received a rather long E-Mail with information on the MPU21. Space does not permit me to print it in this issue, so I will do the new device next time. BDK.

TCJ Staff Contacts

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

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

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

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

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.

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

Regular Contributors:
Dave Baldwin, Voice/FAX (916)722-3877, or DIBs BBS (916) 722-5799 (use "computer", "journal", pswd "subscriber" as log on), Internet dibald@netcom.com, CompuServe 70403,2444.

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

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

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

Helmut Jungkunz, Munich, Germany, ZNODE #51, 8N1, 300-14.4, +49.89.9614574, or CompuServe 100024,1545.

Ron Mitchell, Apt 1107, 210 Gloucester St., Ottawa Ontario, Canada, K2P 2K4. GEnie as R.Mitchell31, or CompuServe 70323,2267.

USER GROUPS

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

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

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 and BBS.

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

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

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

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

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

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.

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. Contact for list of local chapters.

The Pacific Northwest Heath Users Group, contact Jim Moore, PO

Box 9223, Seattle, WA 98109-0223.

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, (503)688-3563. Bi-Monthly user oriented newsletter (20 pages+). Also sells CP/M Boot disks, software.

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

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

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

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

Historically Brewed, A publication of the Historical Computer Society. Bimonthly at $18 a year. HCS, 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.

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

Other Support Businesses

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

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

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

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. 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. See inside front cover.

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

AAA Chicago Computers, Jerry Koppel, (708)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.

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

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

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Volume Number 1:
- **Issues 1 to 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: Operating system extensions to BDOS and BIOS, RSXs for CP/M 2.2.
- Macintosh Data File Conversion in Turbo Pascal.

Issue Number 35:
- All This & Modula-2: A Pascal-like alternative with scope and parameter passing.
- A Short Course in Source Code Generation: Disassembling 8088 software to produce modifiable assem. source code.
- Real Computing: The NS32032.
- S-100: EPROM Burner project for S-100 hardware hackers.
- Advanced CP/M: An up-to-date DOS, plus details on file structure and formats.
- REL-Style Assembly Language for CP/M and Z-System. Part 1: Selecting your assembler, linker and debugger.

Issue Number 36:
- Information Engineering: Introduction.
- Modula-2: A list of reference books.
- Temperature Measurement & Control: Agricultural computer application.
- ZCPR3 Corner: Z-Nodes, Z-Plan, Amstrand computer, and ZFILE.
- Real Computing: NS32032 hardware for experimenter, CPUs in series, software options.
- SPRINT: A review.
- REL-Style Assembly Language for CP/M & ZSystems, part 2.
- Advanced CP/M: Environmental programming.

Issue Number 37:
- C Pointers, Arrays & Structures Made Easier: Part 1, Pointers.
- ZCPR3 Corner: Z-Nodes, patching for NZCOM, ZFILER.
- Information Engineering: Basic Concepts: fields, field definition, client worksheets.
- Shells: Using ZCPR3 named shell variables to store date variables.
- Resident Programs: A detailed look at TSRs & how they can lead to chaos.
- Advanced CP/M: Raw and cooked console I/O.
- Real Computing: The NS 32000.
- ZSDOS: Anatomy of an Operating System: Part 1.

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

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

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

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

Issue Number 42:
- Dynamic Memory Allocation: Allocating memory at runtime with examples in Forth.
- Using BYE with NZCOM.
- C and the MS-DOS Screen Character Attributes.
- Forth Column: Lists and object oriented Forth.
- The Z-System Corner: Genie, BDS Z and Z-System Fundamentals.
- 68705 Embedded Controller Application: An example of a single-chip microcontroller application.
- Advanced CP/M: PluPerfect Writer and using BDS C with REL files.
- Real Computing: The NS 32000.

Issue Number 43:
- Standardize Your Floppy Disk Drives.
- A New History Shell for ZSystem.
- Heath's HDOS, Then and Now.
- The ZSystem Corner: Software update service, and customizing NZCOM.
- Graphics Programming With C: Graphics routines for the IBM PC, and the Turbo C graphics library.
- Lazy Evaluation: End the evaluation as soon as the result is known.
- S-100: There's still life in the old bus.
- Advanced CP/M: Passing parameters, and complex error recovery.
- Real Computing: The NS32000.

Issue Number 44:
- Animation with Turbo C Part 1: The Basic Tools.
- Multitasking in Forth: New Micros F68FC11 and Max Forth.
- Mysteries of PC Floppy Disks Revealed: FM, MFM, and the twisted cable.
- DosDisk: MS-DOS disk format emulator for CP/M.
- Advanced CP/M: ZMATE and using lookup and dispatch for passing parameters.
- Real Computing: The NS32000.
- Forth Column: Handling Strings.
- Z-System Corner: MEX and telecommunications.

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

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

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

The Computer Journal Back Issues

- 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
- Real Computing
- Z-System Corner: Patching MEX-Plus and TheWord, Using ZEX

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

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

Issue Number 51:
- Introducing the YASBEC
- Floppy Disk Alignment w/RTXEB, Pt 3
- High Speed Modems on Eight Bit Systems
- A Z8 Talker and Host
- Local Area Networks—Ethernet
- UNIX Connectivity on the Cheap
- PC Hard Disk Partition Table
- A Short Introduction to Forth
- Stepped Inference 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
- Controlling Home Heating & Lighting, Pt. 3
- The CPU280, A High Performance Single-Board Computer

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

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

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

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

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

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

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

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

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

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

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

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

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

Issue Number 66:
- Small System Support
- Center Fold: Advent Decoder
- DR S-100
- Real Computing
- 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
- Real Computing
- 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
- Real Computing
- 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 69:
- Small System Support
- Center Fold: Jupiter ACE
- Z-System Corner II
- Real Computing
- PC/XT Corner: Stepper Motors
- DR. S-100
- Multiprocessing Part 5
- European Beat

SPECIAL DISCOUNT
15% on cost of Back Issues when buying from 1 to Current Issue.
10% on 10 or more issues.

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

Items: _____

_____

_____ Back Issues Total _____

_____ Shipping Total _____

California state Residents add 7.25% Sales TAX _____

Subscription Total _____

Total Enclosed _____

Name: _____

Address: _____

_____

_____

_____

Credit Card # _____-_____-_____-_____ exp ____/____

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

TCJ The Computer Journal

P.O. Box 535, Lincoln, CA 95648-0535
Phone (916) 645-1670

Regular Feature

Editorial Comment

PLC or 8051?

The Computer Corner

By Bill Kibler

Over the past few issues I have talked about using PLCs. At work I have been programming them for door control. Currently I am moving on to another form of PLC, mainly embedded controllers. These are 8051 based small controllers that do door control just the same as the larger PLC.

I surmise that plenty of companies are using their own or many of the off the shelf products for PLC like operations. In the past PLCs required that you buy them from large corporations, spend big amounts of money getting set up, and spend even more money when trying to upgrade either size or features.

Lately the PLC industry has changed with smaller, cheaper, and direct sales operations. I have one such direct sales system and will be testing and playing with it later this year. Our normal method of determining cost is per point. By that we mean what was the overall cost divided by the number of points available. This gets hard to figure since there is typically a base cost that remains the same regardless of points, type of points vary (AC vs DC), and expansions beyond base size adds extra cost.

Lets do a simple cost breakdown to see how it might compare to doing 8051 alternatives. PLC Direct (1-800-633-0405) is currently the bottom line system and anyone with a credit card can get them. Lets do a simple system of 16 doors. That means we have 16 input switch, 16 output lights or LED's, 16 DC outputs to control relays, and 16 inputs to see if the door is open. This is a very small system, but typical of the design requirements.

From PLC Direct we have three choices, bottom line would provide little expansion options, middle size might handle twice the design, and larger systems that could handle considerably more. Now their base unit cost is only a little lower that other vendors, but their I/O modules run about half everyone else's cost. When figuring on a per point base, the small units cost around $7, mid is $10, and large is $14. Regular vendors price might be $12 for small, $17 for mid, and $19 for large systems.

These I/O units can be either transistor, where you drive the device (typically a relay directly), optoisolator devices to drive other transistor circuits, or relays of the 12, 24, or 110 volt variety. If Large currents are to be controlled, you must use the small relay to drive a big relay or contactor (a REALLY big relay assembly). All these variations have a different pricing per point.

One cost that must be figured in is software (to download your program) or a hand programmers (to entry to program steps one at a time). PLC Direct's software is $495 and handles all of their modules. Other vendors will hit you somewhat the same, but might require a different package for each size used. When adding the software cost, small becomes $15 per point, mid is $18, and large $22. Remember that it is a one time charge, so it is high for a single job, but low if you use it many times. Hand programmers (calculator like device for entering program steps one at a time) can run from same price as software to half and in some cases even more.

8051 PLC

Now some of these PLC designs do use 8051's (and Z80s) inside. But what I am talking about here, is buying a small development system based on the 8051 chip. Since I have AMR's sales literature (see ad inside back cover) at hand, I'll do cost figures based on it. Now the first problem I encounter is figuring which product to get. Since I am interested in I/O points, I look for the number of in and out ports available.

In AMR's last flyer, they list the boards based on types of 8051 used. That can be a 80C51, 80C451, 80C751, 80C652, or 80CE558 (and more). Only in one place do I see 7 I/O ports as a feature, so I would guess it would be closes to my needs (64 I/O points). The 7 8bit ports are on the 80C451 module and I see two cost to start with, $275 for hardware only, or $595 for an entire development package that includes everything I need to get started. The regular single board price is $109. Now remember I am using AMR pricing simply because it is at hand, there are many other vendors all with pricing and systems that are about the same (features and options vary, but pricing will following the differences too).

So what is the cost per point, about $2 for single boards, $5 for hardware development system, and $10 for the full development package. Keep in mind the cost is based on 56 points not 64. So if we were to actually do 64, some multiplexing and latching would be needed. We could add those to the wire wrapping area available on each board, or for more runs build a daughter board. In either case we add some more cost to each board to get more points, but if we compare the base cost and our added circuitry, $3 per point might be pushing it. The nearest PLC cost is $7 per point, and most were considerably higher.

Of course we will have the same design problem of deciding on which I/O circuits are needed. The I/O on the 8051 is TTL non-isolated outputs. For real world use, we will need optoisolators, possibly driver circuits to operate relays, screw terminals to connect to wires (you can get a PLCs that have screw terminals on their I/O cards). Don't forget to order extra power supplies to run the output and input circuits on, different from that driving the 8051 (some PLC's provide 24 volts for that use).

As you can see with the hardware side of doing PLC operations, you will need other devices that are usually part of the cost in getting a PLC. There are companies that sell just the interface cards that can be driven by your TTL computer, but expect to add five dollars a point (or more) for the extra cards.

Alternatives

For one up projects, an old classic system or PC clone might work great. Any parallel port can be multiplexed to provide more outputs. Serial links to serial to parallel converters is possible. Even serial to 8051s could be done. Lets take a Kaypro from a local swap meet, say $50 (prices are going up on these not down). We add multiplex expander on the centronics port, using a handful of TTL chips, wire wrapped (or soldered-my choice) on perf board which adds $5. I would do a 4 bit decode that maps the lower 4 data bits, for a total of 64 points (4bits * 16 addresses), add some driver transistors like Henry did in issue 70's Etch-A-Sketch project ($10- his total cost for 8 points was $4).

Now the cost is over $2 per point in the above example. If the machine was just sitting around, your cost might be $.20 per point and some time. I talked to some people that were doing security systems using used CoCo's. They were buying them at garage sales, making it is easily to see why they could under bid any competitor using real PLCs. Say we got them at $10 each, added new cases ($10), some expanding circuitry ($5), and a few other items to make the total cost $32, that means $.50 a point. I dare any PLC vendor to match that.

Software

The next big problem however is how do you program these cheaper machines. Basically PLC's are the simplest to program. All you do is enter labels or an address for the switch inputs and coil outputs. The programming is very straight forward and could be learned by almost anyone with a fundamental knowledge of electricity and relay operations.

Programming 8051's is not quite the same. You must write your own program from scratch. I provided some Forth PLC code ideas some time back. The idea was to provide a method where tables might be used to contain the points and a simple series of routines would read through the table performing operations. The normal method is to do straight line coding for reading points and turning on relays. This mean any change in design would require changes in coding. Table operation changes do not effect the code, just the processing of the table, which is pretty much how PLC's do it. In PLC design, you really just add information into tables that are scanned, computations made (results placed on stack) and outputs performed (top of stack 1 then do relay ON).

Can we come up with some simple programming procedures to create our own PLC in Forth or Basic, I think so. What features do we need? Well primarily it must be able to read single input points and turn on or off single output points. That is if we want to do a normal PLC ladder type of program. If we know that things happen in groups, then word oriented programming is possible. Whatever the design, it will have to wait till next time and after I get some comments from you.

Till later, keep hacking.

From **Small System Support**, his final words and more ideas on PLCs...

the start button and turns the output on and leaves it on until you press the stop button. Of course you can do much more complex logic with one of these devices. Some of them support dozens if not hundreds of outputs and inputs. If anyone out there is interested, I'll devote part of this column for a while to such a project. Of course you can run one of these ports, and thus the project, using an antique PC XT so such an article will qualify for publication in *TCJ*.

By the way, anyone out there have a copy of Microsoft C/C++ version 6.0 that you want to sell? I want and need a copy for a project and would gladly buy one, but Microsoft doesn't sell it anymore. The new version is "Visual C" or something like that, has all kinds of (ugh) Windows support, and takes many megabytes of hard disk space. I am not asking for a bootleg copy. I would expect to receive original disks and manuals for a valid serial numbered copy. I'm only interested if you've moved on to bigger or different and more interesting things and have a copy you don't use anymore.

I feel the same way about Borland. I have Turbo C++ 1.0 at home and 3.0 at work. We recently upgraded to Borland C++ 4.0 (I think) and found that it does nothing more than 3.0 except that you HAVE TO run it under Windows. Perhaps it has a lot of Windows support in the way of library functions, but we are writing DOS applications so that is all just excess baggage.

I guess as more and more of our people come on line to write DOS applications we will have to appeal like this, to find old copies of 3.0 to buy, write to Borland (who MIGHT be going bankrupt soon) or MicroSoft and BEG them to sell us copies of their old software or let us make copies of what we have for a license fee of some sort. I have a feeling that there are many embedded systems companies around who are in the same boat. The software suppliers have to come up with bigger and BIGGER packages to entice people to buy "upgrades". I'd like to give a good try to doing it the honest way. If we can no longer buy the nicer simpler compiler and the suppliers won't negotiate licences, we'll simply be forced to make extra copies for our programmers.

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

Send your items to:
The Computer Journal
P.O. Box 535
Lincoln, CA 95648-0535

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

Wanted: Good complete floating-point package (IEEE single and/or double precision) for the 8051 Micro. Should be public domain, but commercial better than nothing. Send info to tilmann.reh@hrz.uni-siegen.d400.de.

Wanted: TURBO Pascal for CP/M-86, and a communications program for CP/M-86; also looking for other languages (esp. Forth with file interface) and utilities for CP/M-86. Address correspondence to: Douglas P. Beattie, Jr.; P.O. Box 47, Oak Harbor, WA 98277-0047.

Notice: Historically Brewed has moved. The new address is : 2962 Park Street #1, Jacksonville, FL 32205.

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 2X, P.N. 81-025, looks like a small suitcase. Has 2 disk drives, keyboard, monitor. Has previous owner's social security number on it. Appearance is nice, lights up, no guarantee. Want or trade Ham equipment, grid dip meter, noise bridge, SWR meter, old transmitter crystals, 4 pin tubes, what have you? $60.00 including UPS to lower 48 states. Roger Grosser, RFD 1, Sutton, Vermont 05867.

WANTED

TCJ Needs an FTP site with 1 Gig or more space to collect OLD BIOS source files for possible CD-ROM. Accessing same file space by regular BBS is also very dersirable! If you have the facilities and would like to help continue the computer restoration of older systems, please contact:

Bill Kibler
Editor
The Computer Journal
PO Box 535
Lincoln CA 95648
B.Kibler@GEnie.geis.com

SUPPORT OUR ADVERTISERS TELL THEM "I SAW IT IN TCJ"