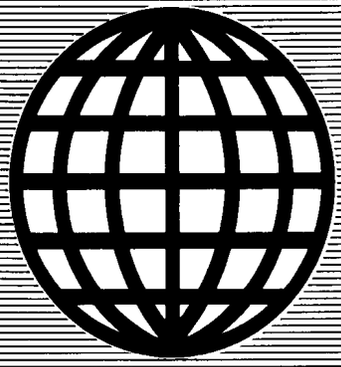


The COMPUTER JOURNAL

Programming - User Support
Applications

Issue Number 55 May / June 1992 US$3.95

ISSN # 0748-9331

Now $4.95 Stops The Clock On Over 100 GEnie Services

For the first time ever, enjoy unlimited non-prime time* usage of many popular GEnie℠ Service features. For just $4.95 a month. Choose from over 100 valuable services including everything from electronic mail and stock closings to exciting games and bulletin boards. Nobody else gives you so much for so little.

You can also enjoy access to a wide variety of features like software libraries, computer bulletin boards, multi-player games, Newsbytes, and the Computer Assisted Learning Center (CALC) for just $6.00 per non-prime hour for all baud rates including 2400. That's less than half of what some other services charge. Plus with GEnie there's no sign-up fee.

Now GEnie not only gives you the information and fun you're looking for. But the time to enjoy them, too.

Follow these simple steps.

1. Set your modem for half duplex (local echo), at 300, 1200 or 2400 baud.
2. Dial toll free 1-800-638-8369. Upon connection, enter HHH.
3. At the U#=prompt, enter XTX99486,GENIE then press RETURN
4. Have a major credit card or your checking account number ready.

For more information in the U.S. or Canada, call us voice at 1-800-638-9636.

TCJ readers are invited to join us in the CP/M SIG on page 685 and the Forth Interest Group SIG on page 710. Meet the authors and editors of *The Computer Journal!* Enter "M 710" to join the FIG group and "M 685" to join the CP/M and Z-System group.

We'll meet you there!

JUST $4.95

Moneyback Guarantee

Sign up now. If you're not satisfied after using GEnie for one month we'll refund your $4.95.

*Applies only in U.S. Mon.-Fri., 6PM-8AM local time and all day Sat., Sun., and select holidays. Prime time hourly rates $18 up to 2400 baud. Some features subject to surcharge and may not be available outside U.S. Prices and products listed as of Oct.1, 1990 subject to change. Telecommunications surcharges may apply. Guarantee limited to one per customer and applies only to first month of use. GE Information Services, GEnie, 401 N. Washington Street, Rockville, MD 20850. © 1991 General Electric Company.

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Chris McEwen

Technical Consultant
William P. Woodall

Contributing Editors
Bill Kibler
Matt Mercaldo
Tim McDonough
Frank Sergeant
Brad Rodriguez
Clem Pepper
Richard Rodman
Jay Sage

The Computer Journal is published six times a year by Socrates Press, P.O. Box 12, S. Plainfield, NJ 07080. (908) 755-6186

Subscription rates• Within US: $18 one year (6 issues), $32 two years (12 issues). Foreign (surface rate): $24 one year, $44 two years. Foreign (airmail): $38 one year, $72 two years. All funds must be in U.S. dollars drawn on a U.S. bank.

Send subscription, renewals, address changes, or advertising inquires to: *The Computer Journal*, P.O. Box 12, S. Plainfield, NJ 07080, telephone (908) 755–6186.

Registered Trademarks

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

Apple II, II+, IIc, IIe, Lisa, Macintosh, DOS 3.3, ProDos; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. DateStamper, BackGrounder II, Dos Disk; Plu*Perfect Systems. Clipper, Nantucket; Nantucket, Inc. dBase, dBASE II, dBASE III, dBASE III Plus, dBASE IV; Ashton-Tate, Inc. MBASIC, MS-DOS, Windows, Word; MicroSoft. WordStar; MicroPro International. IBM-PC, XT, and AT, PC-DOS; IBM Corporation. Z80, Z280; Zilog Corporation. Turbo Pascal, Turbo C, Paradox; Borland International. HD64180; Hitachi America, Ltd. SB180; MicroInt, Inc.

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

TCJ The Computer Journal

Issue Number 55 **May / June 1992**

Editor's Desk

By Chris McEwen

By necessity, my column in this issue is going to be very brief. My health has not been good the last few months and I have spent several weeks in the hospital. As a result, we have skipped the March/April issue and gone straight to May/June. You will not lose an issue from your subscription, however, as the expirations are set to issue number, not month.

In the meantime, please pardon me if the magazine is not right on time over the next few months. It will take some time for my health to fully recover. Your understanding is appreciated.

A new author joins us. Walter Rottenkolber discusses the cyclic redundancy check and gives us Forth code for it. We also welcome back Frank Sergeant with his *Bumbling Mathematician*.

Matt Mercaldo opens a topic we haven't covered in a while: fuzzy logic. I hope to see more on this. Lee Hart has expressed a desire to write more on it.

Add in Paul Chidley's work with the Z180 YASBEC and his new column *Hardware Hacker*, Roger Warren's work with the NZCOM virtual BIOS, Wayne Sung's discussion of IP along with Bill Kibler, Jay Sage, Rick Rodman and Bill Tishey and I'd say we have a pretty good way to spend a few evenings. Enjoy!●

Reader-to-Reader

I noticed a few things in my 6809 assembler article that should be corrected:

a) The table of valid indexed addressing modes (p.8, first column) shows ",—" as the "autodecrement by 2" mode indicator; it should be ",--" (two minus signs). I think this happened at your end.
b) The Forth fragment on page 12, for the word CODE, should read CREATE HERE DUP 2- 1. The DUP was omitted from the article.
c) You neglected to mention that the source code for the assembler is available on GEnie's Forth Roundtable as 6809ASM.ZIP. (Tsk, tsk...shame on you.)

Also, here is an update on the Super8 article:

WAKE-UP:

The transmit wake-up bit works differently on rev 0 and rev A parts. The difference appears to be when the selected wake-up value—as selected in bit 0 of the UART Mode A (UMA) register—is latched for transmit. It seems that the rev 0 parts have a 9-bit transmit shift register, but only an 8-bit Transmit Holding register, so the transmitted wake-up bit depends on the value of the UMA bit 0 _when_the_byte_is_moved_from_the_holding_register_to_the__shift_register_. The rev A parts seem to have a 9-bit Transmit Holding Register, so the transmitted wake-up bit depends on the value of UMA bit 0 _when_the_byte_is_written_to_the_Transmit__Holding_register_ (UIO). In other words, in the rev A parts you must set UMA bit 0 before writing the character to UIO; in the rev 0 parts you must do likewise, but you must also leave UMA bit 0 set until the Transmit Buffer Empty signal becomes active.

—Brad Rodriguez

Thanks for the note, Brad. You're right, the source for the 6809 assembler is available in the Forth Interest Group Roundtable software libraries on GEnie. And here is the bio paragraph that we didn't get to run on your Super8 article:

Doug Fleenor is a microcomputer hardware & software specialist serving the entertainment industry. He left the security of full time employment in September 1990 to join the ranks of struggling (but happy) independents. Doug's current processor of choice is the Zilog Super8, because "it does everything I need...for now". His extensive work on lighting control multiplex systems earned him the nickname Dr. Mux, which he accepted in "yankee doodle" style: he has been known to wear a doctor's lab coat to industry trade shows, with an oscilloscope probe around his neck. Doug can be reached at Doug Fleenor Design, 396 Corbett Canyon Rd., Arroyo Grande, CA 93420.

Brad Rodriguez is another veteran of the entertainment lighting industry, whose infatuation with Forth and Forth-supporting processors is well known. Brad was introduced to the Zilog Super8 by Doug Fleenor in 1987, and has been enamored of it ever since. Currently Brad is putting the finishing touches on a commercial Super8 Forth compiler, and also on a Super8-based multiprocessor lighting control system. He prefers to be contacted as B.RODRIGUEZ2 on GEnie, but will accept email as bradford@maccs.dcss.mcmaster.ca on the Internet, or paper mail at T-Recursive Technology, 221 King St. E., Suite 32, Hamilton, Ontario L8N 1B5 Canada.

Brad and Doug have collaborated on several Super8-based projects, among them the Teatronics Producer II+, Quantum, Comstar, MD288, and Echelon. Between them, Brad and Doug claim to have used every function and feature of the Zilog Super8.●

Letters to the editor and other readers are welcome. Submit to **The Computer Journal**, *P.O. Box 12, South Plainfield, NJ 07080-0012. Letters may also be electronically submitted via GEnie™ to "TCJ$". Submission implies permission to publish your letter unless otherwise stated. Letters may be edited as necessary.*

Fuzzilogy 101

What, precisely, do you think you're doing? Fuzzy Logic may help.

By Matt Mercaldo

Fuzzy Logic! What is fuzzy logic? The Japanese seem to know. They are using it in all types of commercial goods that once were difficult to use. Now by the use of this technique of hardware and program we can shake when video taping, get in focus when filming, drive our cars easier and with more peace of mind and the list continues to go on.

Fuzzy Logic! What *is* fuzzy logic? As program and hardware scriveners we have to know. Is it just a way to make programming easier? Is it just "The right way of doing things that we've done all along anyhow"? Well, sort of.

Let's first talk about fuzzy logic in terms of what it is and is not. Fuzzy logic is not "crisp" logic. Crisp logic means that a square peg will fit in a square hole and a round peg will fit in a round hole and that is the law of the land. For a lot of applications, Crisp logic is the right way of approaching the problem and has been for years and years. When a specific combination of switches is set (and not set) in some huge evolved automaton of the factory floor, a crisp logic based system will find a rule which matches the combination of the on's and off's. When the correct rule is found, the system will take the winning rule's action. Crisp logic usually has one winner and systems tend to become hierarchical (many levels of if's and then's and else's all nested together in the typical complete and confusing construct).

Fuzzy logic is not fuzzy at all! Actually it is the essence of structure; in an extensible sort of a way. Systems using fuzzy logic tend to make creation of a process control or like event driven system old hat. With the proper fuzzy logic system creation tools, complex process control systems can be constructed and simulated in a very short time. Examples of this technique even exist where a fuzzy system can teach itself by adapting its membership sets to a new environment (Membership sets will be explained shortly). (This sort of thing is critical when the six legged explorer you have spent bajillions of dollars creating needs to relearn how to walk when it reaches the moon with the moon's lower gravity!)

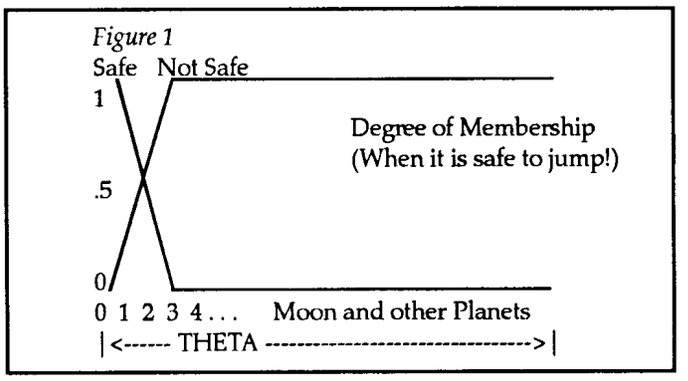
Fuzzy logic systems are not neural networks. Trust me on this one! We do not have time to enter a treatise on Neural nets. Fuzzy logic is closer to a production system (similar to that found in an earlier article of my authorship on stepper motor control) than neural nets.

Let us come up with a simple working definition for Fuzzy Logic. Fuzzy logic is a technique of associating a degree of truth to the relationship between a set of monitored data and a rule's antecedent. Unlike crisp logic, all rules participate in the output result.

"Right!@#%!!!!" you say. Maybe a deeper look into an actual example may help. "Johnny, If I've told you once I've told you a thousand times!!! Some day you are going to break your head open and I'll have to pick up the pieces!! Get off of the swings *only* when the swing has stopped swinging—I mean *completely stopped.*"

Ah, the carefree days of childhood! The mother, in this little remembrance, has laid down the law of the land in quite a crisp fashion. But later the day crawled onward for Mommy, and soon she wished to go home and relax.

Figure 1



"Ok Johnny you can get off of the swing. It is *slow enough* for you to get off now". Interesting! Slow Enough—not quite the crisp answer. If Johnny were driven by a PLC all sorts of interesting things would happen. Good thing Johnny is just a red blooded American boy. There's the jump, and off the swing Johnny flies!

So is Mommy double talking? Well not really, actually she is using fuzzy logic to make a decision as to the law of the land. (Fuzzy Logic in the Practice of Law?..Hum) Let us get really clinical about swings and pendulums. Swings are a lot like pendulums if you really think about it. There is a distance from the center point (Center point being where the law of the land said Johnny could get off the swing the first time the law was spoken forth), and there is a momentum or speed at which the swing is swinging (This must be zero—no swinging—in the crisp law of the land as laid down by the Queen of the land; Mommy). We will call the angle of the swing to the center point THETA, and the momentum or speed of the swing dTHETA. When Johnny is swinging with all of his might (in order to reach the moon or any near planet) his dTHETA is highest

Matthew Mercaldo is employed by a huge firm. With a small group, he develops software tools for field service engineers to do their thing. At 4:30 or 5:00 p.m., when the whistle blows, his thoughts race toward the edge. He dreams of articulated six legged walking beasts, electronic brains that can fend for themselves, and the stuff of "U.S. Robots and Mechanical Men." Someday he dreams of running power out to his garage, and with his wife and a select group of friends, opening his own automoton shop—and thus partially fulfilling his childhood dreams. (Plutonium, Tritium and the like are still not available for public "consumption", but seeing the moons of Jupiter would be spectacular in one's own starcruiser!)

when his THETA is lowest and likewise his dTHETA is lowest as his THETA is highest. Gravity does wonders at keeping little boys feet on the Earth and swing fast and high and not necessarily all at the same time. We will use THETA to help the Queen determine the Fuzzy law of the land, because the law of gravity will be in magical proportion to the THETA and the law of the land.

We will assume THETA to be feet between the center point and the seat of the swing. By careful judgment the Queen (Mommy) has decided that when THETA is between one and two feet from the center it is safe for Johnny to get off of the swing. We could even draw a little graph to illustrate this as you see in *figure 1.*

The careful reader will note that it is very safe to jump when THETA is zero and the degree of membership in Safe is 1. The reader will also note that it is not safe to jump when THETA is three and the degree of membership of Safe is 0. So when Johnny's THETA is 1 foot it is 50% safe to jump. As the speed increases from 1 to 2 THETA, jumping becomes more dangerous, more exhilarating, and altogether more fun; Johnny wants to optimize his exit for a THETA of 1 foot, 11.99 inches. Johnny will still be within the current limits of the law and having the most fun! This my friends is the very essence of Fuzziology!

Let us make a simple fuzzy rule base for the laws of the swinging within and without the limits of the law.

```
IF?
      THETA IS? ZERO
THEN?
      EXIT IS MOST_SAFE
END?

IF?
      THETA IS? LOW
THEN?
      EXIT IS MOST_EXHILARATING
END?

IF?
      THETA IS? HIGH
THEN?
      EXIT IS MOST_DANGEROUS
END?
```

Well, ok. ZERO seems tangible enough, but what about LOW, HIGH, and MOST_SAFE, MOST_EXHILARATING, and MOST_DANGEROUS?... Another graph may illustrate the point. Each fuzzy term such as ZERO, LOW, and HIGH, has a set associated with it. In high and scholarly mathematical circles this means a function with long descriptive symbols and numbers with meaningful descriptions attached. For the Hardware Hack this means a lookup table. However and for whomever the membership set is used, it determines an input's degree of membership in the given class. The output membership set can be either an output function or what is called a Singleton. A singleton is a value that represents the desired non-fuzzied output. We will investigate this more closely a little later. Take a gander at *Figure* 2.

In light of the new set graphs let us look at the rules again and think of how this would work in a system. First the THETA value would be sampled, and held. Each rule antecedent would fire with the new THETA value.

Let us make a little techie decision here: The value will be an eight bit value for THETA. With this bit of coercion, we assume that each membership set has been defined in a 256 byte table. Let us assume that THETA when sampled was what would equivocate 1.4 feet for THETA (somewhere around 7C in our 8 bit hex understanding).

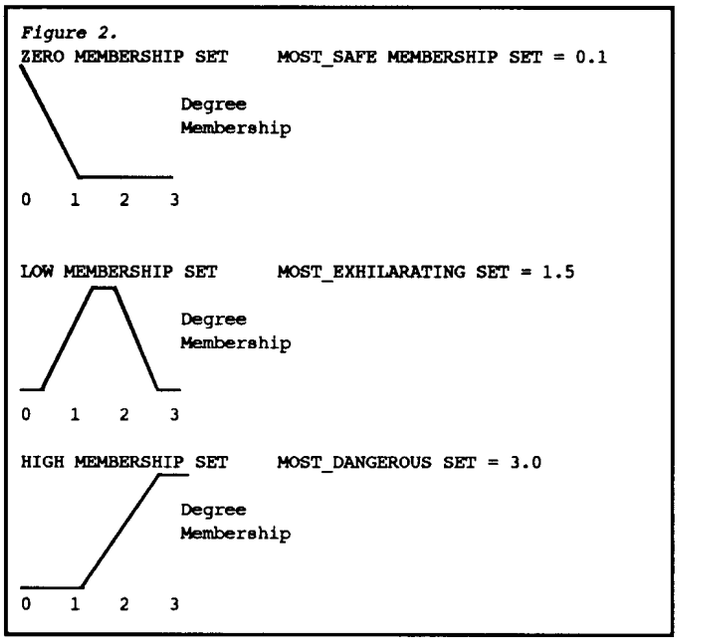
RULE one's antecedent states IF? THETA IS? ZERO. The value for 1.4 is looked up in the membership table for ZERO and found to have a degree of membership of around 10%.

RULE two's antecedent states IF? THETA IS? LOW. The value for 1.4 is looked up in the membership table for LOW and found to have a degree of membership of around 60%.

RULE three's antecedent states IF? THETA IS? HIGH. The value for 1.4 is looked up in the membership table for HIGH and found to have a degree of membership of around 0%.

The percent of membership are remembered for each antecedent for each rule as the rule's Fuzzy Output. Fuzzy Output is a value that indicates a degree of truth for a rule's antecedent. Fuzzy Output is an intermediary value between the portion of the rule which declares a verdict and that portion of the rule which carries

Figure 2.



out the judgment. In fuzzy systems with more than one antecedenal component, the Fuzzy MIN rule is applied to the antecedents subparts. The fuzzy min rule states that the lowest percent of membership of the antecedent's parts will be the Fuzzy Output or degree of truth of the antecedent of the fuzzy rule under scrutiny. The reader can think of the antecedent of a fuzzy rule as a chain. Each antecedenal component is a link in the chain. Just as the strength of the chain is determined by its weakest link, so the degree of truth of a fuzzy rule is determined by the rule's weakest antecedenal component. The above process is called Fuzzification.

After the fuzzy outputs are determined for all of the anticedenal components, they work together to calculate a value for the system output or system outputs. The process of getting a meaningful output is called Defuzzification. A standard way of defuzzifying is by the Center of Gravity defuzzification method. The fuzzified outputs are looked up in the output membership set tables MOST_SAFE, MOST_EXHILARATING, and MOST_DANGEROUS or if Singletons are used, the singleton values for MOST_SAFE, MOST_EXHILARATING and MOST_DANGEROUS are used. Singletons are like a Tone setting on a stereo. A tone knob sets what the tonal quality of the stereo will be. Imagine the notches of HIGH, MEDIUM, and LOW on the tone control as singletons. There are three positions given. Any setting can be described in relationship to one of those settings. In fuzzy rule systems, singletons are used as starting points to determine the output, or in our example, the Tone level of the stereo. Singletons working together with Fuzzy Outputs (used in the center of gravity formula), determine a final setting that will be within a system output's allowable range of outputs. In the tone control example, the system output will be the Tone setting on the tone knob. The tone knob can only go so far to the LOW or

to the HIGH. Also the output value may not necessarily be the same as any singleton value either. Once again, The tone knob may not find itself on LOW or MED or HIGH after all is fuzzily inferred and done.

If output membership sets are used the output membership set takes a degree of truth and use it to look up or find a specific value in a range of values. In the above singleton example illustrated with a stereo's tone control knob, output membership sets give us an equalizer. Each membership set would equivocate a frequency range slider. Assume that instead of three positions on a tone knob we have an equalizer with three sliders adjusting each range of HIGH, MEDIUM and LOW. The LOW slider would not have values in the HIGH range and so on. The antecedents Fuzzy Output value would "move" the slider within the Targeted range of HIGH or MED or LOW as appropriate to the consequent of the fuzzy rule. Note that the consequent of a fuzzy rule is specific to an output and a range. In the above rules we see:

```
THEN? EXIT IS? MOST_SAFE
```

EXIT is the fuzzy output and MOST_SAFE qualifies the output.

Ultimately we would get something very close to the singleton value for that range. Output membership sets are kind of like a fine tuning over Singletons just as an equalizer on a stereo lets the audiophile adjust the stereo's response more resolutely than a tone control knob could. When an output memberships value is ascertained though, it is used the same as if it were a singleton. The Singleton will be used in the below example for ease of explanation.

For each consequent of each rule the fuzzy outputs (degree of truth of the antecedent as captured in the Fuzzy Output variable for each rule) will be combined with the Singleton or output membership set value in the following center of gravity formula.

For each Fuzzy Rule do

$$\frac{FiSumProducts = (Si * Fi) + FiSumProducts}{FiSum = Fi + FiSum}$$

Where Fi is the fuzzy output value of the ith rule, where Si is the Singleton value of the ith rule and where FiSumProducts is used to hold the sum of Si * Fi accumulated over all of the fuzzy rules, and FiSum is used to hold the sum of the fuzzy output values accumulated over all of the fuzzy rules.

In our example the formula would look like the following assuming singletons or output membership values of .1 for MOST_SAFE, 1.5 for MOST_EXHILARATING, and 3 for MOST_DANGEROUS.

$$\frac{(.1)(.1) + (.6)(1.5) + (0)(3)=}{.1 + .6 + 0}$$

$$\frac{.01 + .9 + 0=}{.7}$$

$$\frac{.91=}{.7}$$

```
1.3 = System Output
```

So 1.3 is the system output of EXIT which means that Johnny will have a most exhilarating jump completely within the limits of the law. We can see that ZERO's fuzzy result had an impact on the output, that HIGH had no impact on the output and that MED had the most impact on the output. Membership functions are determined by an intuitive guess initially. The Queen of the land in our swing example knew that when the swing was about two feet from its centerpoint then the jump became dangerous for a boy of Johnny's size and weight. In a similar way an engineer would determine the initial values for the membership set. The membership set would then get "tweaked" into acceptable values. Our fearless Queen Mommy may see Johnny jump from a THETA of three feet, see the excitement of her boy, and see that maybe three feet wasn't so bad and alter the membership sets accordingly to allow safe jumping to include jumping from a THETA of three feet. Yes, this can tend toward the more "black magic" arts of our profession such as analog system designs if attempted without the proper tools and simulation software.

This technique becomes more exciting when doing problems where the output actually drives a motor or some like device in servo applications (The infamous and almost classical inverted pendulum problem). When viewed deeper Fuzzy logic opens new doors to autonomous systems development. Code was actually developed on the F68HC11 New Micros chip; but after long contemplation it is the author's belief that effective fuzzy control will be better done by professional packages that support custom ASC Fuzzy controller chips. If the general *TCJ* readership wishes, I will write an another article around the fuzzy production system internals and tools that I have developed.●

It is a rare enterprise that can assume it will be serving exactly the same market with the same products in ten years time.

—Jesse Werner

The Cyclic Redundancy Check in Forth

By Walter J. Rottenkolber

This study of the Cyclic Redundancy Check (CRC) grew out of a desire to upgrade my faithful modem program, a patchwork of public domain and homespun 8080 assembly code. The thought of tackling 128K of code daunted me and I decided to recast the program, with enhancements, into Forth. Since the source code of the few Forth modem programs I had dated way back and only used the original checksum, this seemed the ideal time to review CRC's in general, to explore the algorithms available to calculate the CRC, and to develop some useful Forth routines.

I searched my library, and fortunately unearthed some articles written in the days when articles routinely included source code (though not often in Forth). T. Ritter's article proved most helpful, though I'd never before considered Pascal to be a write once, read never language.

In transferring data, a change of even one bit can result in great problems if that data is a program. The Cyclic Redundancy Check is designed to detect even minute changes (errors) in a block of data after a transfer, especially burst errors shorter than the CRC polynomial, and errors of three or less bits. The CRC will detect 99.998% of all errors, which is more than 460 times better than the original xmodem checksum. Though it can't correct the error, the CRC can at least sound the alert, and perhaps even initiate a retransmission of the data block.

How Does a CRC Do It?

The CRC works by concatenating all the data into one huge number, and appending a number of zeros to the end equal to one less than the degree of the CRC polynomial. It is then divided by the CRC polynomial using modulo-2 (exclusive-or) arithmetic. The quotient is thrown away and the remainder kept as the CRC value. The CRC polynomial, itself, is a number specially selected to provide (hopefully) unique remainder values.

A polynomial is an algebraic expression of the form

```
ax^n + ax^n-1 + ... + ax^1 + ax^0
```

and it describes the class of algebraic numbers, to which the integers belong. Note that the power (n) begins with zero and goes to n, the degree of the polynomial.

By substituting the radix (number base) for the x, and a number from 0 to x-1 for the 'a', an integer is generated. For example, the decimal number:

```
2*10^3 + 5*10^2 + 7*10^1 + 3*10^0 =
2*1000 + 5*100 + 7*10 + 3*1 =
2000 + 500 + 70 + 3 =  2573
```

Several polynomials are known and used (see *Table 1*) to deal with different size data blocks. The maximum block size a polynomial can accurately handle is given by: (2^n-1)-1. For a CRC of 16 degrees, this works out to (2^15)-1 bits, or 4095 bytes. Larger blocks require larger polynomials, and this is why Z-Modem uses a 32 bit CRC.

The polynomial used by XModem, XModem-1K, and YModem, is the well-known CCITT CRC polynomial: x^16 + x^12 + x^5 + 1. It's a bit hard to visualize, because this is actually a cryptic summary that only notes the non-zero values in a binary number. But if we flesh it out with nulls, we get:

```
1 0001 0000 0010 0001
```

If you kept your eye on the ball, you will notice that a CRC-16 has 17 bits. So how do we handle the division simply, in real time, with a CPU having 16 bit registers? Easier than you think, thanks to the fact that CRC is the remainder, and the quotient is thrown away.

Let's walk through a simplified CRC calculation as using a four bit polynomial to generate a three bit CRC:

```
          100010
        ----------
1001 | 100110000
       1001
        0001
        0000
         0010
         0000
          0100
          0000
           1000
           1001
            0010
            0000
             010 = CRC
```

Note that the dividend is XOR'd with the polynomial only when a data segment has a one in the most significant bit (MSBit), and that this bit in the remainder is always reset to zero in the XOR. Since all we need is the remainder, the MSBit of the accumulating data can be used as a flag to indicate when to XOR, allowing us to eliminate the MSBit of the polynomial from the calculation.

Using the bit shift method with the same 3-bit CRC, 23-bit data, and an abbreviated polynomial we get:

```
          100 110 000
<--- [1] 001 100 00
          001       XOR
          000 100 00

<--- [0] 001 000 0
<--- [0] 010 000
<--- [0] 100 00
<--- [1] 000 0
          001       XOR
          001 0

<--- [0] 010 = CRC
```

Screen 1
```
.\  CRC-16  Load Screen                              WJR14NOV91

ONLY  FORTH ALSO  FORTH DEFINITIONS

DECIMAL

    VARIABLE  CRCVAL

2 13  THRU

INIT-CRCTBL
TADR @ 1500 ASCII F FILL ( test data )
TBIT1
```

Screen 2
```
.\  CRC-16  Bit Shift Method #1 - Low Level       WJR11NOV91
HEX
CODE  (BITCRC16)  ( data-byte old-crc - new-crc )
    H POP   D POP   B PUSH   8 B LXI
    BEGIN
    E A MOV   RLC   A E MOV   L A MOV
    RAL   A L MOV   H A MOV   RAL   A H MOV
    CS   IF   H A MOV   10 XRI   A H MOV
    L A MOV   21 XRI   A L MOV   THEN
    C DCR   0=
    UNTIL   B POP   HPUSH   JMP  C;
DECIMAL

: BIT1-CRC16  ( data-byte - )
   CRCVAL @ (BITCRC16) CRCVAL ! ;
```

Screen 3
```
.\  CRC-16  Bit Shift Method #2 - High Level      WJR11NOV91

HEX
: BIT2-CRC16  ( data-byte - )
    FLIP CRCVAL @ 8 0 DO
    DUP 0< >R 2* OVER 0< IF 1+ THEN
    R> IF 1021 XOR THEN SWAP 2* SWAP
    LOOP  CRCVAL ! DROP ;
DECIMAL
```

Screen 4
```
.\  CRC-16  Byte Shift Method #1 - Hi-Level       WJR07NOV91

HEX
: BYTE1-CRC16  ( data-byte - )
    CRCVAL @  FLIP
    DUP 0FF  AND  U2/ U2/ U2/ U2/  XOR
    DUP 0FF  AND
    DUP  2* 2* 2* 2*  0FF AND  FLIP
    SWAP  2* 2* 2* 2* 2*
    ( data  oldCRC  tmpl1  templ2 )
    XOR  XOR  XOR  CRCVAL ! ;
DECIMAL
```

Screen 5
```
.\  CRC-16  Byte Shift Method #2 - Hi-Level       WJR11NOV91

HEX
: BYTE2-CRC16  ( data-byte - )
    CRCVAL @
    DUP  FF00  AND  U2/ U2/ U2/ U2/  FF00  AND  XOR
    DUP  FF00  AND  DUP  2* 2* 2* 2*
    SWAP  U2/ U2/ U2/   ( data  oldCRC  tmpl1  tmpl2 )
    XOR  SWAP  FLIP  XOR  XOR  CRCVAL ! ;
DECIMAL
```

Screen 6
```
.\  CRC-16  Byte Shift Method #3 - Lo-Level       WJR09NOV91
HEX
CODE   (BYTE3-CRC16)  ( data-byte  old-crc - new-crc )
    D POP   H POP   B PUSH   H PUSH    0 H LXI   3 C MVI
    D A MOV   F0 ANI   RRC RRC RRC RRC   D XRA   A D MOV
    0F ANI   RLC RLC RLC RLC   E XRA   A E MOV   D H MOV
    BEGIN
           A ORA   H A MOV   RAR   A H MOV   L A MOV
           RAR   A L MOV   C DCR   0=
    UNTIL
    E A MOV   H XRA   A H MOV   D A MOV   L XRA
    B POP     C XRA   A L MOV   B POP   HPUSH   JMP   C;
DECIMAL

: BYTE3-CRC16  ( data-byte - )
    CRCVAL @  (BYTE3-CRC16)  CRCVAL ! ;
```

Screen 7
```
.\  CRC-16  Table Lookup Method #1 (1 of 2)       WJR11NOV91

: CELLS  ( n - n*2 )  2* ;

CREATE  CRCTBL  ( - )  256 CELLS ALLOT
\ : CRCTBL ( - )  PAD 200 + ;

HEX
: (CRC16)  ( data-byte  crc-index - crc-template )
    FLIP 8 0 DO
    DUP 0< >R 2* OVER 0< IF 1+ THEN
    R> IF 1021 XOR THEN SWAP 2* SWAP LOOP
    NIP ;
DECIMAL
```

Screen 8
```
.\  CRC-16  Table Lookup Method #1 (2 of 2)       WJR11NOV91

: INIT-CRCTBL  ( - )
    CRCTBL 256 CELLS ERASE
    256 0 DO
    0 I (CRC16) I CELLS CRCTBL + !
    LOOP ;

HEX
: TBL1-CRC16  ( data-byte - )
    CRCVAL @  FLIP DUP 0FF AND  2*
    CRCTBL + @  ROT XOR SWAP
    FF00  AND XOR  CRCVAL ! ;
DECIMAL
```

The data is shifted a bit at a time into the CRC register MSBit first, while at the same time the MSBit of the CRC is shifted into the carry flag [x]. If the carry flag is set, the CRC is XOR'd with the shortened polynomial number, otherwise just the shift is done. The CRC ends up the same. Note that the pattern of the carrys, reading from top down is the same as quotient in the prior example.

The CRC as done in Modem 7 is a three step process. First the CRC register is reset to zero. Next the data block is fed through the CRC generating routine. Finally, on send, null data the size of the CRC data type (16 bits) is fed through. The value remaining in the CRC register is then tacked onto the end of the data block.

On receive, the same procedure is done, except that at the end of the data block, it is the CRC bytes that are fed through. If all goes well, the value in the CRC register ends up as zero.

Although this CRC is a standard, it is not perfect. It is not compatible with hardware generated CRC's because the data is passed to the CRC routine beginning with the MSBit. Normally, serial data is sent least significant bit (LSBit) first, and processed

Table 1	CRC Polynomials
DDCMP, BSC, (EBCDIC)	X^16 + X^15 + X^2 + 1
SDLC, HDLC, CCITT	X^16 + X^12 + X^5 + 1
ATT, Dig.	X^6 + X + 1
BSC (transcode)	X^12 + X^11 + X^3 + X^2 + X + 1
Ethernet/IEEE 802	X^32 + X^26 + X^23 + X^22 + X^16 + X^12 + X^11 + X^10 + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1

by the hardware in that order. Also the use of nulls to begin and end the CRC reduces the ability to detect extraneous leading nulls, or errors in long strings of null data.

For this reason, the SDLC, HDLC, and CCITT synchronous protocols use a variant of the CRC, the Frame Check Sequence (FCS). This begins with the FCS register set to ones. The data is fed in LSBit first using routines otherwise similar to that for the Modem-7 CRC, including the same polynomial. At the end, the FCS register is flushed by entering data composed of ones. The resulting checksum value is then ones-complemented before being transmitted. The receive process is similar except that at the end, the checksum is fed in. The error-free remainder is a non-zero constant, whose value depends not on the data, but on the polynomial used. (Or so my references say. But when I tried it out the error-free remainder was zero, just as with the CRC.)

However, don't panic! Line noise still causes most of my modem transmission problems. This tends to generate ones, not nulls, and to change the block length to boot. Whatever defects the Modem-7 CRC has in theory don't seem to show up in the real world.

A Cornucopia of CRCs

Screens 2-9 show three different methods of generating the Modem-7 CRC's—bit-shift, byte-shift, and table lookup. Both high level Forth and assembly versions are given. Screens 10-12 provide words to test and compare CRC routines.

In the bit-shift and byte-shift assembler routines, the BC Register is saved early on and later restored. Don't change this in a misbegotten attempt to increase speed. In the CP/M version of Forth-83, the BC Register holds the Interpretive Pointer (IP), which refers to the parameter field of the currently executing Forth word. If you trash this register, you trash the system. So assembly routines that use the BC Register must also preserve and restore its contents.

The Bit Shift CRC

The bit-shift CRC method is a scaled up version of the second example given. The assembly version is just a Forth translation of the original Modem-7 assembly code. Data is shifted into the CRC register through the carry-bit. The operating scheme is shift, test, and ?XOR with the polynomial (1021 hex). The high level version is less elegant because there is no ready way to test the carry-bit. So it checks if the MSBit is a one with 0<, since negative numbers have the MSBit set. The method here is to test first, save the flag, shift, and then ?XOR. The data bit also has to be 'shifted' into the CRC indirectly.

The Byte-shift CRC

The byte-shift method is more properly called the bit-block shift method. As the bits are shifted out of the CRC register, the MSBit acts as a trigger to activate the XOR at the other polynomial XOR points. So the bits to the left of a polynomial point behave as a pattern for those bits located to the right of it.

```
[1] 10011101 01010011
 ^       ^       ^     ^ — Polynomial XOR Points
```

We can take this block of bits left of a polynomial point, and XOR the effected bitblock to the right of it. Of course it has to be done for each point, and that is where the algorithm gets tricky.

The following is what happens with BYTE2-CRC16.

```
      10011101 01010011 old CRC
          1001          FF00 AND, SHR*4, FF00 AND
                        XOR
      10010100 01010011

+----  0010100 00000000 FF00 AND, DUP
|                        |
|  +- 0100               SHR*4  mask#1
|  |                     |
|  |  10010100 00000000 <--+
|  |
|  |     10010 100       SHL*3  mask#2
|  +> 0100               XOR
|     01010010 10000000 XOR-template
|
+---- 01010011 10010100 flipped old CRC

      00000001 00010100 XOR
      00000000 01101110 data XOR

      00000001 01111010 new CRC
```

Consider what happens as the CRC shifts to the left and the data byte is fed in. As the leftmost four bits (nibble) exits, the next nibble passes by the XOR point at Bit-12. For every '1' in the first nibble, a corresponding bit in the second nibble is XOR'd. Instead of proceeding a bit at a time, we could accomplish the same changes if we take the first nibble, shift it under the second nibble, and use it as a mask to XOR the bits as a block.

Since four more bits get shifted, we need to repeat this procedure with the second and third nibble, but for the moment we just shift the nibble to form mask#1, and hold it.

Now as the eight bits in the MSB shift out, the XOR point at Bit-5 will effect the five bits to the left of it as well as the first three bits of the data. By shifting the MSB three bits to the left, we get a XOR mask spread over two bytes (mask#2) that reflects the changes that would occur. The two masks are XOR'd to form an XOR template.

Lastly we need to consider what happens to the data as it passes the XOR point at Bit-0. Since every bit in the MSB effects the data, the entire MSB is the mask for the data. Moreover, after the shift, the LSB of the old-CRC must end up as the MSB of the new-CRC. The easiest way to get these two bytes in their proper order is by flipping them. The XOR template is already properly positioned.

So you now have a sandwich of XOR template, flipped old CRC, and data-byte. When these are all XOR'd together you get the new-CRC.

Like a camel, this algorithm looks strange but it works.

The two CRC methods just discussed used Forth words for binary shifts. The shift left (2*) is straight forward, and is accomplished not by using the bitshift opcodes, but by simply adding the number to itself. This is an old assembler trick that does the same thing but is actually faster. The shift right is more complicated as there are two of them. The first (2/) is the arithmetic shift

```
Screen 9
.\  CRC-16  Table Lookup Method #2                    WJR11NOV91

CODE  (TBL2-CRC16)   ( data-byte old-crc crctbl – new-crc )
    H POP   D POP   E A MOV   D E MOV   0 D MVI
    D DAD   D DAD   M E MOV     H INX   M D MOV
    D XRA   A H MOV   E A MOV   D POP     E XRA
    A L MOV   HPUSH   JMP   C;

: TBL2-CRC16   ( data-byte – )
  CRCVAL @  CRCTBL  (TBL2-CRC16)  CRCVAL ! ;
```

```
Screen 10
.\  CRC-16  Test (1 of 3)                            WJR14NOV91
    DEFER  UPDCRC

: CLRCRC   ( – )   0 CRCVAL ! ;

: FINCRC   ( – )   0 UPDCRC 0 UPDCRC ;

: TKERM    ( – )   ['] KERMIT-CRC16 IS UPDCRC ;
: TBIT1    ( – )   ['] BIT1-CRC16   IS UPDCRC ;
: TBIT2    ( – )   ['] BIT2-CRC16   IS UPDCRC ;
: TBYTE1   ( – )   ['] BYTE1-CRC16  IS UPDCRC ;
: TBYTE2   ( – )   ['] BYTE2-CRC16  IS UPDCRC ;
: TBYTE3   ( – )   ['] BYTE3-CRC16  IS UPDCRC ;
: TTBL1    ( – )   ['] TBL1-CRC16   IS UPDCRC ;
: TTBL2    ( – )   ['] TBL2-CRC16   IS UPDCRC ;
```

```
Screen 11
.\  CRC-16  Test (2 of 3)                            WJR02NOV91

    VARIABLE  GENVAL  VARIABLE #DATABYTES   VARIABLE  TADR
        PAD 1000 + TADR !   1024 #DATABYTES !

: CALCRC   ( – )
    TADR @ #DATABYTES @  BOUNDS DO I C@ UPDCRC LOOP ;

: GENCRC   ( – )
    CLRCRC CALCRC FINCRC  CRCVAL @ GENVAL ! ;

: CHKCRC   ( – )
    CLRCRC CALCRC  GENVAL 1+ C@ UPDCRC GENVAL C@ UPDCRC
    CRCVAL @ GENVAL @ OVER
    U. U. ABORT" Check CRC Not Zero " ;
```

```
Screen 12
.\  CRC-16  Test (3 of 3)                            WJR14NOV91

 VARIABLE  #BLKS     10  #BLKS !  VARIABLE DCH   70 DCH !

: CAL0LOOP
    TADR @ #DATABYTES @  BOUNDS DO I C@ DROP LOOP ;

: 0LOOP   ( – )
    CRCVAL OFF #BLKS @ 0 DO CAL0LOOP LOOP  BEEP ;

: TCRC   ( – )
    CRCVAL OFF #BLKS @ 0 DO  CALCRC  LOOP
    CRCVAL @ U.  BEEP ;

: ITB   ( – )  TADR @ 1500 DCH @ FILL ;
```

and is used with signed integers. As the numbers are shifted right it copies the MSBit, which preserves the sign of the number, e.g. 10011110 2/ becomes 11001111, not 01001111. The second (U2/) is the logical shift and is used with unsigned integers. As the bits shift right, the MSBit is set to zero. It's amazing how obscure a bug can be if you mix up these two right shifts.

The Table Look-up CRC

The table lookup method makes use of the fact that the most significant byte merely acts as a trigger for the CRC polynomial, and is then discarded. Therefore, this byte uniquely defines what happens to the least significant byte and the data byte. If we set these latter bytes to zero and calculate a CRC, we form an XOR template that can be used to generate the new CRC.

First you must set up a table of 256 templates calculated on a most significant byte incremented from 0 to 255, and using this byte as an index into the template table.

```
+--------------+  +-----+
| index |   0  |  |  0  | Generate CRC for Table
+--------------+  +-----+
```

To calculate the new-CRC, you use the MSB of the old-CRC as an index into the CRC Table and get the template. Then XOR the LSB of the old-CRC with the MSB of the template, and the data byte with the LSB of the template. Voila! Instant new-CRC.

```
   (index)          LSB       --    old-CRC
    MSB             LSB       --    template
LSB-old CRC    data byte      --    XOR
    MSB             LSB       --    new-CRC
```

CRC How They Run

Screens 10-12 provide words to test the CRCs. I made UPDCRC a deferred word so that the switching from one CRC method to another is easy. TBIT1 switches in BIT1-CRC16 into the test suite, and the remainder of the list does the same for the other CRCs.

To check the accuracy of the CRC, I used GENCRC and CHKCRC. GENCRC acts as a send routine and generates the CRC from data placed at TADR. CHKCRC behaves as a receive routine and calculates a CRC from the data and the check sum passed in GENVAL. The two CRC values calculated are then displayed, along with any error messages. The CRCVAL from CHKCRC has to be zero, or you have a problem.

Time tests are done by TCRC. This does a CRC on several blocks so that a reasonable time elapses for accuracy with hand timing (my Kaypro has no internal clock). I began with #BLKS set at 10 to get a rough estimate, and then increased it to 30 to detect finer differences in performance. Since some time is taken up by the loop itself, 0LOOP defines a null loop whose time is subtracted from the raw CRC times. This allows for more accurate comparisons among the CRC methods.

Table 2 CRC-16 Tests

CRC	Sec/1-KByte	Sec/128-Byte	Bytes/Sec.	Rel. Speed
Bit1	0.29	0.036	3,531	1.45
Bit2	3.50	0.44	293	17.47
Byte1	0.81	0.10	1,264	4.05
Byte2	0.79	0.099	1,296	3.95
Byte3	0.22	0.028	4,655	1.10
Tbl1	0.47	0.059	2,178	2.35
Tbl2	0.20	0.025	5,120	1.00

DCH contains the data used by ITB to fill the block pointed to by TADR. If you want more random data, you can point TADR to the Forth system and do a CRC on parts of it instead.

I performed the time tests of the CRC routines on my 5 MHz. Kaypro II, and summarized them in *Table* 2. Realize, though, that the speed of the routine is not the speed of the file transfer program, but it does give you a starting point for program design.

The table lookup method proved fastest of the group in both the high level and assembler versions. The only disadvantage is

See CRC, page 12

The Internetwork Protocol (IP)

By Wayne Sung

There is a very large and well-connected data network which covers nearly thirty countries and encompasses tens, perhaps hundreds, of thousands of computers. This network is called *The Internet*.

The *Internetwork Protocol*, commonly called IP, is perhaps the most popular protocol in wide-area networking. It also functions quite well in local area networks. It has been used in amateur packet radio networks at speeds of 1200 bps and in supercomputer networks up to hundreds of megabits per second, without modification.

"an IP implementation should be conservative in what it sends, liberal in what it receives."

There are several advantages to using IP rather than another protocol. The most important of these is that IP is not owned by any one company. It was developed by the US Department of Defense and therefore belongs to all taxpayers. This means any company is free to implement IP. It also means that there are several implementations that can be considered reference models against which a new one can be tested.

IP was designed specifically for survivability. Although this was originally meant to insure communications in battle situations, the robustness built into IP makes it particularly able to handle poorly designed networks. For example, there is a rule that an IP implementation should be conservative in what it sends, liberal in what it receives.

This means: do as good a job as possible in making your implementation correct, but try to process incoming packets even if they are slightly malformed. This robustness principle has often been neglected, particularly in local area network situations, because the environment is so sterile compared to what IP was designed for originally.

I have noted earlier that all networks tend to get bigger with time. Starting with a well-built IP means that no software changes will be necessary when the network expands.

I will not attempt to describe everything there is to know about IP here. Entire textbooks have been written for that purpose. In particular, IP is normally used with other protocols, and indeed may have started the whole idea of layered protocols. These other protocols provide functions beyond what IP itself provides, which is primarily addressing.

A minimum IP header is 20 bytes long, of which 11 bytes are involved in addressing. It has been said that IP was not designed by committee, and therefore all the fields in the header are actually necessary. This is certainly true in the minimum header, which is used in by far the majority of situations.

Every IP-capable machine has at least one IP address, which is a 32-bit integer. The way this address is interpreted is what distinguishes IP's wide-area capabilities from other protocols. The most significant bits of the IP address are called the Network Number, and the other bits the Host Number. The exact division point depends on three classes of addresses: A, B, and C.

Class A networks are considered 'big' networks and have eight bits of network number followed by 24 bits of host number. Class B networks are considered 'medium' size and have 16 bits each of network and host number. Class C networks are considered 'small' and have 24 bits of network number and eight bits of host number.

The size designation is based only on the address space available for host addresses, not how large an organization might be. For example, a company that supplies communication services might have switches located throughout the country. There might be hundreds of these switches, making it easier to use one class B network rather than several class C networks. The company itself might not be very large at all.

"Often a company will bring up a network with an arbitrary network number only to find that when they have to connect to the Internet everything has to be renumbered."

All IP network numbers are assigned by the Defense Data Network's Network Information Center, commonly called the NIC. An organization with requirements for IP networking requests one or more network number assignments from the NIC. This registration process is what guarantees IP internetworking, because IP packets are switched from one place to another based on the network addresses.

Often a company will bring up a network with an arbitrary network number only to find that when they have to connect to the Internet everything has to be renumbered. This is why companies

Wayne Sung has been working with microprocessor hardware and software for over ten years. His job involves pushing the limits of networking hardware in attempting to gain as much performance as possible. In the last three years he has developed the Gag-a-matic series of testers, which are meant to see if manufacturers meet their specs.

should consider applying for a network number assignment even if they have no immediate plans to connect to the Internet.

To make the 32-bit number a little easier to handle, it is expressed in terms of four eight-bit numbers. Thus Internet addresses are generally seen in the form W.X.Y.Z, where each of W, X, Y, Z takes on values from 0 to 255. The class distinctions fall on eight-bit boundaries. Class A networks use only W for the network number, class B networks use W.X, and class C networks use W.X.Y.

"Some systems have exploited the ability of Ethernet controllers to be given a user-defined address"

Class A networks have numbers in the range 1 to 126. The values 0 and 127 are reserved for special functions. Class B networks have W values in the range 128 to 191. Class C networks use W values starting at 192. Notice that since class B networks have 16 bits of network address, there can be many more class B addresses than class A addresses. Similarly, since class C has 24 bits of network address, there can be many more class C addresses than class B.

When a network address is assigned, the network portion is filled out for you and the host fields are zeroes. Thus 10.0.0.0 is called Net 10, and 128.1.0.0 is called Net 128.1 (this also shows why a host should not be numbered 0, because this could be confused with the network number).

Certain network numbers are special cases and will not be assigned to anyone. For example, the address 255.255.255.255 is considered a broadcast just as the Ethernet broadcast address is all ones. Net 127 is called the 'loopback' network and is used internally by many IP implementations. There are also host numbers which should not be assigned. The address on any network which is all ones is also considered a broadcast, so a host should not be assigned that address.

In comparison, when you and I each buy an Ethernet controller from a certain company, the controllers will have 48 bit Ethernet addresses which would seem to be usable for routing purposes. The reason they are not is that there is no guarantee that each controller's address falls in a particular range. If we both went to the store on the same day, the addresses might very well be adjacent.

Without a part of the address guaranteed to be distinct across organizations, routers cannot be used. In essence, the Ethernet address is a degenerate case of routing where only the entire address can be guaranteed unique. This also means that the only way to send messages is by the entire Ethernet address, limiting the application to one Ethernet.

Some systems have exploited the ability of Ethernet controllers to be given a user-defined address by programming addresses with more carefully defined network and host portions. Without a central address authority, however, there is still no guarantee that any two companies can connect together. Each may have chosen the same network portion, because many people set up computers using the default supplied by the manufacturer.

Thus 8 bytes of an IP header are the source and destination IP addresses. Every IP packet is considered distinct from every other, and normally routers treat them that way. However, there is a sequence number (16 bits) in each IP packet. The sequence number does not imply that any packet is related to any other, but does supply an additional value which may be used for identification purposes.

It is also possible that two hosts may be sending IP packets to each other, but for different reasons. There is an eight bit number called the protocol field which identifies what higher- layer protocol is being used in this particular packet.

The remaining nine bytes of a minimum header include things such as a checksum (which applies to the header only), a total message length (which applies to the IP message itself and does not include the Ethernet overhead) and other control functions (which are usually not used).

Since IP systems are well-connected, many point-to-point applications are being moved from dedicated lines to using IP as a universal transport. The protocol field of IP makes this quite simple. Note that protocol numbers in IP are also assigned by the NIC, as are most values in header control fields.

I was once shown an Ethernet analyzer which did not do a very good job of capturing packets based on IP addresses (or any protocol-specific address). The manufacturer had some trouble understanding that once a packet passes through a router it loses its original Ethernet address but retains its protocol address.

In my case, seven Ethernet addresses represent several thousand IP addresses, as three of those seven are routers that concentrate traffic from many others. And that's only machines inside North Carolina. Two more Ethernet addresses (the external routers) represent the rest of the whole Internet. I could not possibly pick out an exchange between two machines based on the Ethernet addresses alone.

"Once a packet passes through a router it loses its original Ethernet address but retains its protocol address"

Obviously, there are shortcomings to IP, but there aren't many. In addition, most of these are implementation issues and not bad design in the protocol. The biggest operational problem is with poorly done systems. IP has so many features for robustness that for local-area work it's tempting to bypass many of them. This is particularly true for PCs, which don't have enough cycles to spare anyway. Fortunately, as more and more manufacturers enter the field, the bad versions are retired as obsolete or are fixed.

Even though there are 16,000 class B addresses and millions of class C addresses, networks tend to be sparsely populated. Thus there is a problem with exhausting network numbers even though almost no networks are at a point where all possible addresses in it are assigned. This is the tradeoff for guaranteeing internetworking. Even class C addresses are going fast, as people tend to ask for several at a time.

IP has a broadcast address, and system designers tend not to be very careful about what they broadcast. There have now been established class D and class E addresses which are multicast addresses. Obviously, in Ethernets there is an inherent multicast mechanism, but IP should not be thought of only in local-area terms.

It has been difficult to establish multicast protocols over IP as a result. For example, newsfeeds could be multicast similarly to how radio does it. There you have prearranged times when the news is sent, and interested parties take it. In IP, a separate newsfeed has to be sent to each interested party.

There is a problem with routing information updates in the system of routers. Consider that over 3000 networks are presently connected to the Internet. If you only enumerate each network's number with one more byte to indicate whether it is available, that is over 15,000 bytes of information. At 56 kbps, this takes over two seconds of transmission, which is rather wasteful.

The problem is being handled in two ways. First, line speeds are getting higher, so the transmission time is reduced and the proportion wasted is smaller. Second, update systems are being modified so that nothing is sent unless some change actually happens. This greatly reduces the amount of data to be sent.

A somewhat more serious problem is known as the ARP-cache problem. Recall that in a local area network everybody is available through one port. It then becomes necessary to have a lookup table referencing IP addresses to Ethernet addresses. This is because the Ethernet system switches using Ethernet addresses, even if the application uses IP addresses.

The mechanism that provides this cross reference is called ARP, the Address Resolution Protocol. The initial exchange of ARP is a broadcast, asking the question 'does anybody on this network have the IP address so-and-so?' If that machine does exist, it will send a reply directly to the asking machine.

Some designers thought it desirable to retain knowledge of this exchange even if the original requirement is finished. They reasoned that on a local area network, machines access each other quite often and there really is no reason to ask every time, especially since broadcasts should be used sparingly.

Unfortunately, the design drops address only if it has not been used for any reason during a fairly long period of time. If the destination disappeared for any reason but the source machine kept trying to reach that destination, the entry would never go away.

This is particularly painful if the destination is the external router, and there is a backup router available. Due to the ARP-cache design, the backup is essentially useless because the source keeps trying the original router which has failed for some reason.

The solution seems to be to drop an entry after so much time whether or not it has been used recently. This forces a new exchange every so often, so even if an address changes, there will only be a short outage.

All in all, IP is a very capable protocol and is still gaining popularity. Extensions to the protocol are still being done and generally do not require everyone to upgrade, showing that the original design was well done.

The Internet in the United States is operated under the auspices of the National Science Foundation. The 'backbone', that is, the set of lines connecting major areas in the country to one another, has been operating at T1 speeds for some years now and is actively moving to T3 speeds. Next time we'll look at how exactly T1 lines work.●

CRC, from page 9

the 512 bytes that the table requires. The table can be either hard wired into a saved program, or calculated into dynamic memory. Setting up the table is no problem as even the slow bit-shift routine calculates the table in 1.5 seconds, and that needs to be done only once at the beginning of the program.

The good speed of the byte-shift methods surprised me. This algorithm is as tortuous as any I've ever seen. A week of doodling nulls and ones and scratching my head passed before I could get it sorted out and debugged. Even so, the high level versions could hold their own in a medium speed modem program. After finishing BYTE1-CRC16, I derived BYTE2-CRC16 to save two shifts per CRC update, and increased the speed by 2.5%.

The bit-shift CRC routines finished last though they are closest in operation to their hardware counterparts. The high level version (BIT2-CRC16) amazed me by its significant lack of speed. Simplicity is its only virtue.

Conclusion

I hope that this selection of CRC routines will expand the options in designing your next homebrew project. I doubt that this exhausts all the algorithms as I got a whiff of one or two more in my explorations, but following the trail is difficult in a non-University setting. Meanwhile, I follow Charles Moore's dictum—share the code.●

References

1. Terry Ritter," The Great CRC Mystery", Dr. Dobb's Journal, Feb. 1986.
2. Jerry LeVan," A Fast CRC", Byte, Nov. 1987.
3. Donald Krantz," Christensen Protocols in C", Dr. Dobb's Journal, June 1985.
4. Ethan Winer & Jay Munro," Download Utilities The Easy Way With PC-Access", PC Magazine, Apr. 10, 1990, p. 293-312.

Z-System Corner

Type-3 and Type-4 Programs
Some New Applications of Type-4 Programs

By Jay Sage

Hors d'Oeuvres

Before starting on the main material for this column, I have two subjects to discuss very briefly. First, for those waiting for the CPU280 to become available, your wait will almost surely be over by the time you are reading this. Ralph Becker just became the first American with a working CPU280, and he knows where to get all the parts. Between the two of us, we will be offering CPU280 kits.

Second, I have just started using 4DOS, a command processor to replace those that come with MS-DOS and DR DOS. I am extremely impressed. [The name of the principal author of 4DOS, by the way, is Rex Conn, uncannily similar to that of the principal author of ZCPR, Rick Conn!] Like PCED, which I have used for several years and about which I have already written in *TCJ*, 4DOS offers almost (but still not quite) everything that Z-System does. I will be talking about it in the future, and it has already been added to the Sage Microsystems East product line (the retail price is $69, but SME will have an introductory price of $60). The manual alone is worth the money. It is written in a Z-System style and has lots of good, basic information about MS-DOS-style computers.

I got started with 4DOS because, for reasons that are still not clear, I have not been able to get PCED to work in a DESQview window under DR DOS 6. It works outside DESQview with DR DOS 6 and inside a DV window with MS-DOS 3.3 (and, I am told, MS-DOS 5). But the combination of DR DOS 6 and DESQview just won't go. The problem will probably be solved, but I don't have the solution yet. There are still some things that PCED offers that 4DOS does not, but the reverse is also true.

Some time we should also talk about ZSIM12, a true CP/M emulator for IBM-PC-compatible computers. ZSIM provides so complete an emulation—far and away more complete than 22NICE and Z80MU—that NZCOM runs perfectly under it. PCED, 4DOS, and ZSIM give us three ways to have Z-System-like functionality on a PC platform.

The Main Topic

In my last column, which recounted the historical development of ZCPR, I mentioned that in version 3.3 I had introduced a revolutionary concept for CP/M—programs that would be loaded and run at a fixed address other than the standard 100H. These programs were called, for reasons that will be explained a little later, type-3 programs.

In ZCPR version 3.4, with Joe Wright leading the way, he and I introduced a further extension of this concept, the type-4 program. Joe envisioned this kind of program as loading itself automatically as high in memory as it could possibly run given the current system configuration at load time. However, by moving the special loader code out of the command processor (CCP) and into the program header, I not only saved code space in the CCP but also made the type-4 loader concept more general.

Until recently, however, no one had developed any type-4 loaders other than the one, released with ZCPR34, that performed the task Joe wanted to achieve. This column will present several new examples that I hope will promote some creative thought on the part of readers.

The Type-3 Program

Programs written specifically for ZCPR version 3.0 began with special header code having the following form:

```
org   100h              ; Programs always start at 100H
jp          start       ; Jump past the header
db          'Z3ENV'     ; ID to identify a Z program
db          progtype    ; Type of Z program
dw          envaddr     ; Address of ENV module
start:                  ; remainder of program code
```

The program begins with a jump instruction to make execution skip over the header material. Right after that comes an ASCII string, 'Z3ENV', which can be used by other programs (including the command processor) to determine that this is a Z-specific program.

Next comes a byte that contains a program-type value. A value of 1 (binary, not ASCII) indicates that the program references an external environment module (ENV), which is a virtually universal component of a ZCPR3 system. This module contains information about how the ZCPR3 system is configured, such as module addresses and sizes.

Jay Sage has been an avid ZCPR proponent since the very first version appeared. He is best known as the author of the latest versions 3.3 and 3.4 of the ZCPR command processor, his ARUNZ alias processor and ZFILER, a "point-and-shoot" shell.

When Echelon announced its plan to set up a network of remote access computer systems to support ZCPR3, Jay volunteered immediately. He has been running Z-Node #3 for more than six years and can be reached there electronically at 617-965-7259 (MABOS on PC Pursuit, pw=DDT). Jay is the Z-System sysop on the GEnie CP/M Roundtable and can be contacted as JAY.SAGE via GEnie mail, or chatted with live at the Wednesday real-time conferences (10 p.m. Eastern time). He can also be reached by voice at 617-965-3552. A good time to find him at home is between 11 p.m. and midnight but please do not call Fridays or Saturdays. Jay's mailing address is 1435 Centre Street, Newton Centre MA 02159.

In real life, Jay is a physicist at MIT, where he tries to invent devices and circuits that use analog computation to solve problems in signal, image and information processing. His recent interests include artificial neural networks and superconducting electronics. He can be reached at work via Internet as SAGE@LL.MIT.EDU.

Listing 1

```
com:                    ; Process transient command

      ld    hl,tpa      ; Set default execution/load address
      ld    a,3         ; Dynamically load type-3 and above ENVs
      call  mload       ; Load memory with file specified
                        ; in cmd line
; Next two lines added by Bruce Morgen
      jr    nc,comnrml  ; If not special Type 4, carry is clear
      ld    (execadr),hl ; Revise load address, special Type 4 b/m
comnrml:
      ....              ; Some code omitted here

; Copy command tail into TBUFF

callprog1:
tailsv      equ   $+1   ; Pointer for in-the-code modification
      ld    hl,0        ; Address of first character of
                        ; command tail
      ld    de,tbuff    ; Point to TBUFF
      push  de          ; Save pointer
      ld    bc,7f00h    ; C=0 (byte counter) and B=7F (max bytes)
tail:                   ; Loop to copy tail
      ....              ; Some code omitted here

; Run loaded transient program

      call  defltdma    ; Set DMA to 0080h standard value

; Perform automatic installation of Z3 programs (unless
; type-2 environment)

      ld    hl,(execadr) ; Get current execution address
      call  z3chk       ; See if file is a Z3 program
      ld    de,z3env    ; Get actual ENV address
      jr    nz,execute  ; Branch if not Z3

      cp    2           ; If type-2 (internal) environment
      jr    z,execute   ; ..do not perform installation

; Install types 0, 1, 3 and 4 (I don't think type-0 exists)

      inc   hl          ; Advance to place to put Z3ENV address
      ld    (hl),e      ; Put in low byte of environment address
      inc   hl
      ld    (hl),d      ; Put in high byte of environment address

; Test for RST 0 as first byte and change to JP if so

      ld    hl,(execadr) ; Point to first byte of program code
      ld    a,(hl)
      cp    0c7h        ; Test for RST 0
      jr    nz,execute
      ld    (hl),0c3h   ; Replace with JP instruction

; Execution of the program occurs here by calling it as
; a subroutine

execute:
      ex    de,hl       ; Pass Z3ENV address to program in HL

execadr     equ   $+1   ; Pointer for in-line code modification
      call  tpa         ; Call transient (or resident)

; Return from execution
```

Listing 2

```
mload:
      ld    (envtype),a ; Set up in-the-code modification below
      ld    (execadr),hl ; Set up execution/load address
      call  defltdma    ; Set DMA address to 80H for file searches

      ....              ; Code for searching path omitted
      ....              ; Code for extended command
      ....              ; processing omitted

      call  opencmd     ; Open file for input
      ....              ; Some code omitted here
      call  readcmd     ; Read first record into default
                        ; DMA address
      jr    nz,mload5   ; Branch if zero-length file
      ld    (cmdfcb+32),a ; Set file current record back to zero
      ld    hl,tbuff    ; Pointer to start of code
      call  z3chk
      ld    hl,(execadr) ; Get initial loading address
      jr    nz,mload4   ; If not Z3 file, branch

; The following test is modified by earlier code. For normal
; COM file loading, a 3 is inserted for the minimum
; environment type for dynamic load address determination.
; For the GET command, where the user-specified address
; should be used, a value of 0FFH is put in here so the
; carry flag will always be set.

envtype     equ   $+1   ; Pointer for in-the-code modification
      cp    3           ; See if no higher than a type-2
                        ; environment
      jr    c,mload4    ; If lower than type 3 (or 255), branch
      ld    hl,(tbuff+11)     ; Load address in case type-3
      jr    z,mload3b   ; Type 3 command, do that

; As not 1, 2 or 3, assume Type 4 command

      ....              ; Some code omitted

; Type 4 header does its own calculations

      ld    hl,cmdfcb+32      ; Make HL point to record count byte
      ld    (hl),2      ; Set record count to 2 (had
                        ; been reset to 0)
      call  readcmd     ; Record 2 into tbuff
      jr    nz,mload5   ; File too short
      ld    (hl),a      ; A=0 from READCMD into record count
      ld    hl,(tbuff+11)     ; Size info from code section
      call  readcmd     ; Record 0 into tbuff again
      ld    a,fullget   ; We need this flag
      ld    b,h
      ld    c,l         ; Get size info into BC
      ld    de,entry    ; Beginning of CCP
      ld    hl,z3env    ; Pass EnvDisc address ?
      call  tbuff+9     ; Call Type 4 loader
      call  readcmd     ; Read record 1 to tbuff (point to
                        ; record 2)

mload3b:
      ld    (execadr),hl      ; Set new execution/load address
                        ; -- returned by type 4 loader!

; Load the file, making sure neither CPR nor protected
; memory is overwritten

mload4:
      ....              ; Loading code omitted
      call  closcmd     ; Close loaded file

; In case a program would like to find out in what
; directory the command processor found the program,
; temporary DU is stored in bytes 13 (user) and 14 (drive)
; in the command FCB.

tempusr     equ   $+1   ; Pointers for in-the-code modification
tempdr      equ   $+2
      ld    hl,0
      ld    (cmdfcb+13),hl

logcurrent:             ; Return to original logged directory
      ld    de,(curusr)
      jp    logde
```

A PROGTYPE value of 2 (though I don't believe I ever actually saw it used) would indicate that the ENV data block was included (as an extended header) within the program code itself. This was a vestige from ZCPR2, with which Z programs had to have the environment information configured into the program.

With ZCPR33 I introduced programs with a PROGTYPE value of 3 and a slightly extended header. Its form became the following:

```
org   entry             ; Programs start at ANY ADDRESS
jp          start             ; Jump past the header
db          'Z3ENV'           ; ID to identify a Z program
db          3                       ; Type of Z program
dw          envaddr           ; Address of ENV module
dw          entry             ; Load address for program
start:                        ; remainder of program code
```

There is an extra word (i.e., two bytes) following the program-type byte. This word contains the address to which the program was assembled/linked and to which the ZCPR33 command processor was to load and execute the program.

As a quick aside, there was one thing that I didn't think of when I created this kind of program. If one tried to run it under either CP/M or ZCPR version 3.0, the command processor would not know about type-3 programs. It would load the type-3 program to the standard 100H address and then start to run it. The "JP START" instruction would still go to the place where it would have gone had the program been loaded to the specified ENTRY address. Since the code was not in fact loaded there, very strange—and almost always unpleasant—behavior would result.

Others later developed a fix for this. The "JP START" was replaced by "JR START", a relative jump. This would work no matter where the program was loaded. Special lead-in code at the START address would determine the actual load address and compare it to the proper type-3 load address stored in the header. If the addresses did not match, an error message would be displayed, and execution would terminate before any damage was done. This entire piece of protection code must, of course, use only location-independent instructions.

The Type-4 Program

The loading of a type-3 program presents no special challenges to the command processor. It simply copies the contents of the program's COM file into memory starting at the specified address rather than the standard address of 100H. A type-4 program, on the other hand, is going to be loaded and run at different addresses at different times; the COM file cannot, therefore, contain the actual code as it is to be run.

The problem of storing code in a form that permits it to run at arbitrary addresses had been solved long ago using what is called a PRL file, where PRL stands for "Page ReLocatable". This file comprises two major parts. The first part is the actual code assembled for a starting address of 100H. This is followed by what is called a bitmap. The bitmap tells which bytes in the 100H-address code have to be adjusted in order for the code to run at another address. Each byte in the bitmap has 8 bits and can cover 8 bytes of the program code. Thus, the bitmap is one- eighth the length of the code.

A PRL file has one other component, a one-page (= two records = 256 bytes) header in front of the two parts I described above. This header is almost completely empty (i.e., filled with zeros). The only data it contains is in bytes 1 and 2 (where the bytes are numbered starting with 0). There it stores the length of the code portion of the file. This value allows whatever routine carries out the relocation to determine where the code ends and the bitmap begins.

Another quick aside. The first byte, byte 0, is filled with a "RST 0" instruction, a one-byte opcode that performs a call to address 0000H. This initiates a warmboot. Thus, if an attempt is made to execute a type-4 program under any version of the command processor other than ZCPR34 or later, the program will simply reboot. This is not as nice as displaying an error message, but it is a lot better than trying to execute a raw type-4 file! To allow this same protection to be used with type-3 programs, some extra code was added to ZCPR34 to replace a leading "RST 0" instruction with a "JP" opcode at load time.

Now back to the matter of how ZCPR34 handles type-4 programs. ZCPR34 needs to accomplish two tasks in connection with loading a type-4 program. First, it has to determine the address to which the code should be loaded. Second, it has to perform the address relocation to permit it to run at that address. To do this, it makes use of code in the CCP together with code placed into the available space in the PRL header. How it does this is really quite tricky, and I doubt that anyone other than Joe Wright could have come up with the code to do it!

Since we don't want the type-4 program to interfere with any user code currently residing in memory, we have a problem figuring out where to execute the loader code. Had it been placed entirely in the command processor, there would have been no problem, but a lot of valuable code space would have been taken up.

Listing 3

```
; Program:  TYP4LDR.Z80
; Authors:  Joe Wright, Bridger Mitchell, Jay Sage
; Contributors: Bruce Morgen

;=========================================================
; Record 0 Routine  - Load Address Calculator

      org   100h          ; ORG HEX file at 100h

      .phase tbuff        ; Program actually runs at 80h

rec0: rst   0             ; Anything but ZCPR34 will warmboot
length:     ds    2       ; Length of the code module (SPR or PRL)
      db    'Z3ENV'       ; Z program ID
      db    4             ; A Type 4 program

start:                    ; CCP calls the code at this point

; stack and register manipulations
; calculation of room needed for program, including bitmap
; adjustment for fullget status
; choose larger of load size and memory size as space required
; calculate load address as if RSX defines top of TPA
; calculate load address as if CCP defines top of TPA
; keep the lower of those two values
; optionally force page alignment
; set up some data on stack for second part of loader
; return to CCP with load address in HL

;=========================================================
; Record 1 Routine  -  Code Relocator

      .phase tbuff        ; This code also runs at 80H

; stack and register manipulations
; make sure header of loaded code has type 4 set
; perform relocation of the code
; return to CCP

      .dephase            ; Good Housekeeping

      end

; End of TYP4LDR.Z80 Relocator
```

Listing 4

```
; Now calculate possible load address assuming CCP defines top of TPA

      pop   hl          ; CCP address ;(+4)
      call  adjaddr     ; Calculate address and keep lowest

; HERE IS THE NEW CODE!

; Finally, calculate the possible load address using the value
; specified in the program header.  If this value is lower than
; the highest possible address, then it will be used.  Otherwise,
; the highest possible value will be used.

      ld    de,(ldaddr) ; Get specified load address
      xor   a           ; Make sure carry flag is clear
      call  adjadr1     ; Keep lower address

; Get ready to return to CCP

loadaddr equ      $+1

      ld    de,-1       ; Get final value of load address

       if   align
      ld    e,a         ; Force page alignment (A=0)
       endif

      pop   hl          ; Get file size ;(+2)
      push  de          ; Pass load address to record-1 routine
                        ;(+4)
      push  de          ; Hold a copy on the stack for now
                        ;(+6)

      ld    hl,tbuff    ; Set for for CCP to call second record
      ex    (sp),hl     ; Load address to HL, tbuff to top of stack

       if   test
      rst   38h         ; DDT breakpoint
       endif

load: jp    0           ; Return to CCP (hot-patched above)
                        ; Command processor loads record 1 to
                        ; ..tbuff and records 2+ to load
                        ; ..address, then 'returns' to tbuff
                        ; ..to finish loading

;---------------------------------------------------------

; This subroutine takes an address for the top of TPA in HL and
; the amount of memory required in DE and computes the proper
; load address.  It then compares it to the value stored at
; LOADADDR and replaces the value there if the new value is lower.

adjaddr:
      xor   a           ; Clear carry flag
      sbc   hl,de       ; Compute new possible load address
      ex    de,hl       ; ..and put it in DE
adjadr1:                ; THIS IS A NEW LABEL!
      ld    hl,(loadaddr) ; Get previously computed load address
      sbc   hl,de       ; Compare HL and DE
      ret   c           ; HL is lower, nothing to do
      ld    (loadaddr),de     ; New lower address
      ret
```

There is one area in memory that is available as a scratch buffer: the second half of the base page from 0080H to 00FFH. This area is used to store the command tail when a program is run, and it is the default buffer used by many programs for transferring data to and from disk. We will now describe in very general terms how the ZCPR34 command processor uses this space.

It begins by loading the first record of the header (record 0) into the buffer. After placing some important information into registers in the CPU (the microprocessor Central Processing Unit), the CCP calls the code there (but not at the very beginning, as we will see later). This code computes the address to which the program should be loaded and puts it into the HL register pair. It then pushes the address of the default buffer onto the stack and jumps back to the command processor.

The command processor now loads the second record of the PRL header file (record 1) into the default buffer and then continues with its standard loading code, just as if it were dealing with a type-1 program (in which case HL would have been set to 100H) or a type-3 program (in which case HL would have been set to the load address stored in the type-3 header). As a result, all of the file from record 2 to the end, including both the code and the bitmap, is loaded into memory at the address specified by the first PRL header routine.

Once the CCP loader subroutine (MLOAD) has completed its work, it executes a RET instruction. This would normally return to the CCP code after the call to MLOAD. At that point the CCP transfers execution to the beginning of the program that was just loaded. In the case of a type-4 program, however, the stack has had the address of the default buffer, 80H, pushed onto it, and so execution resumes instead with the second routine from the PRL header. This code carries out the relocation of the program code so that it can be executed at the address to which it has been loaded. When that operation is finished, a RET instruction finally returns control to the CCP after the "CALL MLOAD" instruction. The type-4 program is now executed.

For those who may be interested, I have included some fragments from the ZCPR34 source code in Listings 1 and 2. Listing 1 contains the code that ZCPR34 calls to load transient programs. It starts with a call to MLOAD after setting the HL register pair to the default load address of 100H.

MLOAD is an incredibly hard-working subroutine, excerpts of which are shown in Listing 2. If you look at the code, you will see how it loads the first record into the default buffer and then systematically determines what type program it is. It first identifies standard CP/M programs and branches to the file-loading code with HL set to 100H. Next it identifies Z programs of types less than 3 and again branches to the code with HL set to 100H.

Then, if the program is type 3, it fetches the load address from the header at an offset of 11 bytes, puts that value into HL, and proceeds with the loading code. If things have not been taken care of by now, the code assumes that it has a type-4 program. If you look at the code carefully, you will see that there is an additional complexity that I have not described (and will not). Suffice it to say that it requires a good bit of juggling of the records loaded into the default buffer.

You can see where the ZCPR34 code makes a call to TBUFF+9. This starts the execution of the code in the first record of the type-4 header. On return from that code, the second record is read in, and the load address in HL is stored as the execution address. The rest of the program file is loaded just as other programs are (but starting with record 2 instead of record 0), and then the code jumps to the LOGDE routine. When that routine is finished, its RET instruction is what starts execution of the second record of the type-4 loader.

The Standard Type-4 Header

The latest version of the standard type-4 header used in all type-4 programs distributed to date is available in the file T4LDR11.Z80. Listing 3 contains a pseudo-code version of the source code.

The first record begins with the standard ID header for a Z program. That is followed by code to calculate the proper load address taking into account a considerable number of different

possibilities (for example, various ways to determine the space needed by the program and the top of available memory). All this functionality has to be accomplished in only 119 bytes. Joe Wright and others who helped him (including Bruce Morgen) have had to be very careful and clever in the coding.

All I will say additionally about the second record is that it uses some clever relocation routines developed by Bridger Mitchell. I have not looked to be sure, but I believe these routines were described in one of Bridger's columns here in TCJ a while back.

A Type-3-Like Type-4 Loader

For my first example of a new type-4 loader, I am going to show you one that makes a type-4 program act like a type-3 program, but with the fixed load address determined not at assembly time but simply by patching in the address. In this way, it would be possible to distribute a program that acts like a type-3 program but where the user could choose his/her own load address without having to know anything about assembling and linking code.

Why would one ever want to use a type-3 program instead of the standard type-4 that automatically loads itself as high as possible? Well, I can think of one situation for sure. Suppose one wants to use the JUMP command to rerun it? It is true that many type-4 programs display a signon message that reports the actual load address, and this address could then be used with the JUMP command. However, this requires that one pay careful attention to the signon message, and it rules out using a JUMP command in an alias, where the address has to be fixed and known in advance.

I will be distributing this loader in the file T4T3LDR1.Z80. Its header is modified to the following form:

```
rst         0                           ; Warmboot if
executed
dw                  length              ; Code length
db                  'Z3ENV'             ; ID to identify a
Z program
db                  4                         ; Type-4
program
dw                  envaddr             ; Address of ENV
module
; Entry point of loader code
jr                  start               ; Jump past extra
header
db                  'ADDR='             ; String to
identify patch point
dw                  loadaddr            ; Patch fixed load
address here
start:                                  ; Resume with
T4LDR11.Z80 code
```

There is a place for the user to patch in a fixed load address to be used. This location is made easy to find by the ASCII string in front of it.

Listing 5

```
; Program:  T4PLDR.Z80
; Authors:  Jay Sage
; Date:     December 17, 1988

; This type-4 loader prompts the user for the desired load address. A
; single digit between '0' and '9' is entered.If '0', the program is
; loaded to 100H  For any other digit, the load address is that digit
; followed by "000h". Thus '4' causes the program to load at 4000h.
; There is no checking to make sure that the specified address is
; safe; thus, the system can be crashed if too high a load address
; is specified.

vers  equ   10

tbuff equ   80h          ; Always executes from default buffer

; Macros for printing space free have been omitted here, and their
; invocations later in the code have been stripped out.  The
; entire second record has also been omitted, since it is the same
; as the standard one in T4LDR11.Z80.

;=============================================================
; Record 0 Routine  --  Load Address Calculator

; On entry, we have:
;
;     HL  = Z3ENV, environment descriptor address
;     DE  = ENTRY, address of beginning of CCP
;     BC  = PROGSIZE, program size from program header
;           in record 2
;      A  = FULLGET flag

      org   100h         ; ORG HEX file at 100h

      .phase tbuff       ; Program actually runs at 80h

rec0: rst   0            ; Only a ZCPR34-compatible CCP
                         ; ..can execute this file
length:     ds    2            ; Length of the code

      db    'Z3ENV'
      db    4            ; A Type 4 program

start: ex   (sp),hl      ; Get return address from stack
                         ; ..and put Z3ENV on the stack
      ld    (load+1),hl ; Set in-line jump to the return address
      ld    hl,(length) ; Length of this PRL/SPR code
      push  hl          ; Save on stack for sector 1 routine ;(+2)

; Now we need to get the load address.

      ld    (stack),sp  ; Save stack pointer
      ld    sp,stack    ; Set up local stack pointer
      ld    c,9         ; BDOS string output routine
      ld    de,message
      call  5
      ld    c,1         ; BDOS conin function
      call  5
      ld    sp,(stack)  ; Restore original stack pointer

      ld    hl,100h           ; Default load address
      cp    '1'         ; See if less than '1'
      jr    c,resume    ; If so, use default address
      cp    '9'+1       ; See if between '1' and '9'
      jr    nc,resume   ; If not, use default address
      sub   '0'         ; Convert to binary
      add   a,a         ; Multiply by 16 (10H)
      add   a,a
      add   a,a
      add   a,a
      ld    h,a         ; Use as load address

resume:
      push  hl          ; Pass load address to record-1 routine
                        ; (+4)
      push  hl          ; Fill another stack entry ;(+6)
      ld    hl,tbuff    ; Set for for CCP to call second record
      ex    (sp),hl     ; Make ZCPR34 go to tbuff, and return
                        ; ..with load address in HL

load: jp    0           ; Return to CCP (hot-patched above)
                        ; Command processor loads record 1 to tbuff
                        ; ..and records 2+ to load address, then
                        ; ..'returns' to tbuff

message:
      db    'Digit (0-9) for load address: $'

      ds    2*13        ; Space for local stack
stack: ds   2           ; Place to keep original stack pointer

      .dephase
```

The rest of the code is the same as that in T4LDR11.Z80 except that at the very end, after the highest possible load address has been calculated, it is compared to the fixed load address specified. If the fixed address is lower than the highest allowed, then its value is put into HL. If the fixed address specified is too high, then the highest possible value is used instead. Very little additional code is required because the comparison code is already available in an existing subroutine. The new code is shown in Listing 4.

Installing a New Type-4 Loader

Before going on to the second example of a new type-4 loader, I would like to describe how these new loaders can be installed into an existing type-4 program so that you can experiment. One of the nice things about the way type-4 programs are implemented in ZCPR34 is that you do not need source code for the program. A new loader can be installed at any time. You need only the loader code in the form of a HEX file, which you can either generate yourself from the loader source code or get from someone else who has done the assembly.

To make the HEX file, you assemble the loader source with the appropriate assembler switch on the command line. For example, with ZMAC or the SLR Z80ASM assemblers the command lines would be

```
ZMAC T4T3LDR1/H
Z80ASM T4T3LDR1/H
```

Once you have the HEX file (e.g., T4T3LDR1.HEX), you install it onto a type-4 program (let's call it PROG.COM) using MLOAD or MYLOAD, public-domain loader utilities available through the ZSUS service or from Z-Nodes and, in the case of MLOAD, many RCPM systems. Here is the command line to make a new version of PROG.COM with the new loader:

```
MLOAD PROGNEW.COM=PROG.COM,T4T3LDR1.HEX
```

A More Elegant Type-3-Like Loader

Well, now we have a type-4 program that can be patched easily to load the program to any address we like. How about making life even nicer by having the loader ask us — at the time of loading—where we would like the code loaded? We can then easily load the program to different addresses each time we run it without the nuisance of having to patch it. [Here is an exercise for the reader: why can't we use the GET/POKE/GO or GET/POKE/JUMP techniques to hot patch the address into the type-4 program (or do any hot patching at all)? Hint: will the GO or JUMP commands work right?]

This second new loader contains none of the code for calculating the load address that T4LDR11 has. Instead, it puts up a prompt and waits for the user to press a digit key between '1' and '9'. If an illegal key is pressed, the code is loaded to address 100H. For legal digits, the value is multiplied by 1000H, and that value

Listing 6

```
; Program:  T4WBLDR.Z80
; Authors:  Jay Sage
; Date:     December 17, 1988

; This type-4 loader is similar to T4T3LDR.Z80 but with two
; important differences. First, it loads the code to the address
; specified in the header without any checking, and, second, it
; does not execute the code. It uses Bruce Morgen's addition to
; ZCPR34E that allows the execution address to be changed to the
; value returned in HL by the second record of loader code when
; the carry flag is set.  This loader sets that address to
; 0000H to force a warm boot.

vers  equ   10
tbuff equ   80h         ; Always executes from the default buffer
bdos  equ   05h         ; BDOS entry point

; Macro definitions that we included here and their invocations
; later in the code have been omitted. Some comments that
; duplicate material in other listing have also been stripped out.

;=============================================================
; Record 0 Routine  --  Load Address Calculator

      org   100h        ; ORG HEX file at 100h

      .phase tbuff      ; Program actually runs at 80h

rec0: rst   0           ; Only a ZCPR34-compatible CCP
                        ; ..can execute this file
length: ds  2           ; Length of the code

      db    'Z3ENV'
      db    4           ; A Type 4 program

      jr    start       ; Jump past load address

      db    'ADDR='     ; ASCII marker
ldaddr: dw  1000H       ; This value should be patched to the
                        ; ..desired load address
message:                ; Patch the message, too
      db    'Code loaded to address 1000H$'

start: ex   (sp),hl     ; Get return address from stack
                        ; ..and put Z3ENV on the stack
      ld    (load+1),hl ; Set in-line jump to the return address
      ld    hl,(length) ; Length of this PRL/SPR code
      push  hl          ; Save on stack for sector 1 routine ;(+2)

; Report load address to user.

      ld    (stack),sp  ; Save stack pointer
      ld    sp,stack    ; Set up local stack pointer
      ld    c,9         ; BDOS string output routine
      ld    de,message
      call  bdos
      ld    sp,(stack)  ; Restore original stack pointer

      ld    hl,(ldaddr) ; Get load address from header
      push  hl          ; Pass load address to record-1 routine ;(+4)
      push  hl          ; Fill another stack entry ;(+6)
      ld    hl,tbuff    ; Set for for CCP to call second record
      ex    (sp),hl     ; Make ZCPR34 go to tbuff, and return with
                        ; ..load address in HL

load: jp    0           ; Return to CCP (hot-patched above)
                        ; Command processor loads record 1 to
                        ; ..tbuff and records 2+ to load
                        ; ..address, then 'returns' to tbuff

      ds    2*13        ; Space for local stack
stack:      ds    2     ; Place to keep original stack pointer

      .dephase

;=============================================================
; Record 1 Routine  --  Code Relocator

; Standard code from T4LDR11.Z80 omitted.  We pick up with
; the new code right after the relocation is complete and
; interrupts are reenabled.

      ld    hl,0        ; Force warm boot
      scf               ; Set carry flag so HL will be used
      ret               ; Return to MLOAD3's caller in ZCPR34

      .dephase          ; Good Housekeeping

      end

; End of T4WBLDR.Z80 Relocator
```

is used as the load address. No checking is performed to insure that the code can safely be loaded to the requested address. The user must accept that responsibility. It would be nice if the code could check the address, but the basic code barely fits as it is in the space allowed.

The code for the second record (the relocation routine) is the same as in T4LDR11. The new code for the first record is shown in Listing 5. You will notice that an internal stack is provided. When I tested this code, I found that the CCP stack could not hold everything that XBIOS pushed onto the stack during the BDOS function calls. As I recall, Howard Goldstein did not have this problem when he tested the code on a different computer. Including the local stack in the code seemed to be prudent.

ZCPR34 Release E

Howard Goldstein recently finished implementing a number of minor bug fixes in the ZCPR34D code, which had been the standard production version for some time. At the same time, he also added a new feature invented by Bruce Morgen. The result is version 3.4E.

I will not go into Bruce's reasons for introducing this innovation, partly because this column is already long enough and partly because I don't really remember right now exactly what all of his reasons were. However, my final example will make use of this new feature. Please bear in mind, however, that unless you have ZCPR version 3.4E (which few people have), this final type-4 loader will not work. If anyone who owns a copy of ZCPR34 wants to experiment with this new feature, please contact me about getting the updated version.

In the previous versions of ZCPR34, as we described above, the first record in the type-4 loader calculates the load address and passes it back to the command processor. The command processor then loads the code to that address, adjusts any addresses in the code (that's the relocation), and then—here is the crucial point—executes the code at that load address. Bruce's change allows the execution address to be determined independently of the load address.

Here is the essence of Bruce's observation. The second loader routine normally just performs the code relocation and then returns to the command processor, which proceeds to execute the code. However, some interesting possibilities can be introduced by allowing this second routine to decide that the execution address should be changed from the load address that was calculated by the first loader routine. Bruce observed that the carry flag happened to have been left clear (reset) by all existing type-4 loaders and by the internal CCP code used to load all other types of programs. Bruce added a tiny bit of code to ZCPR34 (you can see it very near the beginning of Listing 1) to have it check the carry flag. If the flag is set, then the contents of the HL register pair are copied into the execution address stored in the CCP.

Our final example of a new loader, T4WBLDR1, is a further variation suggested by *TCJ* editor/publisher Chris McEwen. When I showed him T4T3LDR, he noted that it could be used to install code as a kind of TSR (this is an MS-DOS term for Terminate-and-Stay-Resident, code that loads itself and remains in place for further use) in the NZCOM user buffer area. T4WBLDR1 can be used that way.

The one essential change is a small addition to the code in the second header record to load the HL register pair with the address 0000H and set the carry flag before returning. This causes ZCPR34 to perform a warmboot (whose entry point is at address zero) instead of executing the newly loaded and relocated type-4 program code. The code is now available for execution by the JUMP command at a later time.

Since the NZCOM user buffer area is above the CCP entry address, the usual address verification code in the first record of the header in T4T3LDR would refuse to use the specified address. To get around that, we simply remove all of that code and accept the user-specified address without questioning. Listing 6 shows the code for T4WBLDR1.Z80.

Conclusion

The implementation of the type-4 program follows the tradition of modularity and flexibility that is the hallmark of the Z-System. Because of those two characteristics, there are always more ways to use any Z-System facility than its authors envisioned when they created it, and this allows for and encourages a steady stream of interesting new developments. I hope my examples here will inspire more innovation in the area of type-4 programs. ●

Computer Corner, from page 48

of contents, and then select one section and home in on it. If that section meets their muster it is a buy, if not, sale lost.

FORTH DAY

That overly thick manual was up for sale at the November Forth DAY in the San Francisco bay meeting. I saw lots of people look at it and put it back down quickly. Time is far too short for most people these days. Buying any new project that requires tons of reading to master it, just doesn't cut it. Don't get me wrong, if I need to use the product, I will wade through all the pages to find out what I need. However don't ask me to be happy about it.

Well I would say being happy about Forth was not on the agenda at Forth Day. It seems people are still concerned about where Forth is going, if it is going anywhere at all. There were some good speakers and a few good products for sale. About the only real news item was SUN's and Bradley's pushing for a standard FORTH ROM BASED DEBUGGER. As some of you know the SUN's SPARC based computers boot from a Forth based ROM. Also the expansion cards all have Forth based routines that load their drivers.

Well SUN is so happy with this operation that they are pushing to get a standard word set and operational structure established. It sounded like it will stay rather closely to what the SUNS' ROM does now, but soon all manufacturers may start changing their ROM BIOS's to Forth. From a maintenance stand point I can not support it strong enough. I hate how each vendors debugger works differently as well as always having just too few options to solve my problem.

On Forth itself, things look rather flat. Some more chips are scheduled to come out from Chuck Moore. These will be 200 MHz machines running a form of Forth. Chuck designs the chips with his own Forth based CAD system. He does this because none of the chip making CAD systems on the market can do what his does. It also means he can add features as he sees a need for them. Will these new machines make any big inroads, I doubt it as well as Chuck doubts it. Seems nobody believes he can make a 200

See Computer Corner, page 21

HARDWARE HEAVEN

Trade Mags

By Paul Chidley

Back Again?

Well, curses on you all! Well not really. I wrote the first article just to let off some steam. Unfortunately someone must of liked it because I was asked to write some more. In some ways I was kind of hoping someone would like it, but I had hoped that enough people would say thumbs down so I could spend my afternoons smoking chips instead of typing at my desk. So after a few hours of sniffing solder fumes here I am with *Hardware Heaven* number two.

Now, first let me say that this was not my pick for a title for this column. My first choice I rejected myself to avoid a liable suit from another column. My second choice, *TCJ* turned down probably because they wanted to avoid a liable suit. So here we are with *Hardware Heaven*. Or is it hell? Some days I wonder. Anyway I'm on my way to Trenton for the big computer festival, or "How to spend $600 on air fare to drink beer, eat pizza and talk weird stuff with other Hackers." Hope to meet some of you there.

Trade Mags

As for the column I thought I should dive right in and scare the Editor by talking about other magazines. Not the one sold by the pound, or the PC mags that live in their own little biospheres but the "trade" magazines. Trade magazines are little corners of guru worship all their own. Now what do I mean by a trade magazine? Well those who know what I'm taking about know that *TCJ* has nothing to fear from these guys. A trade mag is basically a glossy professional magazine full of expensive looking ads aimed at ivory tower engineers and R&D managers. The ads are only for the latest greatest toy, no doubt better than last week's toy. The articles are much the same since they are usually written by tech writers from the same companies that placed the ads. These mags are also very focused, aiming for a specific audience and therefore they are also very narrow minded. For example, you might find one called *Component Mounting* dedicated to "pick & place" machines capable of auto mounting components on your PCB. Now don't laugh, that's how focused they can be. So why would anyone be interested in them? Well they are a tool. (You know, like a hammer, saw, IBM-PC, et cetera). Like any tool, what they can do for you depends on how well you use them. I skim read every one that passes my desk. I use those cute little markers to flag any article or ad that caught my interest. I use them as a learning tool to keep me up to date with trends in the electronics industry. Sometimes I find new neat little chips that are actually useful (Dallas' DS1233). Then I file them away.

Now why would I pay good money to see the latest available 32-bit embedded controllers to ever add AI to a soda machine? [Soda machines use 68HC11's!–Ed.] Answer: I don't. The ones I read are free. The only catch is that you are "qualified" to be a free subscriber, otherwise they want a few hundred dollars a year. So what is a "qualified" subscriber? Someone that can fill in the form with the correct answers to the questions. These magazines survive on the high fees they charge to advertisers. They can do this because they show the advertisers that their readers are exactly the kind of people the advertiser is trying to reach. If you know what that is and fill out the form accordingly then they will add you to their list. As before you don't lie, but you do answer with what they want to hear.

Now where do you find such mags? Well the public library is a start but not likely a successful one. Try the library of your local electronics college or university. Try a friend who works in the electronics trade. Our library at work has over 50 titles available. Once you find a few, photocopy or tear out (with permission) the subscription card, fill it out, and send it in. The worst that will happen is that you never get a reply, but with luck you can add several pounds a month to the local letter carrier's bag.

Dallas Semiconductor DS1233

Well here's my chip of the month. (Now before you write to the editor, Yes I know that *TCJ* is every second month, I actually pick a chip each month but only write about every second one). It should be pointed out that I do not work for Dallas Semiconductor, they just happen to make some of my favorite chips. The DS1233 is known as the EconoReset. Its features are listed on the data sheet as;

- Automatically restarts the microprocessor after power failure
- Monitors push-button for external override
- Internal circuitry debounces push-button switch
- Maintains reset for 350mS after Vcc returns to an in-tolerance condition or push-button released
- Accurate 10% or 15% microprocessor power supply monitoring
- Reduces need for discrete components
- Precision temperature-compensated voltage reference and voltage sensor
- Low-cost TO-92 package
- Internal 5K ohm pull-up resistor

Paul Chidley is a senior technologist at NovAtel, an Alberta based cellular phone company. He is a neophyte ZCPR user, but has been active in homebrewed hardware and software design for many years, primarily in the Ohio Scientific and 6502/816 area. Paul can be reached on GEnie (email address: P.CHIDLEY), by regular mail at 162 Hunterhorn Drive NE, Calgary, Alberta, Canada T2K 6H5, or by telephone at (403) 274-8891 during reasonable MST hours.

In other words this is the equivalent of your standard reset circuit. The Yasbec has such a circuit. An RC to provide the power-on delay and switch debounce. The switch is in parallel to the cap, this signal is fed to a HC14 schmit trigger inverter. This provides a nice clean logic signal rather than the RC curve but it is the wrong polarity so a second inverter is needed. This circuit performs fine unless the power is cycled faster than the cap can discharge so a diode is needed to discharge the cap at power off. Purists would also say the a series resistor is needed to limit the discharge current through the diode. Our simple little reset circuit now has seven parts and still only does half of what the DS1233 does. So what does this neat little part look like? A 84pin PLCC? A 16pin dip? Sorry, it only has three leads and comes packaged in your standard TO-92 (like a transistor) or SOT-223 surface mount packages.

For the TO-92 package;
Pin 1 — GROUND
Pin 2 — _RESET
Pin 3 — Vcc

A normally open optional reset button and a capacitor are connected between _RESET and GROUND, the _RESET also going to your microprocessors reset input. That's it. You now have all those features listed above on your reset line. So why didn't I use one on the Yasbec? I wish I had but it wasn't out back then. So next time your circuit needs a reset reach for a DS1233. Contact your local Dallas Semiconductor dealer or rep for a data sheet.

New News

Speaking of new chips, a manufacturer of PC chips has announced an XT in a chip. Not a new processor but an XT. Complete with CGA, bus interface, floppy interface, and such. Why would anyone want an XT/PC in a chip? Well if you're writing code on a PC for an embedded controller why not make that embedded controller a PC. So if the idea takes off, your next ATM transaction could be handled by a XT running MeSsyDOS. Scary stuff.

New on the Yasbec front is the memory board and backplane. I've order a run of boards so I can have them available at Trenton. AND...while typing this article I have on my other monitor a nice digitized image of an Apache helicopter from a DeVry ad on TV. Its still a little rough around the edges (so to speak, graphically that is) but I plan to debut the colour video board in the hotel room at Trenton. I can picture it now, seedy looking characters coming and going all weekend long. Especially the wee hours of the night. Talking about the nice lines. The flushed colours. And oh those pixels. I'll be lucky if someone doesn't stage a raid. Deal of the Century

Have I got a deal for you! If you have a 286 PC or better and you like drawing schematics or even designing your own PCBs, then this is for you. PADS Software, Inc. has announced a $50 shareware version of their PADS-Logic schematic capture and their PCB-PADS pcb layout software. The shareware version is basically the same as the software selling for over a thousand. So why only $50? (for both, not each) Well the shareware versions are limited as to the size of the design they can handle. Two logic sheets for schematic capture and approximately 30 ICs for board layout. The companies' idea is to make friends in the low end market so that if you should need to buy the real thing you will already be sold on PADS. Sounds like good marketing to me and the real winners are the hobbyists (midnight hackers) like us who now have professional software for our home projects. ●

References

PADS Software, Inc.
119 Russell Street
Littleton, MA 01460
(508) 486-9521
(800) 255-7814

Dallas Semiconductor
4401 South Beltwood Parkway
Dallas, Texas 75244-3292
(214) 450-0448

Computer Corner, from page 19

MHz machine. As us Forth people know, they said the same things about the Novix and RTX 2000.

Allen Freeman gave the meeting commentary at the end of the day. Chuck gave his fireside chat at dinner and was rather interesting as always. Allen's talk focused mostly on what or how we might change the direction of Forth. The most important point he made was how Forth lacked integration with existing source and libraries of code. As we all know C has taken over for the most part. The why of that is the large and included C libraries. When someone buys Forth they do not get any libraries. However what they do get is usually all the source code for the Forth and as such no need for the libraries.

For new users buying into Forth however, it seems like they are not getting as much as if they bought one of the new C compilers with it's many megabytes of libraries. If you want to talk about manuals that are over an inch thick, talk C. Where I think this issue really becomes important is accessing new cards and devices. Allen rightly pointed out how most hardware vendors are giving free C code or libraries with their products. If you want to hook Forth into the new product you will have to code it all yourself. In some cases the code is a linkable module without any source to translate to Forth.

What the vendors of Forth need to do is make means of linking in C object modules to their code. It seems to me that might open up applications in Windows and embedded systems that have been going to C. The libraries and linking to object modules are two items Forth people have been passing over as not needed. But are they really not needed?

In discussing this topic with friends it has been pointed out that PolyForth does support libraries. The problem then is just letting people know about it. What I really want right now is a windows based Forth. I am actually looking at using Visual Basic as a means of developing a few quick and simple windows programs. What I would rather be doing is developing them using Forth. I want a F-PC like windows based system where I could take Forth code and add the windows features. Basically what I am talking about is upgrading existing programs to work with windows.

That's not a big request, but then it seems the vendors have not yet moved their products to windows. My question to them is what happened to the old statement that Forth is always the first

See Computer Corner, page 23

REAL COMPUTING

Computers Get Caller ID, Programmers Get No Respect, and Minix Gets a GUI

By Rick Rodman

Earth-Shattering Discovery

Time magazine recently announced: "Why are men and women different? New studies show that they are born that way." Who did these new studies, The Obvious, Inc.? (See *TCJ* 48.)

Of a similar mindset, there are those who say (with a straight face, too) that you should not have Caller ID because the caller's right to remain anonymous is more important than your right to know who's calling. Sometimes it seems that this great mission to protect the rights of the deranged, obscene and criminal elements goes a little too far in trampling upon the rights of the would-be normal.

Anyway, if you live in an area in which Caller ID is available, there is a really simple device available from Bell Atlantic which transmits the data in RS-232 form. It costs about $50. Now this device, it seems to me, would be a natural for BBS systems. The BBS software could maintain (or just print) a log of all the callers and the time they called. Then if any irregularity should occur, the number would be noted. Another possibility is comparing the telephone number against that used by a user before. However, the user could be calling from a friend's computer or from his office. Remember, too, that Caller ID will only work for local (that is, intra-LATA) calls. However, this could change in the future.

Of course, the greatest present threat to BBSs is the RBOCs, most notably Southwestern Bell and US West, who are attempting to force BBSs to pay business line rates. In fact, they have plans afoot to impose surcharges *above* business rates. This is because they are already viewing free BBSs as competition to their as-yet-unavailable information services. Monopolies cannot compete on a level playing ground.

Where are all the APIs?

Let's say you've just invented a fantastic new computer peripheral. What makes this device really neat is the way it could be integrated into other software applications running on a personal computer. This means software—so you have to develop an Application Programming Interface, or API, for the device and provide some software drivers for it. What do you do now?

Well, if you're like most manufacturers of such devices, like fax boards and hand scanners, you keep the APIs *a deep secret*. After all, aren't they part of your competitive edge? Then you provide some shlock software with the device that can't be integrated into any software applications. Eventually, you have a small part of a small market consisting of clumsy devices that are impossible to integrate.

If it was me, I'd give the APIs away. I'd try to make sure that every programmer in the country had those APIs. Just think—your device could be directly supported from WordPerfect or Microsoft Windows. *This* is how standards are made—because most manufacturers hold their APIs too close to their chest, the ones who dare to give them away become the standards.

As for some companies who will provide APIs (in the guise of "development toolkits") at very high prices so that only "major players" can participate in the party, remember that virtually *all* innovation in this industry takes place in small companies—where people design things because they *want* to, not because they're paid to.

Everyone who does any system integration out there is constantly running into these problems. Hey, you manufacturers out there: If you send me your APIs in machine-readable form, I'll be pleased to put them on a BBS where everyone can access them. Let's put them on a CD-ROM! (By the way, where's the CD-ROM API?)

Minix News

Some Minix users out there have successfully ported the Bellcore MGR graphical user interface to Minix. This means that Minix now has its own GUI and it can join the big leagues. No longer will people have to keep on asking, "But when will we get X Windows?"

A couple of caveats apply. I'm told that in 16-bit (read: standard PC Minix), the 64K code / 64K data limitation is so severe that only *two* windows can be opened. In 32-bit (Minix 386), this limitation goes away. Presumably under ST or Amiga Minix, these limitations wouldn't apply in the first place.

Since Minix is not public domain, of course, all of these additions are distributed in "context diff" form, as I discussed before. So, first you apply the virtual console patches, then the patches to those patches; then you take the MGR files and apply patches to *them*, then you concoct the whole mass together. Oh, but before you get started on that, if you're running PC Minix, you have to build Minix 386 first. Set aside a week for that first.

I didn't mention, either, that there's no compatibility between the 16-bit and the 32-bit executables. You need to build all new 32-bit executables. It's especially handy to have executables to log in, list files, copy files, and so on.

This whole pyramid scheme of context diffs depends on people being able to "cdiff" against a standard configuration. A common problem is people considering a particular patch combinations "standard"—for example, the virtual console patches mentioned above are fairly common—but this means that people without those patches, or with some other patches, may not be able to use a given patch file.

Another word of caution here: It's very easy to get carried away and make your own one-of-a-kind operating system that's incompatible with everyone else's.

Alas, I hadn't received MGR in time for this column. I'll have my experiences with it next time. I'm planning to implement

MGR on the PC-532 using an OS/2 window as a "graphics terminal". The PC-532 has an annoyingly narrow umbilical to the universe—a single 9600-baud serial channel is the only way data can get in or out. Then, due to some unknown problem, I haven't been able to get either Zmodem or Umodem to work. What I really need is a SCSI floppy drive or controller. (It's got to come with an API.)

Linux

Linus Torvalds of Finland has written a Unix clone, which he calls Linux, and is giving away. This OS is a pretty good clone of BSD, with 386 dependencies (such as the MMU) coded right in. Now as this column has stated before, the 386 MMU is a blatant rip-off copy of the 32532's, which in turn is a simplified version of the VAX's, so some work is underway to port Linux to the PC-532.

Linux has been causing a sensation on the Internet, for several good reasons. The main reason is, of course, it's completely free. Linus wrote it completely for fun, on his own time. He has no objection to people giving it away or enhancing it in whatever way they see fit. And, it's a more Unix-like design than Minix, which means that porting tools to it has proven easier (for those who have it).

Andy Tanenbaum argued recently that Linux is an "obsolete design", because it is "monolithic"; that, among OS designers, the microkernel design is ascendant. In a microkernel, the kernel does nothing but switch tasks and message traffic; memory management, file systems, et al. are outside the kernel. The most well-known microkernel is Mach. At any rate, he said that "Minix is a microkernel-based OS" but it's "only a hobby" for him.

Of course, it's evident that, because of Minix's original design around the 8088 processor, Minix is a poor implementation of the microkernel concept. And I'm sure many people wish they could have hobbies as remunerative as Minix has been for Andy.

One reason that Minix is a poor implementation of the microkernel concept is that memory managers and file systems cannot be dynamically loaded and unloaded. The kernel and the file system are actually tied together by several knotty strands of code. One of my design features for Bare Metal (mentioned in *TCJ* 41) was separate "managers" which could be loaded to support different directory formats, such as MS-DOS, CP/M, et al.

While I agree that Linux will be difficult to port to different environments, it should be pointed out that porting Minix isn't a piece of cake, either. Unlike a true microkernel, no part of the OS can be brought up without all the others.

What with Mach being released in AT&T-free form, MGR, and Linux all happening, one wonders whether the Free Software Foundation will ever get around to releasing their OS, which they reportedly now call "Hurd". (Presumably, a "hurd" is a group of "gnus"?)

More C sickness

Consider this statement (x and id are ints):

```
x += (( id - 10000 ) * 10 );
```

The variable "id" varies between 10000 and 10020, let's say. Since the subtraction is in parentheses, the maximum value which should be multiplied by 10 is 20, right? Well, that's not the way Microsoft C 6.00a sees it. No, Microsoft reports reports an "overflow in constant arithmetic"—as though the parentheses weren't there. No excuse this time—this is a bug.

Put this in your unofficial Microsoft bug list. If you want to get an *official* Microsoft bug list, you have to pay $300 for access to a bulletin board. Personally, I think they're a bunch of Stalinists. At least they haven't started threatening people who reveal their bugs to the public. Yet.

Other software vendors are pretty Stalinist, too. Oracle, for example, prohibits any database benchmarks from being published, besides their own (laughable) figures. I am free to tell you Oracle is slower than molasses in January—but I can't tell you *how much* slower.

That's why you should go with a free-world OS like Z-system. You can pick up your phone and talk to Jay Sage. You can ask about bugs and performance all you want, and you can get honest information in the pages of *TCJ*. Yet another advantage of Z-system over Microsoft Windows—*support!*

Next time

Well, I didn't get to the image compression/decompression stuff yet. But, looking back, I haven't been too accurate in predicting the next column anyway.

Next time I hope to have more facts on MGR and, hopefully, Linux. If you're interested in experimenting with these, or with other items discussed, drop me a line and we'll see how we can fix you up.●

Computer Corner, from page 21

language available on a new system. I guess the vendors do not consider windows a NEW system then.

MOVING

Well, I have said enough this month, so guess it is time to start packing my books and very heavy manuals again. I am not looking forward to moving again but hopefully this time it will be many years before I have to do it again.●

When people ask me, "What one factor do you believe has contributed most to the growth and influence of your organization?" I don't have to stop and think about an answer. Unquestionably it has been the emphasis laid from the very beginning upon human relationships—toward the public on the one hand, through careful service, and giving the utmost in values; toward our associates on the other hand.
—J. C. Penney

Remapping Disk Drives through the Virtual BIOS

By Roger Warren

INTRODUCTION

As a Z-Node sysop and believer in both Murphy's Law and the concept of the self-fulfilling prophesy, I've resigned myself to the monthly task of performing a full backup of my system's hard disk. This task consists largely of sitting in front of the machine for *hours* at a time listening to the floppy drive step in and out (between the directory and the storage area) with occasional breaks during which fresh disks must be inserted into the drive.

During one such monthly labor of love (read paranoia), I reasoned that if I relocated the disk directory of my backup floppy to a central track location, less time (on the average) would be spent stepping to and from the directory. Simple arithmetic convinced me that I could complete my 40 Meg backup in significantly less time—by as much as an hour—even with step rates of a few milliseconds per track.

With that inspiration to motivate me, I set about designing and implementing a few simple changes to my Ampro BIOS that allowed me to selectively enable or disable the relocation of disk directory on individual floppies. I was quite pleased with the results. These changes were documented and released to the public several years ago.

Recently, I was asked to discuss what I had done in an article for *TCJ*. I was not too certain that such an article would be helpful to more than a few people—those who have their BIOS code and aren't intimidated by the prospect of making and installing a new BIOS. However, in the same issue of *TCJ* that my future article was announced, Jay Sage's column covered the NZCOM Virtual BIOS and ways of using it. I was struck with the simplicity of being able to incorporate the general scheme of my disk track remapping into a loadable module and, most importantly, one that was *not* machine-specific. Such an implementation would be useful to many and painless to install and use.

This article documents this NZCOM track remapping approach. There is sufficient discussion about the technique and detail in the listings to allow anyone having the wherewithal to modify his or her BIOS (if that is preferred or if NZCOM is not being used) to do so, too.

THE BOTTLENECK

Floppy disks on CP/M machines are logically partitioned into three (and sometimes four) areas:

- Reserved tracks (also called System or Boot tracks)
- Directory tracks
- Storage tracks
- Spare tracks (the 'sometimes' category)

These logical areas are arranged physically in the order in which they're presented above. The System tracks are at the outer tracks of the diskette (the tracks having the lowest physical address), the Directory tracks are next, and the Storage area builds inward (toward higher numbered tracks) from the end of the directory. The spare tracks, if provided, occupy the innermost track area. They're a hold-over from the IBM 3740 (8 inch) disk format specification used in the older CP/M machines. These tracks were set aside to be swapped in as replacements for tracks that went 'bad' in the storage area. I'm not aware of any CP/M implementation that ever used them. Spare tracks are not set aside by any machine I've run into that uses 5 1/4 inch drives.

It is the physical location of the directory tracks that is a burden on disk system performance. When writing files to disk, every time the current disk allocation block become full, the BDOS must read the directory, allocate another storage block, update the directory, then seek the newly allocated block. When reading a file, the directory must be referenced to locate the file's next allocation block once the end of the current block is reached. As the disk becomes full and data for files are placed in tracks further 'inboard' on the disk, it's a longer and longer trip between the storage tracks being used and the directory. Even with track-to-track step rates of a few milliseconds, the round trip is time consuming. The activity is quite noticeable if you're listening to the disk drive!

If one were to "fake out" CP/M by swapping the directory tracks with a couple of tracks in the middle of the disk, this bottleneck would be eliminated. With the directory location swapped in this way, the directory would *never* be further away from the current track position than half-way across the disk. But which tracks should be swapped?

Arbitrarily choosing tracks wouldn't be optimal for all cases. Of course, 48 TPI and 96 TPI disks have different numbers of tracks. Some BIOSes treat double sided disks as having twice the number of tracks as a single sided disks, while other BIOSes handle double sided diskettes by doubling the number of sectors per track. Clearly, the track number(s) of tracks to be swapped with the directory are system dependent.

Roger Warren is a former metallurgical engineer who has been making a living as a system software and hardware engineer since being introduced to computing in the early 70's. He's been in the aerospace and defense fields for a majority of that time, working primarily with embedded controls and radiation-hardened memory systems.

Roger has been active in the CP/M and ZCPR communities for several years, having developed and released several enhancements to the Ampro BIOS and, more recently, LZH compression for CP/M (CRLZH and UNCRLZH). He can be reached for comment on his Z-Node, "The Elephant's Graveyard" at 619-270-3148 (PCP: CASAD).

Listing 1 BIOS label and modified vector table

```
        name    ('BIOSW')       ; NSCOM needs 'BIO'
                                ; ...as 1st 3 chars.

        common  /_ENV_/
z3env:
ccp     equ     z3env+3fh
dos     equ     z3env+42h

        common  /_CBIO_/
cbios:
        cseg
                .
                .
       (detail omitted)
                .
                .

; Beginning of NZBIO.  The header structure is absolutely
; crucial to the correct operation of NZ-COM.
; DON'T CHANGE IT.

; Beginning of Header.....

start:  jp      boot            ; Cold boot
wboote: jp      wboot
        jp      const
        jp      conin
        jp      conout
        jp      list
        jp      punch
        jp      reader
        jp      cbios+24        ; Home
        jp      swpseld         ; Seldsk is performed below
        jp      swptrk          ; Settrk is performed below
        jp      cbios+33        ; Setsec
        jp      cbios+36        ; Setdma
        jp      cbios+39        ; Read
        jp      cbios+42        ; Write
        jp      listst
        jp      cbios+48        ; Sectran

        ds      (30-17)*3       ; Room for 30 jumps
```

Listing 2 Data definition and SELECT DISK routine

```
; TRACK SWAPPING begins here.  First, various equates,
; offsets, and storage definitions.

dpbptr      equ     10          ; DPB ptr is 10 bytes
                                ;  into DPH

; DPB offsets:

spt         equ      0          ; Sectors per track
bsh         equ      2          ; Block shift factor
dsm         equ      5          ; Storage block max
cks         equ     11          ; Check vector size
off         equ     13          ; Offset


swapfl:     db       0          ; Flags whether this disk
                                ;  has tracks swapped
offset:     ds       2          ; Holds offset (# reserved
                                ;  tracks)
halftrax:   ds       2          ; Holds computed half-way
                                ;  track
lastdrive:  db      0ffh        ; Holds last drive selected

; An ID string so that a support utility which wants to
; manipulate SWAPVEC can determine that the BIOS is
; installed.

            db      'SWAPVEC'
swapvec:    dw      0000h       ; Swap Vector LSB is drive
                                ;  A, MSB drive P.

; Swapping code begins with disk selection.  If the drive
; being selected is the same as the last one and there has
; not been a warm boot since its last access, then the
; current swapfl, offset, halftrax, are valid.

swpseld:    ld      a,(lastdrive) ; last drive selected
            cp      c           ; Same as last drive?
            ld      a,c         ; Move to A
            ld      (lastdrive),a ; Save
            jr      nz,swsl     ; If not zero, new drive
            ld      a,e         ; Same drive, test
                                ;  new mount
            cpl                 ; Flip bits
            bit     0,a         ; Z flag is set if
                                ; old mount

swsl:       push    af          ; Save Z flag
            call    cbios+27    ; Invoke the BIOS, get
                                ;  the pointer (to H/L)

; Check for bad disk select

            ld      a,h         ; High of vector
            or      l           ; Test vector for 0
            jr      nz,oksel    ; Jump if select OK

; Here if SELECT failed.  Might have been a LEGAL drive with
; an IO error.  Protect against that by changing lastdrive.

            dec     a           ; Was 0, becomes 0ffH
            ld      (lastdrive),a ; Zap lastdrive
            pop     af          ; Tidy stack
            jp      swapex      ; Was bad select!
                                ;  Skip it!

oksel:      pop     af          ; Get flag from stack
            jp      z,swapex    ; Zero if same drive and
                                ;  old mount--just exit
            push    hl          ; Save for exit
            push    ix          ; Save the IX register

            xor     a           ; Assume disk isn't
                                ;  swapped, get a zero
            ld      (swapfl),a  ; and Clear the flag

; Determine whether we're swapping this disk

            ld      a,(lastdrive) ; last drive
            neg                 ; Make -(lastdive)
            add     a,16        ; There are 16 drives.
                                ;  16-lastdrive is number
                                ;  of bits to shift in
                                ;  swapvec
            ld      b,a         ; Copy to counting
                                ;  register
            ld      ix,(swapvec) ; Get the swap vector
flgloc:     add     ix,ix       ; Move bit to carry
            djnz    flgloc      ; Continue shifting till
                                ;  count done
            jp      nc,exitix   ; No carry, drive not
                                ;  track swapped

            ld      de,dpbptr   ; Get offset in DPH of
                                ;  DPB pointer
            add     hl,de       ; Point to DPB pointer
            ld      e,(hl)      ; Get DPB pointer low
            inc     hl          ; Goose HL
            ld      d,(hl)      ; Get DPB pointer high
            push    de          ; Transfer DE to ...
            pop     ix          ; IX

; Check for fixed drive (check size will be 0 for fixed
; drives)
```

A SOLUTION

If one were to divide the Storage tracks of a disk roughly in half and reverse the track numbers on the first half, the directory would be roughly at the center of the disk. Furthermore, the disk would fill up from the center outward until the outer bound of the Storage area was reached, then continue filling from the center inward until the storage limit of the disk was reached.

Although CP/M "thinks" in terms of logical tracks which, for diskettes, originally corresponded to physical tracks, the logical-to-physical mapping is, really, up to the BIOS. The differences in how BIOSes treat dual sided diskettes, mentioned above, are examples of this BIOS-dependent mapping (and the freedom the BIOS has in track assignment).

The reversing of the track numbers of the first half of the diskette is merely a logical-to-physical mapping. There is a precedent for this: the BIOS sector translation via which CP/M allows for a logical-to-physical sector number remapping to improve disk performance. There is even a specific BIOS entry point that BDOS uses to perform the sector translation.

Since the NZCOM virtual BIOS resides between the BDOS and the system's BIOS, it can provide the two necessary portions of the track re-mapping proposed above. These portions are:

1. Determination of the track number of the center of the diskette (a system-dependent number) and,
2. Translation of track numbers for the tracks in the first half of the disk.

When a disk is selected, the BIOS returns the BDOS a pointer to a table of information. Through this table (and tables the table points to) all of the information necessary to determine the track number of the 'center' of the selected disk is available. The NZCOM virtual BIOS can use this information to compute this track number and leave it behind for subsequent use by the NZCOM virtual BIOS track selection routine.

```
            ld      a,(ix+cks)      ; Get low of check size
            or      (ix+cks+1)      ; And the high
            jr      nz,nothard      ; Not zero if this is a
                                    ;  floppy

; Now if we got here, the guy has specified that a hard disk
; is swapped.  Turn off the bit in the vector that selects
; this disk.

            ld      a,(lastdrive)   ; last drive
            neg                     ; Make -(lastdive)
            add     a,16            ; There are 16 drives.
                                    ;  16-lastdrive is number
                                    ;  of bits to shift in
                                    ;  swapvec
            ld      b,a             ; Copy to counting
                                    ;  register
            ld      ix,(swapvec)    ; Get the swap vector
            ld      hl,0            ; Clear hl
            scf                     ; Set carry so a bit is
                                    ;  shifted into the HL

flglc2:     adc     hl,hl           ; Shift bit into HL
            add     ix,ix           ; Next bit to carry
            djnz    flglc2          ; Continue shifting till
                                    ;  count done

; Now offending bit is in carry

            ccf                     ; So...reset it

; Continue shifting from IX to HL until the original bit set
; in HL pops out.  This will signal completion of all bits.

flglc3:     adc     hl,hl       ; Continue on
            jr      c,flglc4    ; Jump if done
            add     ix,ix       ; Get another bit
            jr      flglc3      ; loop

flglc4:     ld      (swapvec),hl    ; Save new vector
            jr      exitix      ; And exit

; Here to continue calculations for a floppy

nothard:    ld      l,(ix+off)      ; Get offset low
            ld      h,(ix+off+1)    ; Get offset high
            ld      (offset),hl     ; Save offset

; Now setup for computation of # tracks in this disk's
; storage area

            ld      e,(ix+spt)      ; Low of sectors/track
            ld      d,(ix+spt+1)    ; High of sectors/track
            ld      l,(ix+dsm)      ; Low of storage blk max
            ld      h,(ix+dsm+1)    ; High of storage blk max
            inc     hl              ; # of storage blks
            ld      a,(ix+bsh)      ; Get block shift count
            add     a,16+1          ; Add 16 for divide cnt
                                    ;  (+1 for dec @ preshift)
            ld      b,a             ; Transfer to counting
                                    ;  register
            push    hl              ; Transfer HL
            pop     ix              ; ...to IX

; Now begin divide of IX by DE. HL:IX is used as 32 bit
; quantity.  IY is quotient.

; First, shift till at least one bit is pushed out of IX.
; This will make the code run faster.

preshift:   add     ix,ix           ; Shift
            dec     b               ; Count
            jr      nc,preshift     ; Loop

            push    iy              ; Save IY so it can be used
                                    ;  for quotient
            ld      iy,0            ; Quotient starts at 0
            ld      hl,1            ; HL:IX is to be divided
                                    ;  (HL starts at 1.
                                    ;  We've shifted until a
                                    ;  single bit popped out
                                    ;  of IX. It was destined
                                    ;  for HL LSB).
            or      a               ; Clear carry....
            jr      div1            ; And warp into division
                                    ;  loop

divloop:    add     iy,iy           ; Shift quotient
            add     ix,ix           ; Shift divdend
            adc     hl,hl           ; ...propagate carry
                                    ;  (Note: leaves carry
                                    ;  CLEAR)
div1:       sbc     hl,de           ; Compare dividend to
                                    ;  divisor
            jr      nc,addq         ; If no carry, HL stands.
                                    ;  Add 1 to Quotient
            add     hl,de           ; Otherwise, restore
                                    ;  original H/L
            jr      div2            ; ...And continue below
addq:       inc     iy              ; Goose quotient
div2:       djnz    divloop         ; Repeat until
                                    ;  count exhausted

; Divide done.  If HL:IX is non-zero then add 1 to quotient
```

When a track is selected, the NZCOM virtual BIOS can use the previously computed "center" track number to translate the track number before presenting it to the BIOS.

The NZCOM virtual BIOS must supply three other important functions:

1. It must determine whether or not track remapping is desired for a particular drive,
2. It must provide for the presence of the System tracks (they shouldn't be remapped), and
3. It must protect against the inadvertent selection of remapping for hard disk partitions. THE TRACK SWAPPING VIRTUAL BIOS

While the concept the track swapping virtual BIOS is not too hard to follow, it winds up that the implementation requires more than just a handful of instructions. This stems primarily from the fact that while all of the information required to determine the number of tracks in a disk's Storage area is available via the pointer returned by the select disk routine, the data must be massaged to yield the number(s) required to perform the swapping. There is also a fair amount of code required to determine whether swapping is enabled for a disk and to protect against inadvertent selection of remapping for hard disks.

The code, itself, is rather straight-forward. The comments in the code include sufficient detail so as to make any "blow-by-blow" description here redundant. However, there are some points I'd like to stress. There is also some general information about the calculations and the BDOS/BIOS interface that isn't presented in gory detail in the listings that should be presented here. This material will be helpful to those wishing to fine-tune the track swapping for their machines.

Listing 1 presents some of the front matter of the virtual BIOS code and the virtual BIOS jump table as modified for the track swapping code. The changes from the "stock" virtual BIOS code are few: only the module name and jumps for the Select Disk and Select Track routines are modified. However, the listing is included for completeness' sake.

Listing 2 presents the track swapping Select Disk routine and related constants and variables. The Select Disk routine performs most of the work in the track swapping BIOS. It intercepts calls to the actual BIOS and determines whether swapping is enabled for the disk being selected. If so, the number of tracks in the storage area of the disk is computed and, from that, the track address of the "half-way" point on the disk. This value (HALFTRAX), the number of system tracks (OFFSET), and the swapping enabled/disabled flag (SWAPFL) are left behind for the Select Track routine.

Track swapping is enabled on an individual CP/M drive basis. A 16-bit quantity, SWAPVEC, is employed to indicate which drives have swapping enabled. The least significant bit of SWAPVEC corresponds to drive A, and the most significant to drive P. SWAPVEC is located immediately before the Select Disk code. This allows it to be located by a separate utility program, listed below, which manipulates the bits in SWAPVEC. The character string "SWAPVEC" precedes SWAPVEC in memory. The utility program checks for the presence of this string (to assure that the track swapping BIOS is loaded) before attempting to modify memory. This program has simple command line operation and can be incorporated into your STARTZCM start-up alias. Optionally, SWAPVEC can have an assembled-in default value.

I'd like to call attention to the use of NEW MOUNT flag passed to the BIOS from the BDOS at Select Disk time. When BDOS calls the BIOS's Select Disk routine, the least significant bit of the E register is 0 if the drive hasn't been selected since the last warm boot, and a 1 if it has. This item of the BDOS/BIOS interface is not well documented in most of the CP/M manuals I've run across, which is why I'm calling so much attention to it here. Indeed, some manuals don't mention it at all. It is, however, faithfully duplicated in the various BDOS replacements I've examined. BIOSes that permit the user to read diskettes of several different formats in the same drive use this flag to determine whether the rather lengthy process of determining the format of the diskette should be performed. The virtual BIOS uses the flag to help decide whether the number of tracks on the drive needs to be determined.

To minimize the impact of the additional code in the track swapping virtual BIOS on system performance, the Select Disk routine first checks to see if the drive being selected is the same as the last one selected. If so, the disk sizing calculations need not be redone. However, if this is the first selection of this drive since a warm boot, the disk parameters could change if the disk has been swapped. Therefore, when the NEW MOUNT flag indicates that this is the first selection since a warm boot, the calculations are performed even if the drive is the same as the previously selected one.

If the BIOS Select Disk routine returns an error, the virtual BIOS's previous drive variable (LASTDRIVE) is set to 0FFH. This is done to protect against true I/O errors on otherwise valid drives. If the drive is valid but the user left the drive door open, an I/O error would occur and the disk sizing calculations would be aborted. If he then closed the door and continued the operation, the next Select Disk call would likely be for the same drive. By forcing LASTDRIVE to 0FFH when an error occurs, the code

```
; (round up).  This is done since the last track of disk
; may only be partially used.  No need to test IX since
; >16 shifts will cause IX to be 0!

            ld      a,h             ; high
            or      l               ; continue test
            jr      z,evenmul       ; Zero if was even
                                    ;  multiple
            inc     iy              ; Add 1 to round up

evenmul:

; Now IY is # of tracks in storage area.  Divide by 2 to get
; half-way point.  First half will have reversed track #'s.
; First add 1 to make even if quotient is odd.

            push    iy              ; Transfer...
            pop     hl              ; Quotient to HL
            inc     hl              ; Add 1
            srl     h               ; Shift right
                                    ;  (divide by 2)
            rr      l               ; Finish shift of H/L

; Finally, we want to FAVOR the directory track(s).  Add 1
; to the computed 'halfway' point.  However, there is need
; to decrement the computed halfway point to make the math
; at the track re-mapping nice, so a +1-1 is a net change
; of 0.

            ld      (halftrax),hl   ; Save # tracks at
                                    ;  half-way point
            pop     iy              ; Restore IY
            ld      a,1               ; Set swapped flag
            ld      (swapfl),a      ; Into memory

exitix:     pop     ix              ; Restore IX
            pop     hl              ; Get saved DPH pointer
swapex:     ret                     ; Return to sender
```

assures that the disk sizing calculations will be performed for the drive if it's the next drive selected.

To protect against inadvertently enabling of swapping for hard disk drive partitions, a separate trap for hard drives is included. Swapping tracks is possible on hard drives, but to do so on an existing drive would wreak havoc. When the code detects that the drive it is performing calculations for is a hard drive (signaled by the fact that the directory checksum vector size is 0), the trap code resets the enable flag in SWAPVEC for that drive. By doing so, subsequent selections of the drive will not waste time making the same discovery over and over again.

The real 'work' in the Select Disk routine is the calculation of the number of tracks in the diskette's storage area. This is done by dividing the number of sectors per track into the number of sectors in the storage area of the disk. The number of sectors per track (SPT) is available in the disk parameter block (DPB). The number of sectors in the storage area is not directly available and must be computed by multiplying the number of storage blocks in the drive by the number of sectors per storage block. The number of sectors in the storage area is determined by adding 1 to the maximum storage block number in the DPB (DSM+1). The number of sectors per block is determined from the block shift count (BSH) in the DPB. BSH is power to which 2 must be raised to yield the number of sectors per block. Shifting (DSM+1) left BSH times yields the number of sectors in the storage area. By adding BSH to the loop counter of the division routine (normally 16), this shift operation is incorporated into the division. Note that the last track in the disk's storage area may be only partially used. Code must be added to 'round up' the computed number of tracks if the division leaves a remainder.

Next, the "half-way point" track number for the disk is calculated. The number of tracks in the storage area is incremented by 1 and divided by 2. If the number of tracks is odd, the increment operation will assure that the extra track will be placed in the "lower" half of the disk. This will be slightly more favorable, since it will more 'centrally' locate the directory.

Finally, the "half-way point" is adjusted to favor the location of the directory. The directory (usually 1 to 2 tracks in size) is accessed *a lot*. If there are X tracks in the storage area of the disk, X-2 are actual storage (assuming 2 directory tracks). So, we add 1 to the computed half-way point. However, there is need to decrement the computed half-way point to make the math at the track re-mapping nice, so there's a net change of 0. The reasoning behind this adjustment might be counter-intuitive, so consider this example:

> *For 100 storage tracks and 2 directory tracks, the half-way point is 50. By adding 1, the new half-way is 51. Tracks 0-50 are reversed, 51-99 are normal. Considering the nature of the storage area use, this means there are 49 tracks of data, 2 tracks of (centrally located) directory, the 49 more tracks of data.*

Listing 3 presents the track swapping Select Track routine and related constants and variables. The Select Track routine merely adjusts the track number (if swapping is enabled for the current drive) and passes the new number on to the BIOS.

THE SUPPORT UTILITY

Listing 4 presents a support utility to manipulate SWAPVEC. The help screen in the program listing details program operation, so no additional detail will be supplied here. However, the statement that calculates the address of the "SWAPVEC" character string preceding SWAPVEC:

```
swapvec = (* (char **) ((*(char **) 1)+25))-9;
```

can be a very obtuse line for the 'C' novice, so an explanation is in order:

> *The value 1 is cast as a pointer to a character pointer. Taking what it points to [*(char **)1] gets the contents of location 1 - the address of the BIOS warm boot location - as a character pointer. Adding 25 to the pointer yields the address of the location that holds the address of the select disk BIOS routine. By casting this value as a pointer to a character pointer and taking what it points to, we finally wind up with the address of the first instruction of the select disk routine. If the SWBIOS is installed, the select disk routine is immediately preceded by the characters 'SWAPVEC' and the 2 byte swap vector. Subtracting 9 from the address of the select disk routine will yield the address of the first character of the string.*

Listing 3 Track selection and remapping code

```
; Swapping code is completed with the track select.  All of
; the important work has been done by the SELECT DISK code.
; Here, we only need to remap the track.

; If swapping is enabled, the track is compared to the offset
; (# reserved tracks).  If the track is in the reserved area,
; nothing is done.  Next, the track is compared to the value
; of 'halftrax', computed during the disk select.  If the
; track is in that area, the track number is altered.  If the
; track is higher than 'halftrax', it is not altered.

swptrk:  ld      a,(swapfl)      ; get flag
         and     01              ; Swapping?
         jr      z,trakok        ; Jump if not

         push    bc              ; Transfer track
         pop     hl              ; ..to HL

         ld      de,(offset)     ; Get offset
         or      a               ; Clear carry...

         sbc     hl,de           ; Track less offset...
         jr      c,trakok        ; If carry, then accessing
                                 ;  reserved area

; Now, track is normalized, having a value from 0...highest
; track.  Subtract this from halftrax, stored above.  This
; will result in an input value of 0 yielding a value of
; 'halftrax', 1 yielding halftrax-1, etc. A value of halftrax
; yeilds 0.  If carry is set as a result of this operation,
; then we're accessing the upper half, so original track was
; ok.

         ex      de,hl           ; HL to DE
         ld      hl,(halftrax)   ; Get half
         sbc     hl,de           ; (Carry was clear)
         jr      c,trakok        ; If carry, track is in
                                 ;  second half of disk

; New track number in H/L. Add back in offset and transfer
; to B/C

         ld      de,(offset)     ; Get offset
         add     hl,de           ; Add it back in
         push    hl              ; Transfer to...
         pop     bc              ; ... BC

trakok:  jp      cbios+30        ; go select track
```

Listing 4 Enable/Disable support utility

```
/*
SWAPVEC
    A quick-and-dirty program to manipulate the SWBIOS track
    mapping enable vector from the ZCPR command line.

    Version 0.0
    01 Sept 91
    Developed at the Elephant's Graveyard by R. Warren
*/

#include "stdio.h"

char *swapvec;
char id[]="SWAPVEC";
char i,operation,*arg1;
unsigned bits,mask;

main(argc,argv)
int argc;
char **argv;
{
    putchar('\n'); /* throw CR and LF */

    operation = 0;
    if (argc <2) operation=1;
    else    {
            arg1=*(argv+1);
            if ((*arg1 >= 'R') && (*arg1 <= 'T'))
                operation = 2;
            else if ((*arg1 != '/') && (*arg1 != '?'))
                operation = 3;
            }

    if (operation) {

/* Locate swapvec */
        swapvec =  (* (char **) ((*(char **) 1)+25))-9;
        for (i=0;i<7;i++)
            if (id[i] != *swapvec++) {
                fprintf(stderr, "Cannot locate swap vector.\
\nMake sure track swapping bios is installed and try again.\
\n\n");
                exit(1);
                }
            }
    switch (operation) {
    case 1:
        printf("Swap vector is %04xh (LSB is drive A)\n",
                *(unsigned *) swapvec);
        break;
    case 3:
        sscanf(arg1,"%x",&bits);
        printf("Swap vector set to %04xh (LSB is drive A)\n"
                ,bits);
        *(unsigned *) swapvec=bits;
        break;
    case 2:
        bits=*(unsigned *) swapvec;
        while ((*arg1>='R') || (*arg1 <= 'T')) {
            i=*(arg1+1)-'A' ;
            if((i < 0) || (i>15)) break;
            mask = ((unsigned) 0x1) << i;
            if (*arg1 == 'S')       bits |= mask;
            else if (*arg1 == 'R') bits &= ~mask;
            else                    bits ^= mask;
            arg1 += 2;
            }
        printf("Swap vector set to %04xh (LSB is drive A)\n"
                ,bits);
        *(unsigned *) swapvec=bits;
        break;
    default:
        printf("SWAPVEC v0.0\n\n\
Usage:\n\
SWAPVEC             - displays current swap vector\n\
SWAPVEC <hexvalue> - set swap vector to hex value\n\
SWAPVEC <Set_Reset_Toggle string> - Set, reset or \
toggle drives per string\n\
SWAPVEC /           - display help\n\
SWAPVEC ?           - display help\n\n\
<Set_Reset_Toggle string> is of the form:\n\n\
   Sd, Rd or Td  where d is a drive letter (A-P)\n\n\
   The individual elements of the string can be\n\
   concatenated (no spaces).\n\n\
   eg:\n\n\
   SWAPVEC SFTGRC\n\n\
   will SET the bit for drives F, toggle the bit for\n\
   drive G, and RESET the bit for drive C.\n\n");

    } /* end switch */
}
```

CARE AND FEEDING OF TRACK-SWAPPED DISKS

I'll not detail how to prepare and load NZCOM virtual BIOSes. That information is presented well enough in the NZCOM manuals and does not need to be presented here. Suffice it to say that the track swapping BIOS can be made an integral part of your system, and that one can set, reset, or view the track remapping enable flags for the system drives via the utility discussed above. This utility can be run by your STARTZCM alias, so that your system can 'load' with track remapping enabled for selected drives.

With the track swapping virtual BIOS installed, one is free to enable and disable the feature for any floppy on the system. The following tips will help keep you out of trouble:

Track swapping has nothing to do with system's disk formatting. However, to avoid any unforeseen problems, it's probably better to not enable any swapping when using your system's disk formatter or disk test. (They may not work with NZCOM installed, anyway!)

The BDOS won't gracefully handle what it considers GARBAGE in the disk directory, so don't try to turn a 'used' disk into a track-swapped disk without preparing it first (old data on the disk will garbage-up the track-swapped directory). To prepare the disk, merely reformat it. I've also included a utility in the support library (see PARTING SHOTS, below) that will perform the necessary directory area initialization.

Track swapping selected on a CP/M logical drive basis - not on a physical drive basis. If your system supports drive swapping or reassignment, be aware of that fact. If your system has the ability to read 'foreign format' disks, track remapping shouldn't be enabled for the drive used for foreign format disks.

FINE-TUNING FOR SPECIFIC SYSTEMS

The track remapping scheme I chose is probably the most simple-minded one, but it is very good choice for my system and purposes. A variety of remapping schemes can exist. The only real requirement is that there be a one-to-one mapping of logical and physical track. Some schemes may be better than others for specific machines and/or specific applications. Some may be worse.

Recently, Jay Sage asked why I hadn't mapped things so that the directory was in the center of the drive with even numbered tracks mapped 'inward' from the center and odd numbered tracks mapped outward. With that kind of scheme, data tracks would be closer to the directory as the disk filled up, giving an even better performance enhancement (my mapping scheme only improves performance on disks that are more than half full.) Frankly, this hadn't occurred to me. It seemed like a wonderful idea, and I quickly tried it.

See Virtual BIOS, page 32

The Bumbling Mathematician—Part 1

Big Numbers

By Frank C. Sergeant

I have an urge to write about math. When I use the word "mathematician," I use it very loosely. I want to do big math things with small computer resources. Since Forth is my favorite language, I want to use Forth, particularly my Pygmy Forth, for the examples. Pygmy is available for MS-DOS, but not yet for CP/M. So, the first thing I want to do, which I absolutely cannot do now, is to prepare a version of Pygmy for you that runs under CP/M. If I could do that first, more of you could run the examples I wish to present. However, perhaps, if I present the code and explanations clearly enough, even if you don't have access to Pygmy, you will be able to use these ideas in assembly language or whatever other language you prefer. Unfortunately, I am not going to be able to present code that is as pretty as I'd like.

Why Big Numbers?

By big numbers I mean numbers with arbitrarily high precision. Would like to do your calculations with 50 digits to the left of the decimal point and 400 digits to the right of the decimal point? Well, this big number approach will let you. Why would you want to do this? I don't know. Because it is possible? Because you want higher precision than any of your other math packages give you? Because you object to the price of math co-processors, and you object to buying a PC into which to plug the co-processor?

Some months ago I wanted to calculate logarithms to 40 decimal places. So, I dashed off the enclosed code to let me handle big numbers. I had the choice of building a floating point or a fixed point package. I picked fixed point. It is not fast, it is not pretty, but it allows you to set the number of digits to each side of the decimal point to just about any size that will fit in your computer's memory.

The Details

This package allows you to create "registers" and double "registers" named whatever you wish. For general purpose registers I usually name them X, Y, Z, etc. I usually name the double registers AA, BB, etc. to remind me that they are double. These registers are really strings of bytes in memory. Let's assume you decide you want 10 places to the left and 20 places to the right of the decimal point.

```
30 CONSTANT #DIGITS     ( set total number of digits)
20 CONSTANT #FRACS      ( set number of digits to right)
                        ( of decimal)
```

The above code sets the total number of digits to 30 and the number to the right to 20 (thus the number to the left is 10).

Then we'll set up some registers and double registers.

```
N: X    N: Y    N: Z    ( make some registers )
DN: AA DN: BB           ( make some double registers)
```

X, for example, is 30 bytes long. Each byte holds a single decimal digit. Wasteful, you say? After all, we could stuff 2 decimal digits into each byte. The reason for using a full byte for each digit is it simplifies handling carries and borrows. Since each byte holds a digit no greater than 9, each byte will also hold the sum or product of any two digits. Thus, when we add two registers, we first add corresponding bytes together, without worrying about carries. Then we "normalize" the destination register so that each byte contains a number (digit) no greater than 9. The word ADD does both of these steps. Since the procedure for borrows is different than for carries, two words are available: NORMALIZE+ and NORMALIZE-. SUB uses NORMALIZE-. Suppose you want to add the contents of register X to the contents of register Y.

```
X  Y  ADD
```

does just that, leaving X unchanged and leaving the sum in Y.

```
Y  X  SUB
```

Subtracts X from Y, leaving the difference in Y and leaving X unchanged.

Multiplication and division are a little trickier. This is where the double registers become necessary.

```
Y  X  AA  MUL
```

will multiply Y times X and place the product into AA. It will leave Y and X unchanged.

```
Y  X  DIV
```

divides Y by X, leaving Y and X unchanged, and always placing the quotient into the double register named QUOTIENT.

In addition to adding, subtracting, multiplying, and dividing two registers, we need the ability to initialize the registers, compare them, copy them to one another, and print their values. The word X! (pronounced "x store") initializes a register. While we might like to do something like

```
75 X X!
```

Many years ago Frank saw a proof that the square root of two was irrational, and, while he may never be a mathematician, he has developed a great deal of respect for those who are mathematicians.

this would only work for small numbers (16-bit numbers). What if we want to initialize the register to something like 1,392,459,667.00000000011239? So, X! gets the initialization value from a string, as follows:

```
" 1,392,459,667.00000000011239"  X  X!
```

The Pygmy Forth Code

```
30 CONSTANT #DIGITS
20 CONSTANT #FRACS

'. CONSTANT PT  ( decimal point character)
 #DIGITS 2* CONSTANT D#DIGITS
 #DIGITS 1- CONSTANT #DIGITS-1
D#DIGITS 1- CONSTANT D#DIGITS-1
#DIGITS #FRACS - CONSTANT #WHOLES
: N: CREATE ( -) ( - reg)  #DIGITS ALLOT  ;
: DN: CREATE ( -) ( - dreg) D#DIGITS ALLOT ;
N: U  N: V  N: W  N: X  N: Y  N: Z  ( make some "registers" )
DN: AA   ( double accumulator for mult & div)
: ZERO ( reg -)  #DIGITS 0 FILL  ;
: DZERO ( reg -) D#DIGITS 0 FILL ;


: X! ( $ reg -)
  ( e.g.  " 12345.893" X X!          )
  ( number must be positive number, with a decimal point)
  DUP ZERO
  PUSH COUNT 2DUP ( a # a #)
  PT -LEADING<> 2DUP 1 +UNDER 1-
  R@ #WHOLES + SWAP CMOVE ( move in fractional part)
  SWAP DROP - ( a #) #WHOLES OVER - R@ + SWAP CMOVE (   )
  POP #DIGITS
  FOR ( a) DUP C@ DUP NOT NOT $30 AND - OVER C! 1+ NEXT
  DROP ;

: .x  ( a # - a')  FOR DUP C@ $30 + EMIT 1+ NEXT ( a)  ;
: .X ( reg -) CR #WHOLES .x  PT EMIT #FRACS .x  DROP ;
: .DX ( dreg -) CR #DIGITS .x 44 EMIT .X  ;


( scale number left or right - this can only be done to or
  from a double precision accumulator)
: REG! ( reg1 reg2 -) #DIGITS CMOVE ;
: DREG! ( reg dbl-reg -)  DUP DZERO  REG! ;
: DREG@ ( dbl-reg reg -)  REG! ;
: NORMALIZE+ ( reg -) #DIGITS-1 +  0 ( ie initial-carry)
  #DIGITS FOR ( a cy)
   OVER C@ + ( a v+cy) 10 U/MOD ( a new-v new-cy)
  PUSH OVER C!  1- POP  NEXT ( ABORT" overflow! ")  2DROP ;

: NORMALIZE- ( reg -) #DIGITS-1 +  0 ( ie initial-carry)
  #DIGITS FOR ( a cy)
   OVER C@ + ( a v+cy) 245 OVER U< DUP PUSH 10 AND +
   ( a new-v )
   OVER C! 1- POP NEXT  ( ABORT" underflow! ") 2DROP ;

: RIPPLE+ ( reg n pos -) FOR OVER I + DUP PUSH C@ +
  DUP 9 > WHILE 10 - POP C! 1 NEXT ELSE POP C! POP
  THEN 2DROP ;

: RIPPLE* ( reg n pos -)
  FOR OVER I + DUP PUSH C@ OVER * POP C!  NEXT DROP ( reg)
  NORMALIZE+  ;

: (9COMPLEMENT ( reg # -)
  FOR DUP I + 9 OVER C@ - SWAP C!  NEXT DROP ;
: 9COMP ( reg -) #DIGITS (9COMPLEMENT ;
: D9COMP ( dreg -) D#DIGITS (9COMPLEMENT ;

: XNEGATE ( reg -) DUP 9COMP   1 #DIGITS RIPPLE+ ;

: ADD ( reg reg -) #DIGITS FOR OVER I + C@ OVER I + DUP PUSH
  C@ + POP C!  NEXT  NORMALIZE+ DROP  ;
: SUB ( a b -) ( a-b- - > a )   SWAP ( b a)
  #DIGITS FOR OVER I + C@ ( bv) OVER I +
  DUP PUSH C@ ( bv av) SWAP - POP C!  NEXT  NORMALIZE- DROP ;


: XCOMP ( reg1 reg2  -  -1|0|+1) #DIGITS COMP ;

: X< ( reg1 reg2 - f) XCOMP -1 =  ;
: X> ( reg1 reg2 - f) XCOMP  1 =  ;
: X= ( reg1 reg2 - f) XCOMP  0=   ;
: X0< ( reg1 - f) C@ 4 SWAP U<  ;

: LEFT-JUST ( reg1 # - #places-shifted) PUSH 1+ DUP POP
  1- 0 -LEADING= ( a+1 a+3 5) PUSH
  ( a+1 a+3) 2DUP ( a+1 a+3 a+1 a+3)
  SWAP R@ ( a+1 a+3 a+3 a+1 5) CMOVE ( a+1 a+3) OVER -
  ( a+1 #) POP +UNDER ( a+1+5 #) DUP PUSH 0 FILL POP  ;

: S->R ( n reg -) SWAP ( reg n)
  <# #FRACS FOR '0 HOLD NEXT PT HOLD # #S  DROP DUP PAD C!
  PAD 1+ SWAP FOR SWAP OVER C! 1+ NEXT DROP PAD SWAP X!  ;


: DREG/10 ( dreg -)
   DUP DUP 1+  D#DIGITS-1 CMOVE> ( dreg) 0 SWAP C! ;

: MUL-STEP ( reg dreg -)
  ( reg dreg)  DUP #DIGITS + C@ ( ie multiplier) ?DUP IF
  #DIGITS FOR PUSH OVER C@ R@ * OVER C@ + OVER C!
   -1 +UNDER 1- POP NEXT SWAP 1+ NORMALIZE+ THEN 2DROP  ;

: MUL+ ( reg1 reg2 dreg -) ( accumulating)  DUP PUSH
  SWAP OVER ( reg1 dreg reg2 dreg)
  #DIGITS DUP +UNDER CMOVE ( reg1 dreg)
  #DIGITS-1 +UNDER  #DIGITS-1 + ( reg dreg)
  #DIGITS FOR 2DUP MUL-STEP POP R@ DREG/10 PUSH NEXT
  2DROP POP
  #FRACS FOR ( dreg) DUP DREG/10 NEXT DROP  ;

: MUL ( reg1 reg2 dreg -) ( non-accumulating)
  DUP DZERO MUL+ ;

 N: DIVISOR    N: TEMP1
DN: DIVIDEND  DN: DTEMP  DN: QUOTIENT

: x10* ( a # -)
  ( shift left 1 place; force sign-digit to zero)
  2DUP OVER DUP 1 +UNDER ROT CMOVE ( a # a)
  + 0 SWAP C!  DROP  ;
: REG10* ( reg -) DUP #DIGITS-1 x10*  0 SWAP C! ;
: DREG10* ( dreg -) DUP D#DIGITS-1 x10* 0 SWAP C! ;
: SDREG10* ( dreg -) D#DIGITS-1 x10* ;

: .REGS ( -)  DIVIDEND .DX ."  by " DIVISOR .X
  ."  -> " QUOTIENT .DX ?SCROLL CR  ;


( division      divide r1 by r2 and put quotient into dreg)
(   both operands must be non-negative )
: DIV-INIT ( r1 r2 - #divisor-shifts  #dividend-shifts)
  DIVISOR REG! DIVIDEND DUP DZERO REG! (   )
  DIVISOR #DIGITS LEFT-JUST DIVIDEND #DIGITS LEFT-JUST
  QUOTIENT DZERO  ;

: DIV-STEP ( -) DIVISOR DTEMP REG! DTEMP ( reg)
  DIVIDEND DUP 1+ C@ SWAP C@ 10 * +  DIVISOR 1+ C@ U/
 ( reg tquot)
  DUP QUOTIENT DUP DREG10*  D#DIGITS-1 + C! ( reg tquot)
  #DIGITS RIPPLE* ( ie multiply divisor & trial quotient)
  (   )  DIVIDEND DTEMP SUB (   )
  BEGIN DIVIDEND X0< WHILE DIVISOR DIVIDEND ADD
    QUOTIENT D#DIGITS-1 + DUP C@ 1- SWAP C!
 REPEAT  DIVIDEND SDREG10*  ;

: DIV ( dividend divisor -) DIV-INIT ( # #)
  -  #FRACS +  2 + ( for rounding)
 FOR DIV-STEP  NEXT
  QUOTIENT DUP  5 D#DIGITS RIPPLE+  DREG/10 ( round)  ;

: TEST ( $ $ -)  X X! Y X!   Y X DIV ."  -> " QUOTIENT .DX ;
```

To copy one register to another use REG! (pronounced "reg store"), e.g.

```
X  Y  REG!          ( copy X to Y, leaving X unchanged)
```

The words ZERO and DZERO zero out a register or double register, e.g.

```
X ZERO
AA DZERO
```

Comparison is accomplished by X<, X>, X=, and X0<, e.g.

```
X  Y  X<       returns true if X is less than Y
X  Y  X>       returns true if X is greater than Y
X  Y  X=       returns true if X is equal to Y
X     X0<      returns true if X is less than zero
```

Printing registers is accomplished by .X and .DX, e.g.

```
X .X    prints the contents of register X
```

QUOTIENT .DX prints the contents of double register QUOTIENT

How To Read The Source Code

In the stack comments a dollar sign represents an address of a string, where the first byte contains a count and the following bytes contain the string. Pronounce $ as "string."

FOR ... NEXT must be preceded by a <count>. In Pygmy, the loop will be done <count> times. Within the loop, I is the down-counting index. For example,

```
: TST  10 FOR  I .  NEXT  ;
TST
```

will print

```
9 8 7 6 5 4 3 2 1 0
```

Possible Enhancements

The names should be improved. The clarity of the code and comments should be improved. Additional words are needed to drop non-significant leading and trailing zeros when printing registers. Furthermore, these large numbers are kept in fixed locations. This is still very useful, but it might be more Forth-like if they were kept on a stack.

References

I was encouraged in doing this after seeing John Wavrick's polynomial math package; after all, a base 10 number *is* a polynomial, i.e. a(10^0) + b(10^1) + c(10^2) + d(10^3) + ... + z(10^25), if we keep 26 digits.

What's Next?

I titled this Part 1 in hopes there might be a Part 2 sometime. Perhaps now that we can handle large numbers even on a little micro we might do something useful with them. Suggestions and very gentle criticisms are welcome.●

Virtual BIOS, from page 29

It didn't improve things. In fact, it decreased performance. In retrospect, the reason is obvious: it was *not* a good mapping scheme for my system because the tracks on my Ampro do not contain an even number of allocation blocks.

Double sided 96 TPI disks on the Ampro have 40 sectors per track (5k bytes/track). The allocation block for those disks is 2k. Thus, the last block on all even tracks is completed on the next odd track. Normally, the 'next' track *is* actually the next track (or the other head on the same track). This is the case, too, with the "first half reversed" system—with the single exception of the two tracks on opposite sides of the half-way point. Not too much time is spent getting to an adjacent track.

With the "even in—odd out" scheme, the "next" track is on the other "side" of the disk—and gets further away as the disk fills. To make things worse, after the "split" allocation block is read or written, a trip to the directory is required to find the next allocation block. Thus, the scheme actually causes *more* time to be spent stepping the heads around the disk! The obvious solution would be to map tracks in "pairs"—two tracks inward, two outward, et cetera.

Each system is different in it's use of the floppies. As I mentioned above, different schemes exist for how tracks are numbered on two-sided disks. Different systems also use different physical sector sizes, resulting in differing number of CP/M 128 byte sectors per track. With proper attention to a machine's disk system architecture, the mapping scheme can be quickly fine-tuned for a system. Variations of the "odd in—even out" scheme appropriate for the host machine should produce noticeable improvement over what I've presented here.

PARTING SHOTS

What I've presented here is a NZCOM Virtual BIOS solution that will produce a noticeable disk system performance increase on any machine. This specific mapping scheme has worked well for me in both sequential disk operations (system backups) and random operations (normal use). Running time for application programs that do a great deal of disk accesses—both sequential and random—can be greatly reduced when their data files are on disks with remapping enabled (try running FATCAT with the data files on remapped disks!)

Remapping can be fine-tuned for a specific system. What I've presented here will serve as a basis for system-specific schemes.

As a companion to this article, I've prepared a library named SWBIO10.LBR which contains ready-to-load ZRL versions of the virtual BIOS presented above, a compiled version of the SWAPVEC manipulation utility, and a utility to initialize disk directories. This file is likely to be available on your local Z-Node. If it isn't, you can get copy from my system, *The Elephant's Graveyard,* at 619-270-3184.●

YASMEM

(Yet Another Static Memory Expansion Module)

By Paul Chidley

Before I get into the Yasmem board let me update you on the Yasbec (Yasbec is the *Yet Another Single Board Eight-bit Computer* discussed in *TCJ* issues 51 and 52, "When eight bits is enough"). The monitor ROM for the Yasbec has been finished and running bug free for some time now. Wayne did an excellent job on the ROM and the documentation, the ROM is well suited to running the Z180 without any formal operating system. A non-banked BIOS starter disk is available from Cam Cotrill (see *TCJ* 54 title?) and the banked version is under testing and may be ready by the time this is in print. I had a limited number of Eurocard backplanes for the Yasbec cards and can have more made if interest warrants it. We have a strong Yasbec topic going in the CP/M section of GEnie, over 140 messages since July 1990, as well as the occasional Yasbec Round Table (real time conference) night, so if you want the most up-to-date information I suggest looking into GEnie. It is the only place I regularly check my Email, usually every day. Anyway on to the Yasmem.

The Yasbec CPU card supports up to 1 Megabyte of SRAM on board, so why would anyone need a memory expansion board? Well the Yasbec has two 32 pin sockets for RAM so two 512K x 8 SRAM chips would provide the 1MB but monolithic versions of this chip do not commercially exist yet. There are modules available, at the time I wrote this article a 512K x 8 SRAM module, 32 pin 600 mil dip, 100ns, commercial temperature range, cost around $185 US in small quantities. On the other hand the 128K x 8 SRAM chips are around $22 so the 512Kx8 module is still twice the price of four 128Kx8 chips. This means that eight 128Kx8 SRAM chips and a memory expansion board are currently a cost effective solution until the 512Kx8 monolithic chips become a reality. The Yasmem was designed to provide my prototype system the full 1MB of SRAM and to test running programs in external RAM across the I/O connector and backplane. With eight 32 pin dip sockets on the Eurocard there was lots of space for a battery backup controller and two lithium batteries. *Figure 1* shows the general layout of the board.

Figure 2 shows a partial schematic of the Yasmem board, the SRAMs were left off to save space. The backplane signals are fully buffered by U9, U10, U11 and U12. U12 driving the bi-directional data bus is a 74act245 while the other three buffers can be 74act245 or 74act541. The 541 is the logical choice for uni-directional signals but since a 245 is required for the data bus these three buffers can optionally also be 74act245 by changing the R1 and R2 jumpers as indicated. The buffered address and data lines go directly to the SRAMs as required. Address lines A15-A19 and the control lines go to the 16L8 pal U13. Using a pal gives the flexibility to change the hardware address decoding as required depending on how the Yasmem is to be configured in your system. The pal does not do absolute address decoding, instead it decodes the desired board address down to three lines, these lines are then passed to the DS1211 (U14) which does a 1-of-8 decode to get the individual chip selects. In other words, when using 128Kx8 chips the address lines A17 through A19 are passed straight through but should you desire to make a 256K SRAM board using 32Kx8 chips only a new pal is needed.

The Dallas Semiconductor DS1211 is used on the Yasmem to give us battery backup control of the SRAMs. The features listed in the data sheet are;

- Converts full CMOS SRAMs into nonvolatile memories
- Unconditionally write protects when Vcc is out of tolerance
- Automatically switches to battery when power fail occurs
- 3 to 8 decoder provides control for up to eight CMOS SRAMs
- Consumes less than 100nA of battery current
- tests battery condition on power-up
- Provides for redundant batteries
- "Powerfail" signal can be used to interrupt processor on power failure
- Optional 5% or 10% power fail detection
- Available in 20 pin dip (DS1211) or 20 pin SOIC (DS1211S)

The DS1211 provides a 3 to 8 decoder to provide an active low chip select signal to one of 8 SRAMs. Input voltage (Vcci) is monitored by comparing it to 4.75V or 4.5V depending on the level of the TOL pin (tolerance GND=4.75 Vcc=4.5). If Vcci drops below the tolerance level then write protection is accomplished by ensuring that all chip select outputs are held high to the SRAMs' Vcc (Vcco). Vcco to the SRAM chips is switched from Vcci or the battery supply depending on which is greater. This provides constant power to the SRAM since the switch is transparent and with only a 0.2V drop. An internal switch also selects the battery voltage input that is greater, should a battery fail voltage input is transparently switched to the other battery. Battery voltage is also tested on power up, if the battery voltage is less than 2.0 volts then the SECOND access to memory is inhibited. A simply battery test can by done by the processor by reading a memory location, writing a different value to that memory location and then reading that memory location again. If the values read are equal then the battery voltage is too low since the write cycle was inhibited. This makes a great bug for people that forget about this feature and have a dead battery. If using one battery the other input must be grounded.

The flip flops of U15 provide an optional Wait state on opcode fetches, something that is not support on the Yasbec. The Z180 has programmable wait states so why add hardware to do it here? Well I'm thinking of the day I install my 20MHz Z180 and 45nS expensive SRAMs. The Yasbec can have its fast CPU and fast SRAM running full speed with no wait states but the Yasmem will add one if its slower SRAM is accessed. So someday this will prove useful, for now I jumper mine to disabled so everything runs at full speed. (no need to slow down to access our bus) (yet)

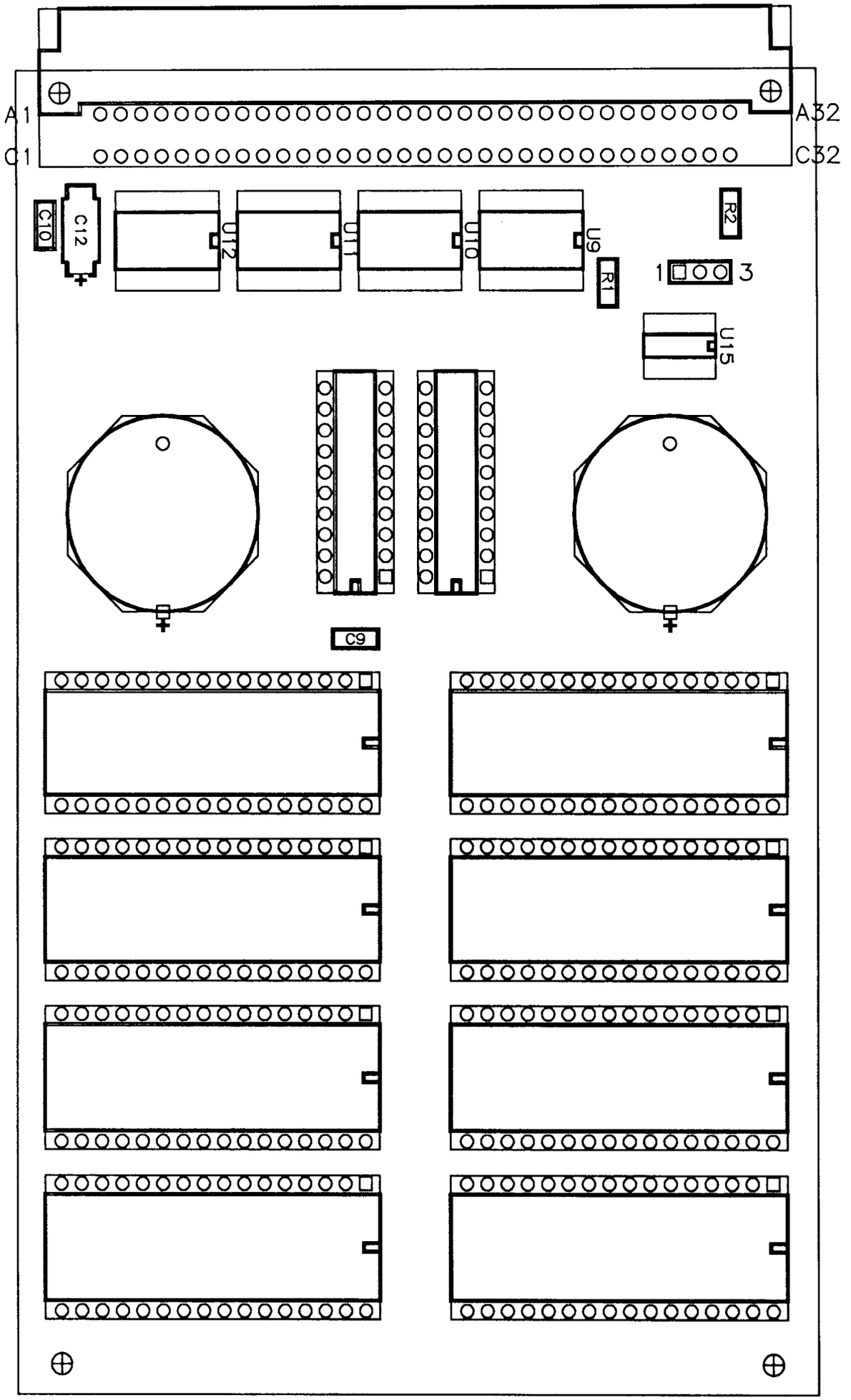


Figure 1: YASMEM board layout

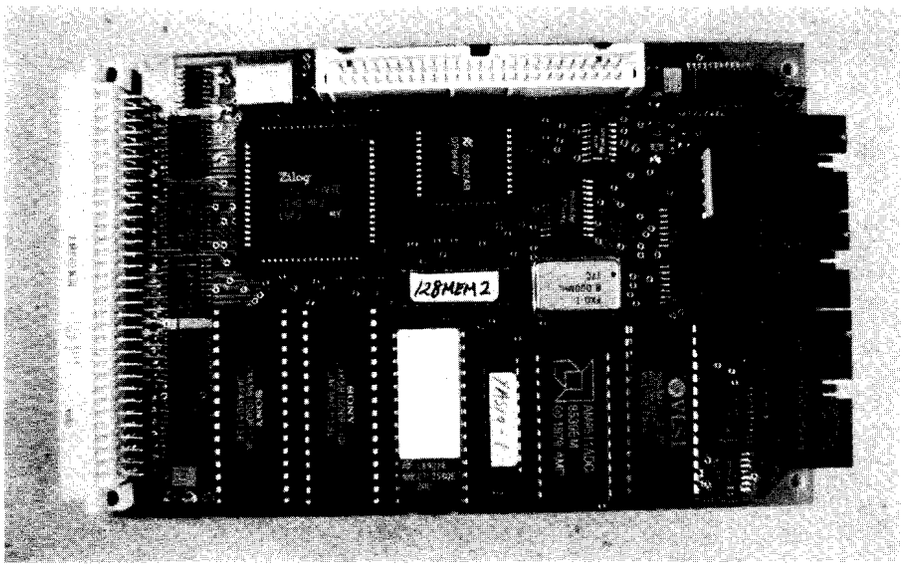


Photo 1: The YASBEC motherboard. This and all other YASBEC boards may be mounted on the Eurocard backplane.

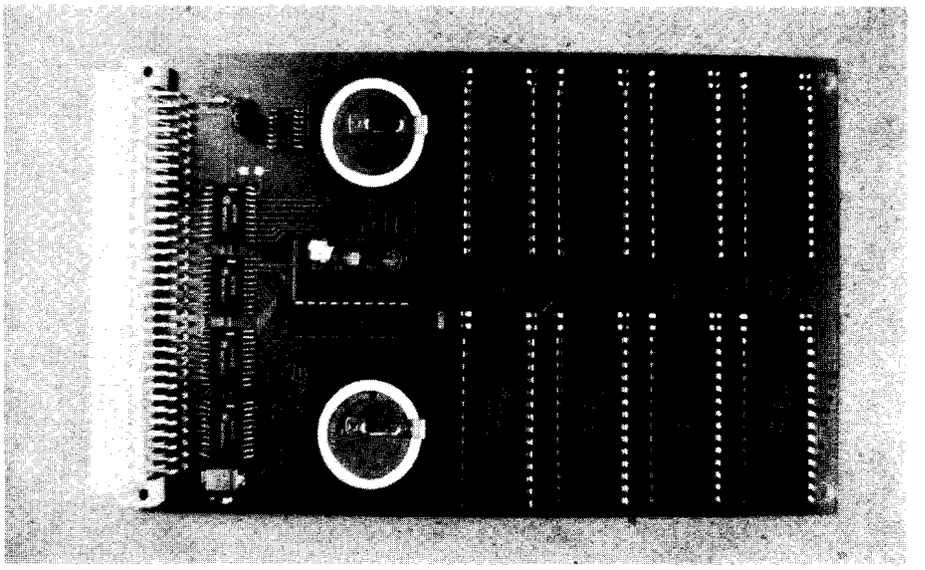


Photo 2: The YASMEM memory expansion board.

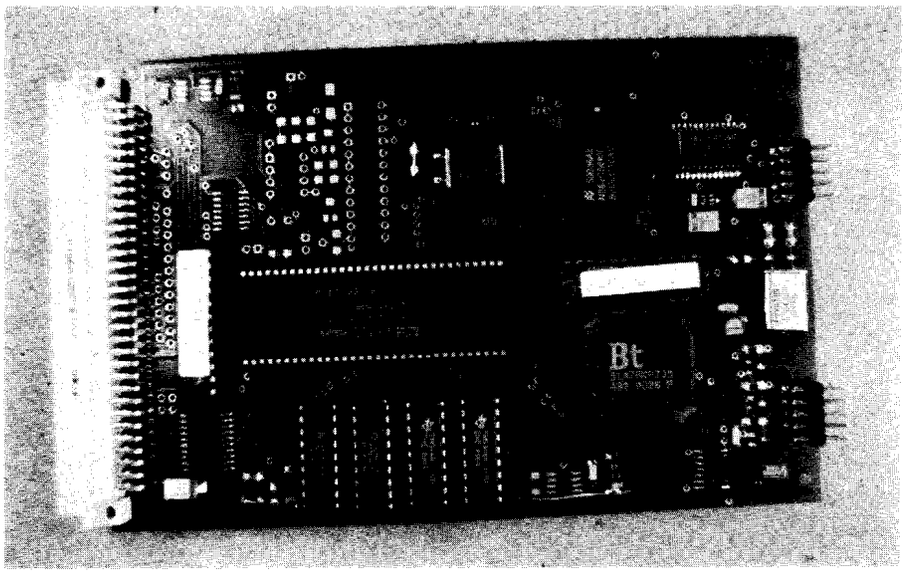


Photo 3: YASBEC color graphics board.

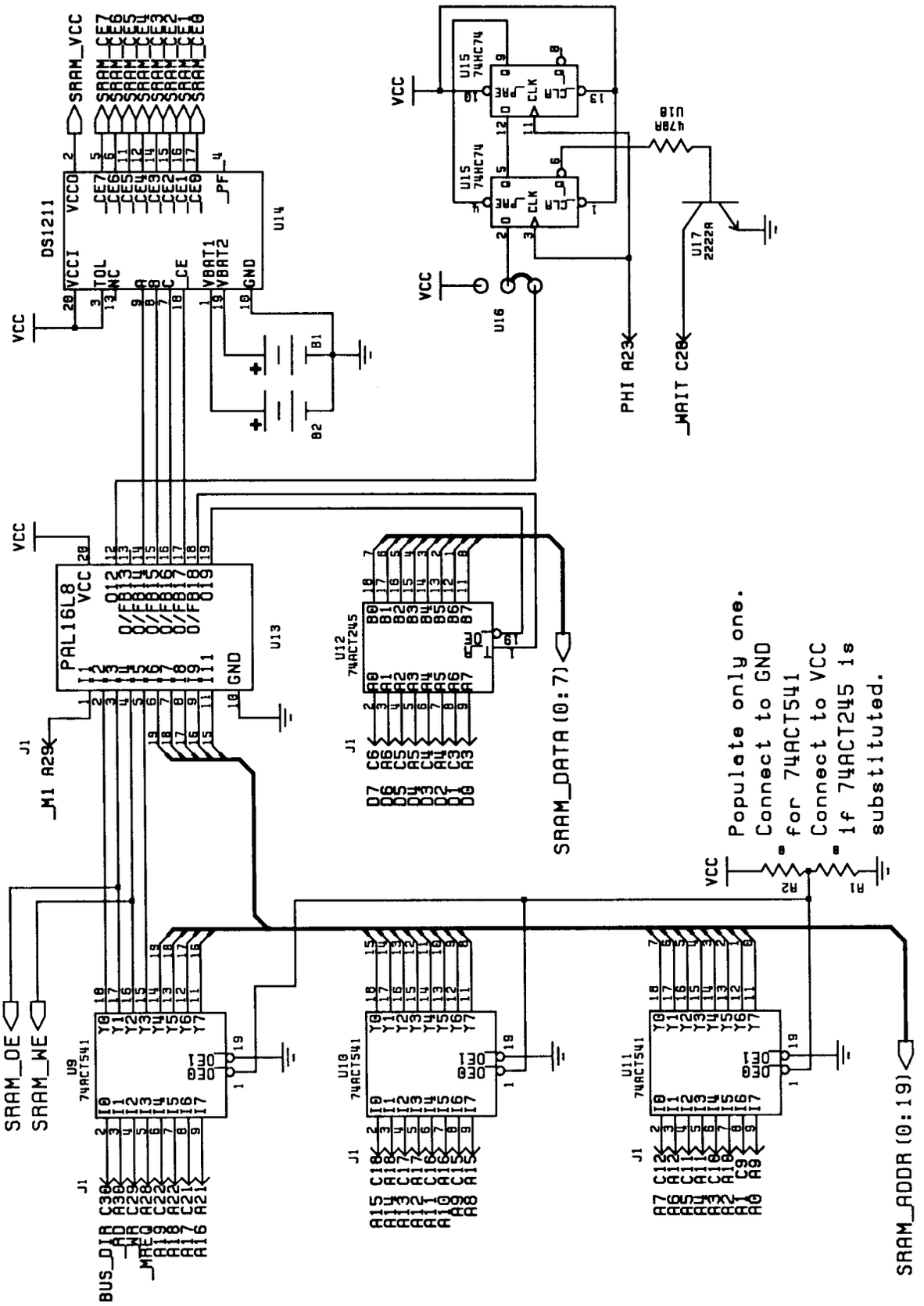


Figure 2: YASMEM Schematic Diagram

Now that we have an external memory board we must configure memory in our Yasbec system. Like many things there are many ways to do this so here's my suggestion but I'm sure people will dream up other ways to do it. Use up to seven 128Kx8 chips on the Yasmem leaving the first 128K of physical space empty on the Yasmem. This would provide for a maximum of 896K of battery backed up memory for a RAM disk. On the Yasbec CPU the first 32K of physical memory is occupied by the monitor ROM. If using 32Kx8 chips, two would provide the 64K of system RAM needed for a non-banked system. If using a single 128Kx8 SRAM 32K would be missing due to the overlap with the monitor ROM leaving 96K free for a banked system.

I hope Yasbec users and others have found this article interesting. There are a good number of Yasbec systems out there up and running and installing their software. Enough people that a few of them should take the time to write an article for *TCJ*! You know, "Yasbec from the Outside World", "How I built my Yasbec", or the now infamous "How I let the Smoke Out". I'm sure inquiring *TCJ* minds want to know.

As before, contact me by voice, GEmail, postal snail, et cetera, if you are interested in the Yasmem and/or Yasbec boards. I'm always available to answer questions but I am on Calgary (MST) time so give me a chance to get home from work before you call and be warned, if you wake me up it will be a very short answer.●

Paul Chidley
162 Hunterhorn Dr. N.E.
Calgary, Alberta
Canada T2K 6H5
Voice (403) 274-8891 MST
GEmail P.CHIDLEY

Cam Cotrill
1935 Manhattan Ave.
La Crescenta, CA 91214
Voice (eves/wkends) (818) 248-0553

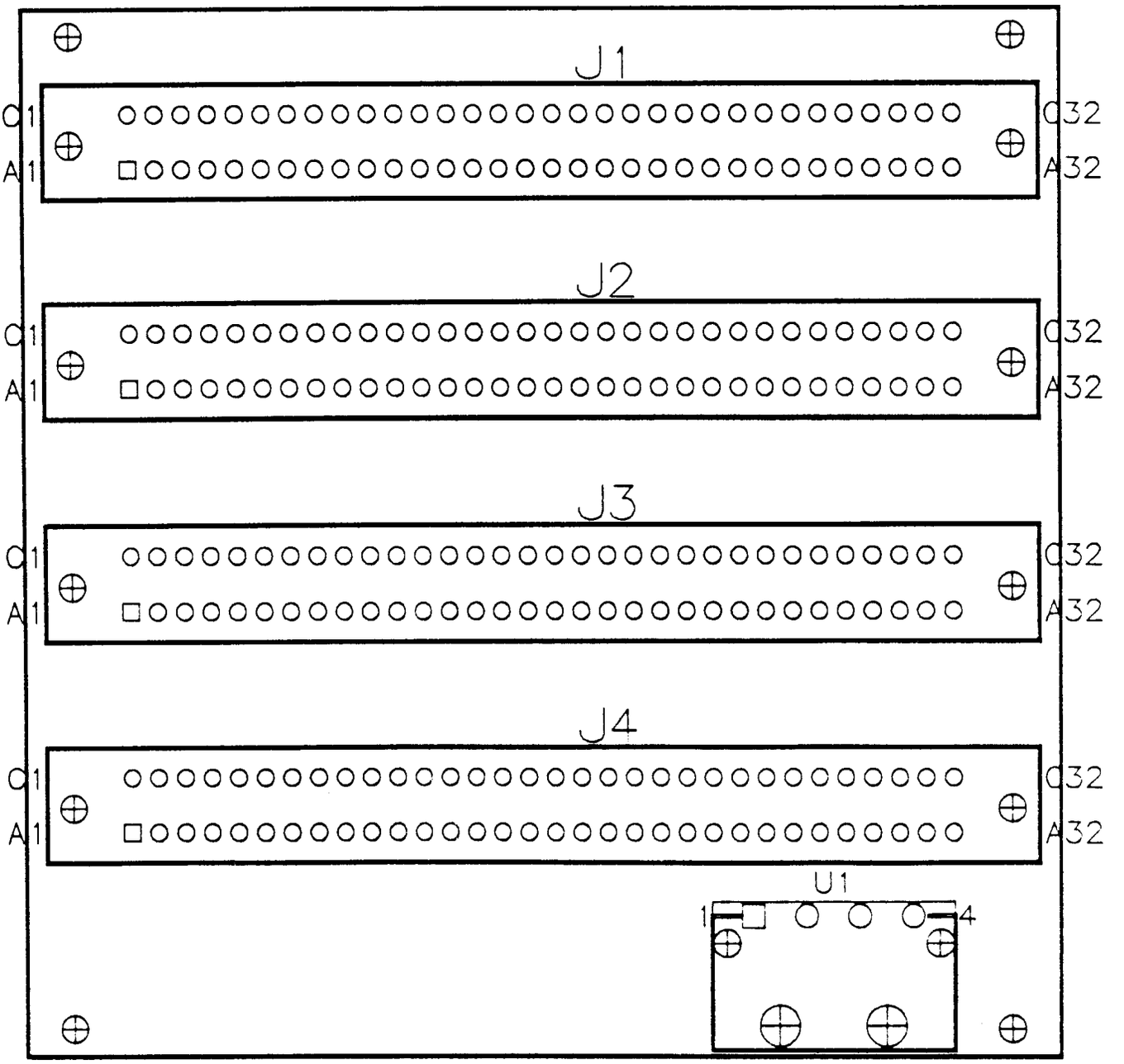


Figure 3: EuroCard backplane

TCJ The Computer Journal Market Place

Discover

The Z-Letter

The Z-Letter is the only monthly publication for CP/M and Z-System. Eagle computer and SpellBinder support. Licensed CP/M distributor.

Subscriptions: $15 US, $18 Canada and Mexico, $45 Overseas

Write or call for free sample.

The Z-Letter
Lambda Software Publishing
720 South Second Street
San Jose CA 95112-5820
(408) 293-5176

Advent Kaypro Upgrades

TurboROM. Allows flexible configuration of your entire system, read/write additional formats and more. $35

Hard drive conversion kit. Includes interface, controller, TurboROM, software and manual—Everything needed to install a hard drive except the cable and drive! $175 without clock, $200 with clock.

Personality Decoder Board. Run more than two drives, use quad density drives when used with TurboROM. $25

Limited Stock — Subject to prior sale

Call 916-483-0312 eves weekends or write Chuck Stafford, 4000 Norris Avenue, Sacramento CA 95821

TCJ The Computer Journal Market Place Advertising for Small Business

Looking for a way to get your message across? Advertise in the Market Place!

First Insertion:	$50
Reinsertions:	$35

Rates include typesetting. Payment must accompany order. Visa, MasterCard, Discover, Diner's Club, Carte Blanche, JCB, EuroCard accepted. Checks, money orders must be in US funds drawn on a US bank. Resetting of ad constitutes a new advertisement at first insertion rate. Inquire for rates for larger ads if required. Deadline is eight weeks prior to publication date. Mail to:

The Computer Journal
Market Place
PO Box 12
S. Plainfield NJ 07080-0012 USA

CP/M SOFTWARE

100 page Public Domain Catalog, $8.50 plus $1.50 shipping and handling. New Digital Research CP/M 2.2 manual, $19.95 plus $3.00 shipping and handling. Also, MS/PC-DOS Software. Disk Copying, including AMSTRAD. Send self addressed, stamped envelope for free Flyer, Catalog $1.00

Elliam Associates
Box 2664
Atascadero, CA 93423
805-466-8440

Kenmore ZTime-1 Real Time Clocks

Assembled and Tested with
90 Day Warranty
Includes Software

$79.95

Send check or money order to
Chris McEwen
PO Box 12
South Plainfield, NJ 07080
(allow 4-6 weeks for delivery)

Z-System Software Update Service

Provides Z-System public domain software by mail.

Regular Subscription Service
Z3COM Package of over 1.5 MB of COM files
Z3HELP Package with over 1.3 MB of online documentation
Z-SUS Programmers Pack, 8 disks full
Z-SUS Word Processing Toolkit
And More!

For catalog on disk, send $2.00 ($4.00 outside North America) and your computer format to:

Sage Microsystems East
1435 Centre Street
Newton Centre MA 02159-2469

Major Upgrade Announcement Z3Help and Z3COM Packages Write for Details

Major Upgrade Announcement Z3Help and Z3COM Packages Write for Details

ZBest Software

By Bill Tishey

ZSUS News

Z3LBR Package: This package was completed last October and is the third in a series of "toolkits" which I have planned to organize the many Z utilities according to function. The first was Z3PROG (The Programmer's Toolkit) and the second was Z3WORD (The Z-System Text/Word Processing Toolkit). These packages are not just a dump of every utility available, but a careful selection (with advice from others) of the "latest and greatest" of tools in each category, including not only Z utilities, but (still essential!) CP/M programs. How do I decide what to include? Well, I have a few simple guidelines. Programs are generally *not* included if their functions are felt to be duplicated or improved upon in more recent or "modernized" tools. Tools with the same function *are* included if one is felt to have some unique, valuable feature. Some programs which perform the same function, however, are worth including simply because they are styled a bit differently (it's good sometimes to have alternatives). Let's look at Z3LBR.

Z3LBR contains Z-System and general CP/M programs useful in library and archive operations. It consists of three general groupings:

- LBR-member handling utilities (directory listers, file finders, typers, extractors, etc.)
- utilities for handling ARC, ARK, ZIP, and ZOO files
- major library shells and miscellaneous programs.

See *Figure 1* for a list of the programs selected. Those with an asterisk are general CP/M programs.

Figure 1

Group #1			Group #2			Group #3		
CL10	.LBR	25k	ARC-FILE.IZF*		5k	LU310	.LBR*	47k
LBREXT34	.LBR	37k	ARC20	.ARK	69k	LUSH12	.LBR	24k
LBRHLP20	.LBR	45k	ARC20COM	.LBR*	24k	LUDEF	.DOC*	8k
LDIRB22	.LBR	25k	ARK11	.ARK	16k	LX22	.LBR	48k
LFIND	.LBR*	6k	BOOZ4CPM	.ARK*	34k	NULU152A	.LBR*	55k
LFINDPAT	.LBR	2k	UNARCZ13	.LBR	86k	VLU108	.LBR	46k
LGET13	.LBR	5k	UNZIP099	.LBR*	16k	ZLUX27	.LBR	74k
LLF11	.LBR	10k	UNZIP15	.LBR	23k			
LLF12PAT	.LBR	2k	ZIP-APP	.LBR*	7k			
LPUT22	.LBR	35k	ZIPDIR12	.LBR	21k			
LREPAIR	.LBR	9k						
LT30	.LBR*	56k						
ZLT15	.LBR	39k						

Comments: This package contains plenty of tools and documentation for anyone interested in library/archive management. You'll note that LLF is included along with LDIRB. This is because, as we noted in *TCJ* 51, LLF has the added ability to report member-file CRCs and member-file indices (both LDIRB and LLF, however, have been outdone by a new tool, LD—see the review below). Among unZIPpers, UNZIP099 is included along with UNZIP15 because of its unique ability to extract from SFX (self-extracting) ZIP files. And, while ZLT15 is the preferred library "typer", LT30 (now LT31) is also included for its ability to extract a file to a specified DU. VLU108 is included even though many complain of its limitations and, in particular, problems in preserving datestamps. The development of VLU was never fully completed and, depending on one's system, it may or may not work reliably as a LBR shell. Still, there are those who like its ZFILER-like interface and other features (for example, its handy way of making CRUNCH comments). For building LBRs, LPUT is a more reliable tool, is lightning fast, versatile and command-line oriented. LPUT and LBREXT, used either from the command line or in a combination of ZFILER macros and ARUNZ scripts, are probably the safest Z-System tools to use in building and dissolving LBR files. Some programs *not* included in the package are: ZLDIR, EXL12, TYPELZ22, LRUNZ302, LCRC, and UNARC16. These were felt to be superseded by, respectively, LDIRB, LBREXT34, ZLT15, LX22, LREPAIR, and UNARCZ13.

Other "toolkits" which I hope to make available in the future are ones dedicated to "file", "system", "disk/directory", and "date/time" management. If readers have other ideas, I'd be happy to hear them.

Z3COM Package Update: The ZSUS set of program executables (.COM files) was updated in November to include the latest versions of ZCPR3 and Z-System programs. This package now contains 641 files, totaling 2,696k and is bundled on 10 5.25 DSDD disks for $50. Configuration files (.CFG), if available, are included in a separate CONFIG.LBR for convenient usage with Al Hawley's new ZCNFG tool which recognizes .CFG files from within LBRs. The package also contains vital CP/

Bill Tishey has been a ZCPR user since 1985 when he found the right combination of ZCPR2 and Microsoft's Softcard CP/M for his three-year-old Apple II+. After graduating to ZCPR30 and PCPI's Applicard CP/M, he did a "manual install" of ZCPR3.3 (with help from a lot of friends!), and in late 1988 switched to NZCOM and ZSDOS, all on the same vintage Apple II+. Bill is the author of the Z3HELP system, a monthly-updated system of help files for Z-System programs, as well as comprehensive listings of available Z-System software and is the editor of the Z-System Software Update Service.

Bill may be contacted on CompuServe (76320,22), GEnie (WATISHE), on Jay Sage's Z-Node #3 (617-965-7259) and by regular mail at 8335 Dubbs Drive, Severn MD 21144

M programs such as CRUNCH, UNCR, and NULU.

Z3HELP System Update: I went on a blitz during the first part of November to update the Z3HELP system and by mid-month managed to bring it current with existing releases. Of course, I'm now several months behind again, but I always look forward to examining the new creations. Over 100 files were modified and an additional 500k was added to the last release (04/18/91) of the system. The only new format change is the use of the "y" LBR now for help for all the Libraries and related programming modules and routines. Users can update their current setups by picking up the update LBRs (Z3HELP43, 43b, 43c, 43d) on various Z-Nodes. Since these are over 400k, however, you might find it easier to acquire the entire system anew. Z3HELP sells for $30 (6 disks, now totaling 1,700k). Current ZSUS subscribers and Z-Node SYSOPs get it for half the price ($15). For those not interested in source code, who want the complete complement of Z-System software, I think that the combination of the Z3COM package and the Z3HELP package (for on-line documentation), is an excellent alternative to downloading or purchasing the many individual Z-System distribution LBRs.

Figure #2

```
        E(Edit ZCM) R(Recalculate ZCM) W(Write File) ESC/Q/X Quit ?

CBIOS                   CBIOS    (F106)   30 Records                 0F00h
  User Memory Area      UMA       EC00  ( 10)Records                 0500h
External Stack          EXTSTK   (EBD0)                              0030h
Multiple Command Line    Z3CL    (EB00) (203)Bytes                   00D0h
Wheel Byte              Z3WHL    (EAFF)                              0001h
External Path           EXPATH   (EAF4) (  5)elements                000Bh
External FCB            EXTFCB   (EAD0)                              0024h
Message Buffer          Z3MSG    (EA80)                              0050h
Shell Stack             SHSTK    (EA00) (  4) (32)Byte Entries       0080h
Environment Descriptor  Z3ENV    (E900) (  2)Records                 0100h
Named Directory         Z3NDIR   (E800) ( 14)Names                   0100h
Flow Control Package    FCP      (E600) (  4)Records                 0200h
Resident Command Pkg    RCP      (DE00) ( 16)Records                 0800h
NZCOM BIOS              BIO      (DD00) (  2)Records                 0100h
Disk Operating System   DOS       CF00  ( 28)Records                 0E00h
Command Processor       CCP       C700  ( 16)Records                 0800h
Input/Output Package    IOP      (0000) (  0)Records                 0000h

                  Effective TPA:    49.50k
```

The Value of User Input

The team of Z-System developers works hard to refine and enhance the Z utilities we use. A lot of thought is given to any changes suggested for a program and much care is taken when any improvements are implemented. Upgrades are released usually after consultation with other authors and after much testing for the problems which can surface when programs are modified. The developers, however, can't uncover every bug and can't foresee every situation in which a program will be used. They also must rely on users' input. Here is what Jay Sage had to say recently to one programmer who seemed somewhat discouraged over an undetected problem:

"Actually, I think the reason why no one else had reported the problem before is low expectations, but not with respect to you, but with respect to themselves. Many users, when they experience a problem, seem to assume that it is their fault. They think that we programmers are too good to make mistakes! I have found some pretty serious bugs in some of my own programs that could not possibly have gone unnoticed by the entire user community. What those users don't realize, I think, is how helpful they could be by pointing out not only full-fledged bugs but also features they don't like."

Enough said? The Z team values your comments and suggestions. They take pride in their work and like to know that their programs are meeting your and their expectations!

Figure #3

```
B0:MODEM>fl //

FL   Version 1.0
Usage:
   FL {{dir:}{afn} {...}} {-afn} {/options}
Matches files that don't match negative (-) filespec.
DIR specs after the first are ignored.
Options:
   O   omit output to file FILELIST
   D   include DU's in FILELIST
   S   include system files
   P   don't page display
   Q   quiet mode

B0:MODEM>fl d15:*.lbr

File list from D15:
a       .lbr | b       .lbr | c       .lbr | d       .lbr | e       .lbr
f       .lbr | g       .lbr | h       .lbr | i       .lbr | j       .lbr
k       .lbr | l       .lbr | m       .lbr | n       .lbr | o       .lbr
p       .lbr | q       .lbr | r       .lbr | s       .lbr | t       .lbr
u       .lbr | v       .lbr | w       .lbr | x       .lbr | y       .lbr
z       .lbr | z3help44.lbr
   27 matching files found on D15:

B0:MODEM>fl d15:*.* -*.lbr

File list from D15:
-read   .me  | -z3help .doc | -z3help .lst | -z3help .new | -z3hlp44.doc
filelist.    | hdr     .    | hdr2    .    | mast    .cat
    9 matching files found on D15:
```

New Releases and Updates

Since I missed the last issue (I apologize for that), I have four months to make up in discussing Z-System program development! *Listings 1 and* 2 I think prove that there were warm fingers on at least some keyboards this winter. Thanks go to the editor for giving me the space this issue to bring you up to date.

Terry Hazen: EDZCM10, ENVCFG12, JTHLIB12, DATEFN11, DD19, ENVSRC12, NZTIM12, PRNTXT16, REMIND15, RENAMZ19

Terry is famous for his fine line of system utilities which operate in both ZCPR3 and CP/M environments (ACOPY, RENAMZ, ERAZ, UNERAZ, DD). He has also released some interesting tools which help to simplify making changes to the Z-System environment. (Note his article on IOPLDR in *TCJ* 53).

EDZCM10 (ZSUS V3 #2) is a screen-oriented editor for NZCOM ".ZCM"

Figure #4

```
B0:MODEM>ld //

LD   Version 1.1
Usage:
   LD {dir:}lbrname {afn} {{/}options}
Options:
   C   display embedded comments
   X   alternate display: CRC's and indexes
   S   show summary only
   D   prefer embedded date stamp
   P   don't page screen
   Q   quiet mode
   M   put matching and free entries in registers 18-21
   L   echo to printer
   F   send final form feed
Option X overrides embedded comment display.

B0:MODEM>ld f0:ldl1
à
Library: F0:LD11     .LBR   Created: 10/25/91 21:13   Modified: 10/25/91 21:14
Member Name        Size      Mth     Created          Modified         Real Name
============   ========= ===  ==============  ==============  ============
LD11     .CZG    29r   3.62k CR   10/20/91 17:40  10/20/91 17:40  >LD11.CFG
LD11     .CZM    30r   3.75k CR   10/25/91 21:10  10/25/91 21:10  >LD11.COM
LD11     .DZC    48r   6.00k CR   10/08/91  0:17  10/25/91 20:37  >LD11.DOC
LD11     .FOR     4r   0.50k --   10/08/91  0:17  10/24/91 22:53
LD11     .ZZ0   151r  18.87k CR   08/20/91  2:53  10/25/91 21:10  >LD11.Z80
LDCF     .TZT    22r   2.75k CR   10/07/91  2:29  10/20/91 17:18  >LDCF.TXT
LDCF     .ZZ0    52r   6.50k CR   10/07/91  2:21  10/20/91 17:40  >LDCF.Z80
============   ========= ===  ==============  ==============  ============

LD11     .LBR members: 7 matched, 7 active, 4 free, 0 deleted, 11 total

B0:MODEM>ld f0:ldl1 /x

Library: F0:LD11     .LBR   Created: 10/25/91 21:13   Modified: 10/25/91 21:14
Member Name        Size      CRC Index | Member Name        Size      CRC Index
============   ========= ==== ==== | ============   ========= ==== ====
LD11     .CZG    29r   3.62k ECB2    3 | LD11     .CZM    30r   3.75k 580F   32
LD11     .DZC    48r   6.00k 926F   62 | LD11     .FOR     4r   0.50k EE29  110
LD11     .ZZ0   151r  18.87k 3F94  114 | LDCF     .TZT    22r   2.75k D08E  265
LDCF     .ZZ0    52r   6.50k 5147  287 |
============   ========= ==== ==== | ============   ========= ==== ====

LD11     .LBR members: 7 matched, 7 active, 4 free, 0 deleted, 11 total
```

descriptor files. More flexible than MKZCM, it allows users with unique systems to configure non-standard gaps between system elements or even to place elements in above-CBIOS locations. Its syntax is: "EDZCM [[dir:]zcmfile[.ZCM]]" which allows loading of the descriptor either from a ZCM file or directly from the Z3ENV. See *Figure* 2 for a typical screen display.

The display is sorted by system element address, the highest address first, and editable addresses and sizes are displayed in standout video (illustrated here in parentheses). The effective system TPA is displayed at the bottom. In edit mode, you edit the address fields in hex and the size fields in decimal. You can position system elements wherever you wish, even above the CBIOS, and insert 'dummy' elements to effect a desired order. On exit from edit mode, EDZCM will resort and redisplay the elements in edited address order. The 'Recalculation' command can be used to recalculate the element addresses beginning with the CBIOS and working down (so you don't have to use a calculator!). When you're satisfied with the order, addresses and sizes, you can write the new system out to a ZCM file. EDZCM10.LBR includes a sample AMPRO.ZCM file for Ampro users in which the wheel byte and path have been relocated from page 0 to addresses above the CBIOS.

ENVCFG12 (ZSUS V3 #2) is another new program which provides a way of configuring the contents of the Z3 environment through use of ZCNFG. Typical usage from an alias script is: "ENVCFG L; ZCNFG ENVCFG; ENVCFG S; GO I". This loads a copy of the current environment into ENVCFG.COM's buffers, allows configuring of the contents of those buffers, saves the modified environment to memory, then clears the contents of the buffer in ENVCFG.COM. The new Z3ENV can also be saved to a specified file (.ENV is the default filetype).

Two related tools which Terry developed some time ago are ENVSRC, which creates a commented source code file (Z3ENV.Z80) from the Z3 environment, and TCSRC, which creates a source code file (MYTERM.Z80) from a given terminal file (MYTERM.Z3T). The source code output from both of these programs can be used as a reference for patching or can be edited themselves, assembled and loaded to install changes to the environment.

Terry has also released his set of custom routines (JTHLIB12), originally developed for use in REMIND and ZP, which are modified versions of various SYSLIB, ZSLIB and VLIB routines. Vers 1.2 fixes a bug in the FNAMZ filename scanner and adds extended versions of VLIB routines GXYMSG, VPRINT and VPSTR.

Terry's other updates include:

- DATEFN11—fixes a hi-bit filtering problem with the help filename and adds JTHLIB filename display calls.
- REMIND15—the "upcoming month" display now highlights the current date and offers three options: C—display current month's upcoming reminders, A— display all upcoming reminders and N—no upcoming reminders.
- RENAMZ19—several new options include the ability to add or delete filename prefixes.
- DD19—now displays the status line on user abort as well as at normal exit; adds a printer init string.
- PRNTXT16 (ZSUS V3 #2)—now ZCNFG configurable and includes a printer status check.

Gene Pizzetta: CONCAT16, CRCBLD14, D21, DATSTP19, ECHO14, ERASE57, FA16A, FILT81, FL10, JUST13, LCS10, LD11, PRETTY31, RCOPY21, SWZ10, UMAP18, XFOR15, ZSLIB34, ZTIME14

Remember the excitement over XFOR, which we covered in depth in *TCJ* 49? Well, Gene has developed another utility based on collaboration similar to that with XFOR. While Gene was working on CPD, his "Compare Directory" utility, he noted that there was no comprehensive set of routines for reading and generating lists of disk files. None of the current directory listers, for example, can output a single-column list of files. It became evident that some generalized routines were needed for handling file lists from within programs as well as a separate directory utility that specialized in creating disk-based file lists. Well, Gene has come up with both!

Gene describes his approach to reading disk-based file lists thus: "Filelist processing is accomplished as a stream, with the end of a filename assumed when the read routine reaches a comma or a non-printable character. This is not only efficient, but allows

using lists produced by the likes of dBASE and BASIC. Multiple non-printable characters are simply skipped over. This allows certain control characters to be used as markers in the list, either to tag particular files or to mark blocks of files."

Gene makes use of the new routines in RCOPY21, CPD15, and the new FL10, and other programmers are now considering their use. Rob Friefeld's new version of ZFILER (ZF10Q) writes out tagged files to disk so that the tagging can be preserved across macro invocations. Gene's routines raise the possibility of tagging files based on datestamps, etc., without having to add considerably to ZF's code. Rob's XOX utility presently uses a complex internal file list generator and might also benefit from this being handled externally to the program. Gene is also thinking of modifying W.COM to read file lists for programs that don't otherwise handle them. He suggests that versions of CRUNCH and UNCR which handle lists might also be handy.

Now let's look at Gene's FILELIST (ZSUS V3 #3), a ZCPR3 utility which creates filelists on disk for use by other programs. See *Figure 3* for FL's usage screen and some sample displays.

FL allows multiple ambiguous file specs, including a negative specification (filespecs preceded by a tilde). The error flag is set if no matching files are found. Matching filenames are written to a disk file, in uppercase, one name per line in standard command line format ("filename.typ"). Optionally, the DU in which the file is located can be included with the filename. As is customary with Gene's utilities, a number of configuration options are available using ZCNFG: display of filenames in upper- or lowercase; output of FILELIST to either the currently logged or target directory; giving FILELIST a different name; setting the error flag and invoking the error handler on abort (to allow aborting a ZEX or SUB batch operation).

Like XFOR, FL offers a lot of possibilities for enhancement and we should see a number of future upgrades. Some sophisticated negative filespecs might be possible, for example, as well as an ability to select files by attributes or combinations of attributes.

Another of Gene's accomplishments is LD (ZSUS V3 #2), his new Library Directory tool. LD is based on LDIR-B, but is intended to be much more versatile, with numerous command line and configuration options, and making more extensive use of Z-System facilities, while remaining compatible with vanilla CP/M and Z3PLUS. See *Figure 4* for LD's help screen and several sample displays.

Listing #1 **New Releases:**

```
name          vers sys/sus   kb rec  crc library/size issued   author
-------------------------------------------------------------------------------
C128.Z3T      1.00   3 0      1   1 487B C128TCAP   2 12/05/91 Ed Flinn
  SYS     Extended TCAP for the C-128, implementing many of the extended
  HLP=N   functions, including simple ASCII charactersfor the graphics.
  CFG=N

CRCBUILD.COM 1.40   0 0      3  18 04D5 CRCBLD14  20 10/03/91 Gene Pizzetta
  SYS     Z'ified version of SIG/M's CRCBUILD.  Recognizes DU/DIR specs, takes
  HLP=N   the disk number automatically from the disk label (if one exists),
  CFG=Y   sets the error flag, and is compatible with CRCZ's CRC-checking.

EDZCM.COM     1.00   3 V302   7  54 5FS4 EDZCM10   36 11/08/91 Terry Hazen
  SYS     Screen-oriented editor for the NZCOM ZCM system descriptor file.
  HLP=Y   Displays and can edit NZCOM system elements and sizes from a ZCM file
  CFG=Y   or from the Z3ENV.

ENVCFG.COM    1.20   3 V302   2  13 710C ENVCFG12  21 11/21/91 Terry Hazen
  SYS     Used with ZCNFG and ENVCFG.CFG to configure the current contents of
  HLP=Y   the ZCPR3 environment (valid drive vector, max drive, etc.).  Buffers
  CFG=Y   the Z3ENV contents and writes them back after modifications.

FILEATTR.COM 1.6a   4 V303   4  32 8A5C FA16A      38 01/18/92 Gene Pizzetta
  FILE    Displays or sets the attributes of selected groups of files under
  HLP=Y   CP/M or any of its replacements.  Configurable with ZCNFG.
  CFG=Y   Originally distributed with ZDOS vs 1.1.

FL.COM        1.00   3 V303   4  31 C5DD FL10       24 01/19/92 Gene Pizzetta
  FILE    FL creates file lists on disk for use by other programs.  Multiple
  HLP=Y   ambiguous file specs, including a negative spec, may be given.
  CFG=Y   Matching file names are written to a disk file.

HELPCH.COM    1.00   3 V303   6  42 D5D7 HELPLZH    48 01/21/92 Howard Goldstein
  HELP    Z-System Help tool which works with LZH-encoded help files (.HYP)
  HLP=N   instead of crunched (.HZP) or squeezed (.HQP) files.  Handles
  CFG=Y   standalone files.

IOPLDR.REL    1.10   3 0      2   9 6656 IOPLDR11  14 06/07/91 Terry Hazen
  IOP     Generalized IOP loader REL module that may be linked to load and
  HLP=Y   remove a target custom NZCOM IOP module.  Designed to be called by
  CFG=Y   a user-written custom loader.

JTHLIB.REL    1.20   0 0      2   9 5C6F JTHLIB12  13 01/27/92 Terry Hazen
  PROG3   Custom routines originally developed for use in REMIND and ZP.
  HLP=N   Includes extended versions of VLIB routines GXYMSG, VPRINT, and
  CFG=N   VPSTR.
à
LCS.COM       1.00   0 0     11  86 3941 LCS10      20 10/03/91 Gene Pizzetta
  SYS     Loads character sets to Wyse 60 and Televideo 965 terminals, including
  HLP=N   a Roman font similar to that on MS-DOS machines and an easier-to-read
  CFG=Y   Courier font.  Provides also for a user-defined font.

LD.COM        1.10   0 V303   4  32 D55E LD11       45 10/25/91 Gene Pizzetta
  LBR     LBR directory utility that shows member files along with their create
  HLP=Y   and modify dates and times, compression method and uncompressed file
  CFG=Y   names.  Displays embedded comments and has optional printer output.

LHH.COM       1.00   3 V303   7  49 8863 HELPLZH    48 01/21/92 Howard Goldstein
  HELP    Z-System Help tool which works with LZH-encoded help files (.HYP)
  HLP=N   instead of crunched (.HZP) or squeezed (.HQP) files.  Handles help
  CFG=Y   files in libraries.

SWZ.COM       1.00   3 0      3  21 D4A4 SWZ10      19 11/15/91 Gene Pizzetta
  DATE    Replacement for SWX and SWTIME for SB180s running ZSDOS.  Reads the
  HLP=Y   Smartwatch chip and sets the system "heartbeat" clock via ZSDOS.
  CFG=Y   Works under both Micromint BIOS and XBIOS.
```

LD's many options set it apart from other LBR directory listers: embedded comments can be displayed on a separate line (up to 76 characters); an alternate display provides member names, sizes, CRC's, and indexes in two-column format; the number of matching members and free library directory entries can be stored in message registers; and printer output and wheel protection are supported. All this and more in less than 4k.

Gene's other updates include:

- CONCAT16 (ZSUS V3 #2)—Many changes have been made since vs

Listing #2 **Revised Programs:**

```
name          vers sys/sus   kb rec  crc library/size issued   author
-------------------------------------------------------------------------------
BCINST.COM    3.0b   0 0      5  39 F23C BCOMP30B  31 01/27/92 Rob Friefeld
  FILE   Screen-oriented, control-key installation program for BCOMP.
  HLP=Y
  CFG=N

BCOMP.COM     3.0b   0 0      9  71 4C45 BCOMP30B  31 01/27/92 Rob Friefeld
BCOMP.4OM     3.0b   4 0     12  90 35E6 BCOMP30B  31 01/27/92 Rob Friefeld
  FILE   Compares two binary files visually.  Can compare selected sections
  HLP=Y  of a file, or a file and memory.  Requires ZCPR3.0+ or Z3PLUS,
  CFG=N  79X24 CRT with EREOL, CLS, GOTOXY.

CMAZE.COM     2.40   0 0     29 229 7DA2 CMAZE24   38 09/24/91 Lee Bradley
  GAME   MBASIC/BASCOM game which uses Z-System TCAP data.  Creates a maze of
  HLP=N  walls and asks you to position a ball in the maze.  Data entry is
  CFG=N  best done with ZEX scripts.  Uses Z3BAS library of routines.

CONCAT.COM    1.60   0 V302   7  56 C5BD CONCAT16  53 11/12/91 Gene Pizzetta
  WP     Concatenates two or more source files into a destination file,
  HLP=Y  similar to PIP, or appends them to an existing file.  Accepts both
  CFG=Y  DIR and DU specs.  Checks for adequate disk space.  For ZCPR3 only.

CPD.COM       1.50   0 V303   4  32 F156 CPD15     32 01/19/92 Gene Pizzetta
  DIR    Compares two dirs and indicates which files exist in both dirs and
  HLP=Y  which exist only in the first dir.  Allows setting of archive bit on
  CFG=Y  those files which exist in both dirs.  Upper- or lower-case display.

D.COM         2.10   3 0      4  32 BA87 D21       40 11/18/91 Gene Pizzetta
  DIR    Directory utility providing:  display in lowercase, total disk
  HLP=Y  capacity, total bytes used by a file, remaining bytes on disk, etc.
  CFG=Y  Configurable with ZCNFG.  Original (1984) by Hank Blake.

DATEFN.COM    1.10   0 0      2  15 4334 DATEFN11  16 01/19/92 Terry Hazen
  DATE   ZCPR3/ZSDOS utility which reads the ZSDOS clock and creates or
  HLP=Y  optionally deletes an empty label file with the correct date as the
  CFG=Y  filename.  Filename date format configurable with ZCNFG.

DATSTP-P.COM 1.90   0 0      5  35 C417 DATSTP19  64 12/16/91 Gene Pizzetta
DATSTP-P.3OM 1.90   3 0      5  35 A97E DATSTP19  64 12/16/91 Gene Pizzetta
DATSTP-P.4OM 1.90   4 0      6  42 607A DATSTP19  64 12/16/91 Gene Pizzetta
  DATE   See DATSTP-U.COM.  For Z3PLUS only.
  HLP=Y
  CFG=Y

DATSTP-U.COM 1.90   0 0      6  41 4C1B DATSTP19  64 12/16/91 Gene Pizzetta
DATSTP-U.3OM 1.90   3 0      6  41 8B9A DATSTP19  64 12/16/91 Gene Pizzetta
DATSTP-U.4OM 1.90   4 0      6  48 48EC DATSTP19  64 12/16/91 Gene Pizzetta
  DATE   Displays or changes the create and modify date stamps on any file
  HLP=Y  from the command line.  Universal version (ZSDOS/ZDDOS/ZRDOS with
  CFG=Y  DateStamper; under Z3PLUS will display but not change date stamps).

DATSTP-Z.COM 1.90   3 0      4  27 5461 DATSTP19  64 12/16/91 Gene Pizzetta
DATSTP-Z.3OM 1.90   3 0      4  27 206A DATSTP19  64 12/16/91 Gene Pizzetta
DATSTP-Z.4OM 1.90   4 0      4  32 B69E DATSTP19  64 12/16/91 Gene Pizzetta
  DATE   See DATSTP-U.COM.  For ZSDOS and ZDDOS only.
  HLP=Y
  CFG=Y

DD.COM        1.90   3 0      4  30 1941 DD19      35 12/31/91 Terry Hazen
  DIR    Disk directory utility which can display files selected by file
  HLP=Y  attribute as well as by file mask.
  CFG=Y

ECHO.COM      1.40   0 0      1   8 8A06 ECHO14    18 10/10/91 Gene Pizzetta
ECHO.3OM      1.40   3 0      1   8 6C5F ECHO14    18 10/10/91 Gene Pizzetta
ECHO.4OM      1.40   4 0      2  12 BF64 ECHO14    18 10/10/91 Gene Pizzetta
  SYS    Allows text entered at command line to be typed to the screen
  HLP=Y  without the operating system acting on it.
  CFG=Y

ENVSRC.COM    1.20   4 0      6  41 1FFF ENVSRC12  13 11/15/91 Terry Hazen
  PROG2  Creates commented source code from a Z3 environment using the latest
  HLP=Y  extended environment definitions in NZCOM.ZCM.
  CFG=N

ERASE.COM     5.70   3 0      4  26 F876 ERASE57   36 11/14/91 Gene Pizzetta
ERASE.3OM     5.70   3 0      4  26 1EB1 ERASE57   36 11/14/91 Gene Pizzetta
ERASE.4OM     5.70   4 0      4  32 3EC7 ERASE57   36 11/14/91 Gene Pizzetta
  FILE   Transient counterpart of ERA.  Erases read/write files from command
```

1.4: CON:, LST: and AUX: can be used in place of the destination file; a new "P" option pages output to console or printer; a new "H" option resets hi-bits and removes control characters from the screen; the "I" option doesn't have to be at the end of the option list; new, more specific error messages; more configuration options; ZSLIB HV routines are now used in place of larger VLIB ones.

- CPD15 (ZSUS V3 #3)—Since version 1.0, options have been added to 1) include DU specs with filenames in the CPDLIST output file, 2) output matching filenames to a diskfile, and 3) suppress console output. More configuration options have been added, changes to the display and usage screen, bug fixes and code optimization. CPD is also much faster now that both directories to be matched are loaded into memory. D21—adds several new features; fixes some long-standing bugs; now works under vanilla CP/M.
- DATSTP19—VLIB references have been replaced with much shorter ZSLIB34 routines, keeping the executables to 6k and under.
- ECHO14—corrects a problem when running under BYE; adds matching RCPECHO, Type 3 and Type 4 versions, and ZCNFG configuration.
- ERASE57—adds a Type 4 version; the Type 3 version has been made CPM-safe; the program banner is improved; some code changes help keep the size to 4k.
- FA16A (ZSUS V3 #3)—FILEATTR is a utility to quickly set, reset, or display file attributes. It was originally distributed with ZSDOS and has since been made compatible for use under ZSDOS, ZRDOS, and CP/M 2.2. Under ZCPR3, it allows error flag setting, error handler invocation, and an enhanced display. FA16A adds much needed documentation.
- FILT81 (ZSUS V3 #2)—now checks for ambiguous filenames; the display of line count progress reports is now a configuration option.
- JUST13 (ZSUS V3 #2)—same changes as made to FILT81.
- PRETTY31—cases of labels and opcodes are now a command line option; adds a few SLR and ZMAC pseudo-ops to the table; a bug fix, some minor code tweeks.
- RCOPY21—corrects a bug in the listfile parsing routine.
- UMAP18—corrects a problem when running under BYE; other minor changes.
- XFOR15 (ZSUS V3 #2)—now prints the "searching" message only when a

search string is given, not after each entry during sequential FOR file displays.

- ZSLIB34—adds switchable output routines and makes some bug fixes.
- ZTIME14—rereads the clock after setting it; corrects some parser bugs; adds Type 3 and Type 4 versions.

Bruce Morgen: MENU42, PWD21, SAVNDR14, SUB35, Z33IF16

Bruce continues to bring some of the older, but still useful, Z-System utilities up to "modern" standards.

In SUB35, Bruce has made some ambitious improvements while still preserving backward compatibility with scripts written for SUBMIT, SUPERSUB, and previous versions of the ZCPR3 SUB series. Probably most significant is the ability to read SUB scripts from a standard LBR file. Like ZCNFG's .CFG files, .SUB files tend to be very short and quickly eat up disk space. It makes sense to bundle them into LBRs. SUB scripts can now be searched in LBRs 1) from the command line in the same manner as LX—"SUB -[DIR:]LBRNAME[.TYP] SUBNAME[.TYP]"—or 2) by searching a user-patched library after the search for a standalone SUB script has failed. SUBINST.COM is provided to patch the name and location of this secondary library. SUB35 also recognizes the UNIX-style "$*" (entire command line) parameter, supports Z34ERR12-compatible error handling, and observes the Z3ENV quiet flag.

Bruce's other updates include:

- MENU42—adds direct support for Bridger Mitchell's "AT" background timer and shrinks the tool back to its 1984 size of 4k.
- PWD21—adds a Type 4 version; improves the program banner; uses the new ZSLIB video highlighting routines in place of the larger VLIB routines.
- SAVNDR14—implements a Type 4 version; adds the Type and load address to the program banner; contains some minor code changes.
- Z33IF16—uses the new ZSLIB video highlighting calls in place of VLIB ones. Two other developments have to be mentioned before closing out this rather lengthy column:

Howard Goldstein has reworked the Z-System Help tool into a version which will read LZH-encoded help files (.HYP). HELPLZH.LBR (ZSUS V3 #3) contains HELPCH.COM, which handles standalone files, and LHH.COM, which handles files within libraries.

```
  HLP=Y  line and prompts for erasure or read-only files.
  CFG=Y  Vs. 5.0 (05/84) by Richard Conn.

FILT.COM     8.10   0 V302   4  30 0393 FILT81     24 10/03/91 Gene Pizzetta
  WP     ZCPR3 rework of Irv Hoff's FILT7 tool which sets or expands tabs and
  HLP=Y  removes several types of unwanted characters in ASCII text, WordStar
  CFG=Y  documents or assembler source code files.

HELPLSH.COM  1.1b   4 0      3  21 69CB HELPLSH     3 01/06/92 Rob Friefeld
  HELP   Help program for LSH vs 1.1.
  HLP=Y
  CFG=N

JUST.COM     1.30   0 V302   4  32 A6A9 JUST13     41 10/03/91 Gene Pizzetta
  WP     ZCPR3 rework of Irv Hoff's tool to justify ASCII and WordStar text
  HLP=Y  files.  Full command line operation, DU support, error-flag setting,
  CFG=Y  quiet mode, and transfers create datestamps under ZSDOS.

LT.COM       3.10   C 0      7  54 EE89 LT31       64 12/16/91 Brian Murphy
  LBR    Library Type can type normal, LZH-encoded, crunched or squeezed files
  HLP=N  whether standalone or in a .LBR.  Can extract/uncrunch any/all files
  CFG=N  at the same time.  Adapted from Steven Holtzclaw's LUXTYP (06/83).

MENU.COM     4.20   0 0      4  32 2437 MENU42     48 10/30/91 Bruce Morgen
  SHELL  The ZCPR3 menu front-end processor.  Shell which reads a .MNU file
  HLP=Y  and processes commands from it.
  CFG=N

NZTIM.REL    1.20   4 0      3  24 2678 NZTIM12    43 01/27/92 Terry Hazen
  PROG3  Library of Date/Time subroutines which allow data or pointers to data
  HLP=Y  to allow utilities to do whatever necessary with Date/Time.  Entirely
  CFG=N  self-contained.  Original (2/10/90) by Joe Wright.

PDAT.REL     1.20   4 0      2  12 0F01 NZTIM12    43 01/27/92 Terry Hazen
  PROG3  Library of routines which use NZTIM and SYSLIB routines to print
  HLP=Y  various forms of Date and Time.  Original (2/10/90) by Joe Wright.
  CFG=N

PRETTY.COM   3.10   3 0      5  38 5CD5 PRETTY31   32 10/03/91 Gene Pizzetta
  PROG2  Standardizes the case of opcodes and labels in Z80 and 8080 source
  HLP=Y  code.  Based on Carrol Bryan's PRETTY23.  Preserves create date
  CFG=Y  stamps under ZSDOS.  Configurable with ZCNFG.

PRNTXT.COM   1.60   0 V302   1   6 E39D PRNTXT16   26 10/05/91 Terry Hazen
  SYS    Console/printer message utility program which creates "instant"
  HLP=Y  text display .COM files to the console or printer.
  CFG=N

PWD.COM      2.10   0 0      4  25 813E PWD21      48 11/12/91 Bruce Morgen
PWD.30M      2.10   3 0      4  25 A760 PWD21      48 11/12/91 Bruce Morgen
PWD.40M      2.10   4 0      4  30 CEB4 PWD21      48 11/12/91 Bruce Morgen
  SYS    Displays the names of DU and DIR names with paging.
  HLP=Y  Vs 1.0 (03/84) by R. Conn.
  CFG=Y

RCOPY.COM    2.10   0 0      4  29 E445 RCOPY21    24 01/11/92 Gene Pizzetta
  FILE   Copies a list of files between two directories.
  HLP=Y
  CFG=Y

REMIND.COM   1.50   3 0      5  39 3730 REMIND15   38 01/26/92 Terry Hazen
  DATE   Appointment reminder utility for ZCPR3 and ZSDOS with clock.  Will
  HLP=Y  display and optionally print a sorted and paged list of dated appt
  CFG=Y  reminder lines with optional time entries from a text datafile.

RENAMZ.COM   1.90   3 0      5  33 9FD3 RENAMZ19   64 11/08/91 Terry Hazen
  FILE   Enhanced Z80 file renaming utility for CP/M 2.2 and ZCPR3 CCP's
  HLP=Y  running under CP/M 2.2, Z80DOS, ZRDOS, and ZSDOS.  Companion to
  CFG=Y  ACOPY, DD, ERAZ, and UNERAZ.  Vs 1.0 (04/87) by Terry Hazen.

SAVNDR.COM   1.40   0 0      2  12 5A49 SAVNDR14   12 11/22/91 Bruce Morgen
SAVNDR.30M   1.40   3 0      2  12 00C2 SAVNDR14   12 11/22/91 Bruce Morgen
SAVNDR.40M   1.40   4 0      2  16 1ED9 SAVNDR14   12 11/22/91 Bruce Morgen
  NDR    Writes a reloadable (by LDR) copy of the resident Named Directory
  HLP=Y  module to a disk file.
  CFG=N

SCOPY.COM    0.80   4 0     15 120 D6DB SCOPY08    26 10/21/91 Rob Friefeld
  FILE   Screen-oriented file copy utility.  Displays source and destination
  HLP=Y  directories in vertical windows.  ZF-like commands.  Supports file
  CFG=Y  selection and copy by datestamp.  Extended TCAP required.
```

```
SUB.COM      3.50   0 0      4  32 7379 SUB35     24 01/30/92 Bruce Morgen
  SYS    Replacement for the SUBMIT program provided with CP/M.  Provides
  HLP=Y  nestable SUBMIT runs, interactive entry, and user customization.
  CFG=N  Vs. 1.0 (10/81) by Richard Conn.  From SuperSUB 1.1 by Ron Fowler.

SUBINST.COM  3.50   0 0      1   8 7ED5 SUB35     24 01/30/92 Bruce Morgen
  SYS    Install program for SUB.COM vs 3.5.
  HLP=Y
  CFG=N

TCSELECT.COM 3.00   0 0      5  35 780B TCSELE30  13 12/04/91 Brian Moore
  SYS    Allows user to interactively review the contents of a Z3TCAP.TCP
  HLP=Y  file and select a terminal from it.  Supports extended TCAPs
  CFG=N  compatible with the Version 4 libraries.

UMAP.COM     1.80   0 0      2  16 903C UMAP18    23 10/10/91 Gene Pizzetta
UMAP.3OM     1.80   3 0      2  16 D4A5 UMAP18    23 10/10/91 Gene Pizzetta
UMAP.4OM     1.80   4 0      3  22 EC57 UMAP18    23 10/10/91 Gene Pizzetta
è  DIR    Shows which user areas have files and how many directory entries those
  HLP=Y  files use.  Can also show:  total nbr of entries used, those still
  CFG=Y  free, free disk space, as well as a list of empty directories.

XFOR.COM     1.50   0 V302   6  41 2343 XFOR15    44 12/01/91 Gene Pizzetta
  BBS    Z-System FOR utility for displaying "- - " delimited file catalogs.
  HLP=Y  Command line source file specification.  Numerous configuration
  CFG=Y  options can be set with ZCNFG.

Z33IF.COM    1.60   3 0      4  28 6B0E Z33IF16   39 10/13/91 Bruce Morgen
Z33IF.3OM    1.60   3 0      4  28 90C9 Z33IF16   39 10/13/91 Bruce Morgen
Z33IF.4OM    1.60   4 0      4  32 742E Z33IF16   39 10/13/91 Bruce Morgen
  SYS    Adaption of COMIF.COM for ZCPR3.3.
  HLP=Y
  CFG=N

ZBIB.COM     1.00   0 0      8  61 4E0B ZBIB10    18 11/29/91 Joe Mortensen
  DBASE  Bibliographic database manager derived from ZDB.
  HLP=Y
  CFG=Y

ZCNFG.COM    2.10   4 V212   8  62 FB96 ZCNFG21  115 11/07/91 Al Hawley
  SYS    Universal configuration utility which configures option data in
  HLP=Y  executable files.  Uses .CFG overlay file.
  CFG=Y

ZDB.COM      1.80   0 0      8  64 D1C2 ZDB18     51 01/30/92 Joe Mortenson
  DBASE  Small, fast name and address file manager with built-in label and
  HLP=Y  envelope addressing features.  NZTCAP and VLIB4D support.
  CFG=Y

ZDT.COM      1.20   0 0      8  59 A635 ZDT12     20 01/27/92 Joe Mortensen
ZDT+.COM     1.20   0 0      8  62 ED48 ZDT12     20 01/27/92 Joe Mortensen
  DBASE  Z-System Day Timer, a daily planning calendar derived from ZDB.
  HLP=Y  Automatically reads real-time clock and displays current day's
  CFG=Y  schedule.  Requires ZCPR3.0+ and extended TCAP.  Version for Z3PLUS.

ZF10QD4+.COM 1.0q   3 0     15 119 4B0E ZF10Q    118 01/14/92 Rob Friefeld
ZF10QD5+.COM 1.0q   3 0     15 119 94E5 ZF10Q    118 01/14/92 Rob Friefeld
ZF10QD5.COM  1.0q   3 0     15 115 537C ZF10Q    118 01/14/92 Rob Friefeld
ZF10QR4+.COM 1.0q   3 0     15 119 04BB ZF10Q    118 01/14/92 Rob Friefeld
ZF10QR4.COM  1.0q   3 0     15 115 BD22 ZF10Q    118 01/14/92 Rob Friefeld
ZF10QR5+.COM 1.0q   3 0     15 119 1ED9 ZF10Q    118 01/14/92 Rob Friefeld
  FILE   Enhanced version of VFILER (follow-on to VF42B) designed to take
  HLP=Y  advantage of ZCPR3.3+ facilities.  Versions for 4/5-cols, dim/
  CFG=Y  reverse video, DateStamper support.  Vs 1.0 (1/1/87) by Jay Sage.

ZSLIB.REL    3.40   4       29 228 1C4E ZSLIB34   84 10/09/91 Gene Pizzetta
ZSLIBS.REL   3.40   4       27 211 B72C ZSLIB34   85 10/09/91 Gene Pizzetta
  PROG1  Assembly language routines to assist programmers in handling date-
  HLP=Y  stamp maintenace under ZSDOS, Z3PLUS, and CP/M Plus.  Microsoft
  CFG=N  REL and SLR formats.
è
ZTIME+.COM   1.40   0       4  27 874C ZTIME14    4 10/10/91 Gene Pizzetta
ZTIME+.3OM   1.40   3       4  27 691A ZTIME14   40 10/10/91 Gene Pizzetta
ZTIME+.4OM   1.40   3       5  34 4B58 ZTIME14   40 10/10/91 Gene Pizzetta
ZTIME.3OM    1.40   3       4  27 07C0 ZTIME14   40 10/10/91 Gene Pizzetta
ZTIME.4OM    1.40   3       5  35 D986 ZTIME14   40 10/10/91 Gene Pizzetta
ZTIME.COM    1.40   0       4  27 CACA ZTIME14   40 10/10/91 Gene Pizzetta
  DATE   A hardware independent clock utility for setting or displaying the
  HLP=N  date and time under ZSDOS or Z3PLUS.  Options include the ability to
```

Rob Friefeld has released an update to ZFILER (ZF10Q) with the following enhancements: 1) the ability to filter VIEW and PRINT output (high bits are removed, only alphanumerics, CRs, and LFs are printed, and TABs are expanded); 2) the ability to remember all the file tags on return from a Z command or macro run (mentioned above). The file list is written to a temporary disk file (ZFILER.TAG, in a configurable directory), and automatically read back. 3) the ability to set the Group Tag/Untag and Wild Tag/Untag to work on the entire ring, or just from the file pointer (Group Reverse always works on the entire ring). List macros now soft-tag files just like regular group macros.

Z Message Base

Question: What is the acceptable way of handling the quiet flag? Answer. I usually handle the quiet flag this way:

a. The program has a configuration byte that allows setting it to default to quiet mode.

b. If the quiet flag is set, the program defaults to quiet mode no matter how the configuration byte is set.

c. The command line Q option toggles whatever is the current default, whether the default has been set by the configuration byte or the quiet flag.

Many programs I have released use this method, including DATSTP, CONCAT, etc. (Gene Pizzetta, Z-Node #3, 2/2/92) ●

"Reading made Don Quixote a gentlemen, but believing what he read made him mad."

-George Bernard Shaw

TCJ The Computer Journal

The Spirit of the Individual Made This Industry

Back Issues

Telephone Orders: (800) 424-8825 / (908) 755-6186, 24 hours

Special Close Out Sale
Issues 1, 2, 3, 4 only

3 or more, $1.50 each postpaid in the US
$3.00 postpaid airmail outside US.

Issue Number 1:
- RS-232 Interface Part 1
- Telecomputing with the Apple II
- Beginner's Column: Getting Started
- Build an "Epram"

Issue Number 2:
- File Transfer Programs for CP/M
- RS-232 Interface Part 2
- Build Hardware Print Spooler Part 1
- Review of Floppy Disk Formats
- Sending Morse Code with an Apple II
- Beginner's Column: Basic Concepts and Formulas

Issue Number 3:
- Add an 8087 Math Chip to Your Dual Processor Board
- Build an A/D Converter for Apple II
- Modems for Micros
- The CP/M Operating System
- Build Hardware Print Spooler: Part 2

Issue Number 4:
- Optronics, Part 1: Detecting, Generating and Using Light in Electronics
- Multi-User: An Introduction
- Making the CP/M User Function More Useful
- Build Hardware Print Spooler: Part 3
- Beginner's Column: Power Supply Design

Issue Number 18:
- Parallel Interface for Apple II Game Port
- The Hacker's MAC: A Letter from Lee Felsenstein
- S-100 Graphics Screen Dump
- The LS-100 Disk Simulator Kit
- BASE: Part Six
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 1

Issue Number 20:
- Designing an 8035 SBC
- Using Apple Graphics from CP/M: Turbo Pascal Controls Apple Graphics
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K

Issue Number 21:
- Extending Turbo Pascal: Customize with Procedures & Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition & Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC

Issue Number 22:
- NEW-DOS: Write Your Own Operating System
- Variability in the BDS C Standard Library
- The SCSI Interface: Introductory Column
- Using Turbo Pascal ISAM Files
- The Ampro Little Board Column

Issue Number 23:
- C Column: Flow Control & Program Structure
- The Z Column: Getting Started with Directories & User Areas
- The SCSI Interface: Introduction to SCSI
- NEW-DOS: The Console Command Processor
- Editing the CP/M Operating System
- INDEXER: Turbo Pascal Program to Create an Index
- The Ampro Little Board Column

Issue Number 24:
- Selecting & Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assemble Code for CP/M
- The C Column: Software Text Filters
- Ampro 186 Column: Installing MS-DOS Software
- The Z-Column
- NEW-DOS: The CCP Internal Commands
- ZTime-1: A Real Time Clock for the Ampro Z-80 Little Board

Issue Number 25:
Sold Out

Issue Number 26:
- 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: CCP Internal Commands, Pt 2
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS

Issue Number 27:
- 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

Issue Number 28:
- 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

Issue Number 29:
- 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
- The ZCPR3 Corner

Issue Number 30:
- 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
- The ZCPR3 Corner
- The CP/M Corner

Issue Number 31:
- Using SCSI for Generalized I/O
- Communicating with Floppy Disks: Disk Parameters & their variations
- XBIOS: A Replacement BIOS for the SB180
- K-OS ONE and the SAGE: Demystifying Operating Systems
- Remote: Designing a Remote System Program
- The ZCPR3 Corner: ARUNZ Documentation

Issue Number 32:
- Language Development: Automatic Generation of Parsers for Interactive Systems
- Designing Operating Systems: A ROM based OS for the Z81
- Advanced CP/M: Boosting Performance
- Systematic Elimination of MS-DOS Files: Part 1, Deleting Root Directories & an In-Depth Look at the FCB
- WordStar 4.0 on Generic MS-DOS Systems: Patching for ASCII Terminal Based Systems
- K-OS ONE and the SAGE: System Layout and Hardware Configuration
- The ZCPR3 Corner: NZCOM and ZCPR34

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

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

Issue Number 35:
- All This & Module-2: A Pascal-like alternative with scope and parameter passing.
- A Short Course in Source Code Generation: Disassembling 8088 software to produce modifiable assembly 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.
- The Computer Corner

Issue Number 36:
- Information Engineering: Introduction.
- Module-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
- The Computer Corner.

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 & cooked console I/O
- Real Computing: The NS 32000.
- ZSDOS: Anatomy of an Operating System: Part 1.
- The Computer Corner.

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
- 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, Pt 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.
- The Computer Corner.

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 dEXL: Writing your own custom designed business program.
- Advanced CP/M: ZEX 5.0The 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
- The Computer Corner.

Issue Number 41:
- Forth Column: ADTs, Object Oriented Concepts.
- Improving the Ampro LB: Overcoming the 66Mb 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 & printer functions w/C.
- LINKPRL: Making RSXes easy.
- SCOPY: Copying a series of unrelated files.
- The Computer Corner.

The Computer Corner

By Bill Kibler

A new year has started and with it a few new leafs are being turned over by myself. Hopefully I am getting ahead of things as I will be moving houses again soon. What I want to cover this time is some understanding about a few key words I use. They are important words and as such I feel a whole article on them is needed.

READABILITY

is my first word and is not what most users understand it to mean. To me to be readable, in the case of a manual, is not just covering all the topics, but educating the reader as well. To be readable it must have a clearly defined structure. The reader needs to be able to find items of interest quickly. The structure must clear enough that a reader can see the structure and start using it immediately.

This topic is fresh with me as I have been spending a large amount of time with one of our technical writers. We have a new product going out and thus a new manual. They had taken an old manual and just changed a few key words and items. Well, that was until it hit my desk for review. To say the red ink flowed over many pages would be an understatement. I feel the writer had never had anybody in the company ever really review one of his manuals before.

What am I changing to make the manual more readable? The first problem was a lack of structure. The installation section ran on forever and intermixed topics and instructions. The new manual now has a clear introduction that describes terms used and how things work. That brings the reader up to the point of knowing what and why the remaining steps are needed. Those steps now are broken into separate chapters or sections that coincide with a logical grouping of needed actions.

In short, the document lacked any clear structure. It was almost impossible to find a given topic without some hunting and then the why or how of the function might not be apparent. That "how and why" is actually very important as it helps the reader weigh the information for relevance to their needs. I remember reading one section and thinking we had already covered it, when I suddenly discovered the topic was actually different. The problem was two terms that were very close to each other.

I can not complain about use of terms enough. We all have had troubles with acronyms. Those wonderful two and three letter statements that mean nothing and are never explained. Manuals are full of them and my complaint is they do not get explained often enough. Do it once or twice a section at least. But another problem with terms is use of similar or repetitive terms. In our example was the overly used term "file".

I am sure we all have seen the term *file* used 10 or 15 times in one paragraph. How many of you can keep track of which file we are talking about. In our manual it wasn't even a file, but really a list of lists stored in a file structure. We have multiple indexes into these lists which determine which file gets sent to where (see even describing the problem gets confusing). The manual referenced the list as "XX file" and then said it loaded files from it. Not only was it confusing but inaccurate as well. The solution involved explaining that it was a list. Then explaining how the list is multiply indexed. Lastly we showed the relationship between the indexes and the file name that was ultimately loaded.

STRUCTURE

I could go on with the manual problems and pick over lots of the fine points. The objective here is to get you to save time and money when preparing your own manuals. I say this after having seen a friends manual he product for his new product. His first problem was size, over an inch of standard letter size pages. The printing cost are about $15 each and the labor cost had he paid for them are enormous. The reason I feel it was wasted was lots of extra work was performed that will not return him any buyers.

Now writing to sell is not the best way to write a manual. Writing with a clearly defined audience and structure will however produce a readable and marketable item. How so? Over the years I have used the structure that says a manual should be broken into a beginners, intermediate, and advanced section. I find that idea is sound no matter which group is using the document. Sales literature, fliers, and each section of a manual can also use this structure. We start a sales pitch with the old adages of tell them "WHO, WHAT, and WHY".

I have changed the who, what, why, to introduce it, explain it, and show it off. These really are still my favorite structures of beginner, intermediate, and advanced. Remember that whenever you start a new topic or look at a new product your start as a beginner. You learn enough about it to then start asking questions about what it will do for me. At that point you have passed into the intermediate state where we then start asking question like "why can't I do this or that".

My suggestion to my friend was change his structure. Most of the information has already been written, it just needs to be restructured. I feel also that as he restructures he will see that many items and topics are covered more than once. Once the structure is clearly outlined it also makes it possible to judge the content of sections. Too often I find whole subsections that just do not belong either in that location or in the manual.

When talking about sales, I like the clear structuring as it allows for a single section (usually the beginners section) to be pulled out and shipped to would be buyers. That may also help you understand the importance of what a buyer looks for in a manual. I have my own items I check for before buying as I am sure you do to. I find however that most people check out items like indexes, table

See Computer Corner, page 19