The COMPUTER JOURNAL®

Programming - User Support
Applications

Issue Number 37 March / April 1989 $3.00

ISSN # 0748-9331

THE COMPUTER JOURNAL

Editor/Publisher
Art Carlson

Art Director
Donna Carlson

Circulation
Donna Carlson

Contributing Editors

C. Thomas Hilton
Bill Kibler
Bridger Mitchell
Bruce Morgan
Richard Rodman
Jay Sage
Barry Workman

The Computer Journal is published six times a year by Publishing Consultants, 190 Sullivan Crossroad, Columbia Falls, MT 59912 (406) 257-9119

Entire contents copyright © 1989 by Publishing Consultants.

Subscription rates—$16 one year (6 issues), or $28 two years (12 issues) in the U.S., $22 one year in Canada and Mexico, and $24 (surface) for one year in other countries. All funds must be in US dollars on a US bank.

Send subscriptions, renewals, or address changes to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, Montana, 59912.

Address all editorial and advertising inquiries to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912 phone (406) 257-9119.

The Lillipute Z-Node sysop has made his BBS systems available to the TCJ subscribers. Log in on both systems (312-649-1730 & 312-664-1730), and leave a message for SYSOP requesting TCJ access.

The COMPUTER JOURNAL

Features

Issue Number 37

March / April 1989

Everything you see here...
12 MHz 80286
AT-Compatible.
1Mb on-board DRAM
Full set of AT-compatible controllers
EGA, CGA, MDA, Hercules video
SCSI/FD controllers
. . . and more

you see here.

THE AMPRO LITTLE BOARD™/286

Big power for smaller systems. Little Board/286 is the newest member of our family of MS-DOS compatible Single Board Systems. It gives you the power of an AT in the cubic inches of a half height 5 1/4" disk drive. It requires no backplane. It's a complete AT-compatible system that's functionally equivalent to the 5-board system above. But, in less than 6% of the volume. It runs all AT software. And its low-power requirement means high reliability and great performance in harsh environments.

Ideal for embedded & dedicated applications. The low power and tiny form factor of Little Board/286 are perfect for embedded microcomputer applications: data acquisition, controllers, portable instruments, telecommunications, diskless workstations, POS terminals . . . virtually anywhere that small size and complete AT hardware and software compatibility are an advantage.

Compare features.
Both systems offer:
- 8 or 12MHz versions
- 512K or 1Mbyte on-board DRAM
- 80287 math co-processor option
- Full set of AT-compatible controllers
- 2 RS232C ports
- Parallel printer port
- Floppy disk controller
- EGA/CGA/Hercules/MDA video options
- AT-compatible bus expansion
- A wide range of expansion options
- IBM-compatible Award ROM BIOS

But only Little Board/286 offers:
- 5.75" x 8" form factor
- EGA/CGA/Hercules/MDA on a daughterboard with no increase in volume
- SCSI bus support for a wide variety of devices: Hard disk to bubble drives
- On-board 1Kbit serial EPROM. 512 bits available for OEMs
- Two byte-wide sockets for EPROM/RAM/NOVRAM expansion (usable as on-board solid-state disk)
- Single voltage operation (+5 VDC only)
- Less than 10W power consumption
- 0-70°C operating range

*AT is a Registered Trademark of IBM Corp.

Better answers for OEMs. Little Board/286 is not only a smaller answer, it's a better answer . . . offering the packaging flexibility, reliability, low power consumption and I/O capabilities OEMs need . . . at a very attractive price. And like all Ampro Little Board products, Little Board/286 is available through representatives nationwide, and worldwide. For more information and the name of your nearest Rep, call us today at the number below. Or, write for Ampro Little Board/286 product literature.

408-734-2800

Fax: 408-734-2939 TLX: 4940302

AMPRO
COMPUTERS, INCORPORATED
1130 Mountain View/Alviso Road
Sunnyvale, CA 94089

Reps: **Australia**-61 3 720-3298; **Belgium**-32 87 46.90.12; **Canada**-(604) 438-0028; **Denmark**-45 3 66 20 20; **Finland**-358 0 585-322; **France**-331 4502-1800; **Germany, West**-49 89 611-6151; **Israel**-972-3 49-16-95; **Italy**-39 6 811-9406; **Japan**-81 3 257-2630; **Spain**-34 3 204-2099; **Sweden**-46 88 55-00-65; **Switzerland**-41 1 740-41-05; **United Kingdom**-44 2 964-35511; **USA**, contact AMPRO.

Editor's Page

Is UNIX in Your Future?

The hardware developers have made great strides in producing more powerful CPUs. Thirty-two bit processors with clock speeds of 30 MHz or more will be the normal high-end system by the end or this year. Desktop units with these processors can provide processing power which surpasses the mainframes of a few years ago, and the trend towards replacing mainframes with desktop units will accelerate. Systems using microprocessors such as the Z-80, 680X0, 8086, or 80X86 have been called microcomputers, but when we have desktop units with more power than the minicomputers, we'll have to coin a new term.

These powerful CPUs provide us with the hardware portions of high speed multiuser multitasking systems for large scale business applications, but the software developments have not kept pace with the hardware. While the CPU is the brain of the computer, the operating system is the nerve center which connects the brain to the parts which do the work.

Most of attempts at Multiuser Multitasking Operating Systems (I'll call them MMOS to save keystrokes) have failed to achieve the critical mass required to attract the high performance reasonably priced application programs which are absolutely necessary for commercial success. Success is determined by both the hardware and the operating system, but the respective hardware platforms would not have sold without the standardized environments provided by the CP/M, AppleDOS, PC/MS-DOS, or Macintosh systems.

Long term readers will be well aware of my negative feelings about MMOSs. I was turned off by the results of implementations on slow, limited memory, eight and sixteen bit systems—and by experience with the early multiuser minicomputers. (A local government agency is still using a mini which sometimes takes as long as eight seconds to return a keystroke during wordprocessing!) I have always wanted multiple CPUs for a single user, but the new hardware can provide the performance required in business applications—if only we had the right OS and software.

It would be nice to grow a MMOS from one of our current systems so that we could continue to use all of our familiar tools and utilities. It is difficult to abandon our favorite TSR's and our expert knowledge of PC-DOS tricks, but upgrading DOS to an MMOS by adding makeshift patches is about as practical as replacing the mule team on an 1850 covered wagon with a 440 cubic inch supercharged engine—it just ain't going to work! We have survived the change from AppleDOS to CP/M to ZCPR3 to PC-DOS, and now we have to change to another different OS.

In spite of the efforts of IBM, Microsoft, and Digital Research with OS/2 and Concurrent DOS, it appears that UNIX® is the only real contender for serious business use. In view of this, TCJ will be publishing information on UNIX and the choice of upgrade paths from a PC/AT platform. Our goal is to provide migration path technology to minimize the trauma of the change. One of the products we will be working with is the *MKS Toolkit* (Mortice Kern Systems, Inc., 35 King Street North, Waterloo, Ontario, Canada N2J 2W9, phone (519) 884-2251), which is a highly recommended UNIX learning tool. I picked up the descriptive term "migration path technology" from Ruth Songhurst at MKS.

UNIX is in your future if you intend to be involved in more than just hobby type systems—start planning for it now. As usual, your suggestions and articles are welcome.

Industry Watch

I've been hearing a lot of rumors about Kaypro, so I called their PR department to find out what is going on. I asked, "Have there been any significant changes in the past few months?" The answer I received was, "Yes, but I'm not the person to tell you. I'll have Mr. Kay call you back." I have not heard any more, but it has only been a few days and they may still return my call. I tried to call the local Kaypro dealer, but they have closed the business. I don't feel very confident about Kaypro's future.

The computer hardware and software market is turning into a simple commodity distribution channel. The problem is that computers are not a simple commodity. Users need technical support, but there is not enough profit margin to enable the dealers to provide the required support.

If a reputable dealer spends five to ten hours performing a needs analysis for the user, and then the user buys the system somewhere else where the price is a few dollars cheaper, who is the real winner? Is it the low priced seller who made a small profit without doing the work? Is it the user who used one dealer's services and then bought elsewhere? What does the low priced buyer do when they need technical support? How will it affect the user and the low price seller if the full service dealer goes out of business?

There is no easy answer. We all like to save money on a purchase, and I bought my AT system mailorder where I could get the best value (not necessarily the lowest price). But, I felt that I was in the position to analyze my needs, and I was prepared to provide whatever support I needed. I was even mentally prepared to suffer the loss if I ended up with a useless system. And, I did not ask someone else to provide free consultation services.

It appears that there is a need for independent consultants who can perform the needs analysis required to determine the necessary hardware and software, and to provide the after sale support and training. The problem is (at least here in Northwest Montana) that the people who shop the hardest for the lowest price are not willing to pay for any support—they expect everything to be given to them with the system. They consider themselves "Smart Shoppers" but they end up with the wrong systems or with systems which they never learn how to use.

Selecting the proper system and software with after-the-sale training and support is vital to a business. I don't know what the outcome will be, but reputable

(Continued on page 44)

C Pointers, Arrays & Structures Made Easier

Part 1: Pointers

by Clem Pepper

From my own experience with the C language as a hobbyist and experimenter I consider the three topics of the title as among the most difficult to understand fully. Failure to understand the basics of these places significant limits on what we may achieve with our programs. It is to our advantage to make use of their special attributes.

An objective of combining these three topics in a single article series is to bring out the close relationship that exists between them. That is, pointers and arrays have much in common, as will be seen. The structure, from certain viewpoints, is a super array in its capabilities. At an even higher level we find the union, though not with the frequency of the array or structure.

The example exercises provided should compile and run with any C compiler. The one possible exception is the ANSI screen clear, but this has no bearing on the examples themselves.

Pointer Basics

In the construction of functions for our programs we make type declarations of various kinds. These typically are int, char, float, double, unsigned ... At times we see declarations such as "int *value;" or possibly "char *string;." Unless we are experienced with pointer usage we will wonder at the meaning of these.

Pointers are confusing to be sure but an understanding of their usage is essential to working in C because they are to be found everywhere.

By definition, "A pointer is a variable that contains the address of another variable." K & R refers to this variable as an "object" (Chapter 5, p89). The asterisk, when used in this manner, is known as a "unary operator which treats its operand as the address of the ultimate target and accesses that address to fetch the content." (The operand is the expression immediately following the *.)

At this point the meaning of "unary" and "binary" as applied to operators must be understood. These refer to the number of associated operands. A "unary" operator has application to a single operand; a "binary" to two. For example, the minus sign in the assignment "this__var = -3;" is a unary operator. When "this__var = (var1 - var2);" the minus sign is a binary operator.

A related unary operator is the ampersand, &. Where the unary * points to an address, the ampersand is the address. We quite likely have encountered this use of the & when using scanf. Whenever we use scanf to read input from the keyboard we must specify an address at which that input is to be placed. We write:

```
scanf(''%d'', &entry);
```

where "entry" is the keyboard input stored in memory at the address given by "&entry."

The asterisk as a binary operator appears as:

```
result = (var1 * var2);
```

multiplying the two variables. The binary ampersand in a bit masking operation such as

```
masked_var = (var & 0xF6);
```

performs a logical AND between var and hexadecimal F6.

In pointer operations the * and the & are unary operators. The compiler distinguishes between the two by their usage. The same is true for ourselves.

A declaration of "int *value" or "char *string" defines the type of whatever is stored at the address pointed to. Thus "int" is the type of a quantity located at memory address "value." Similarly for the character declaration; the address pointed to by "string" contains a type char. Luckily we do not have to be concerned with the actual locations—the compiler takes care of this for us.

Earlier we employed the example:

```
scanf(''%d'', &entry);
```

to illustrate the unary &. A reasonable query is how are we to make use of the variable "entry?"

The direct approach is to employ the variable in its integer form. Such as:

```
if(entry != 510) printf(''Sorry 'bout that!);
```

The indirect approach on the other hand is to assign a pointer variable:

```
this_ptr = &entry;
```

or alternatively

```
int entry = *this_ptr;
```

A distinction to be stressed here is that "&entry" is a constant whereas "*this__ptr" is not so constrained. That is, we can make "this__ptr" point to any address. Thus, as our program requires we can re-direct "this__ptr" to whatever variable is appropriate. To illustrate:

```
int this_ptr, first, last, inbetween;
this_ptr = &first;     /* initially */
this_ptr = &inbetween; /* later */
this_ptr = &last;      /* finally */
```

In this illustration we are assigning an address to "this__ptr." In general our interest is not so much in the address as its content. We obtain the content through indirection when we make the pointer declaration:

```
 int *this_ptr, first, last, inbetween;
*this_ptr = first;     /* initially */
*this_ptr = inbetween; /* later */
*this_ptr = last;      /* finally */
```

When we declare "int val = 510;" we are assigning the value 510 to the integer variable "val." When we declare "int *val = 510;" we are not making the assignment to "val" but to the content of the memory location pointed to by "val."

Suppose there are three friends—Al, Joe and Jane. Through some means Jane learns there is going to be a party at Joe's in May, but her information doesn't include which day. Joe knows the day, but Jane doesn't have a current address (or phone number) for Joe. But she knows that Al does. So Jane calls Al, whose number she has, to learn the day of the party. (Admittedly, from a practical standpoint, Jane needs to know more than the date, but let's not get into that here!)

In other words, Jane wants information known to Joe. Jane cannot access Joe directly. But Al can. So Jane passes a request to Al who interrogates Joe. Al gets a reply and passes it back to Jane.

Then:

```
    Al = &Joe;
```

"Al" is assigned the address of "Joe." Thus, while Al 'points' to Joe's location, *Al indicates the content of what Al is pointing to, which is Joe's knowledge of the party date. Note that content may be written into or read out from Joe.

Now, we have this other integer, Jane. We make the assignment Jane = *Al. With this, Jane acquires the content of whatever Al is pointing at. But the inverse of the pointer operator, *, is the address operator, &.

So if Al = &Joe then *Al = *&Joe = Joe = Jane.

It is necessary to declare each of the participants. These are:

```
    int Joe, Jane, *Al;  /* Why is *Al an int? */
```

Let's write this into a brief illustrative program:

```
#include <stdio.h>
main()
{
int Jane, Joe, *Al;  /* addresses are type int     */
Joe = 18;            /* knows date for the party */
Al = &Joe;           /* Al has Joe's address     */
Jane = *Al;          /* She wants party date     */
  printf(''Jane got the date of the party, May %d, from Joe \
through Al.\n'',Jane);
   exit(0);
}
```

When you compile and run this program your screen will read:

```
''Jane got the date of the party, May 18, from Joe through Al.''
```

(The complete program is given in Listing 1.)

A more realistic illustration is given by Listing 2. Here we employ scanf to enter two numbers from the keyboard. Pointers are assigned to the two entries, and variable assignments made to the pointers. Addition and subtraction are then performed using the variable assignments.

In a real application we would not take this round about route—we will simply perform the arithmetic with the two variables, entry1 and entry2, directly.

A frequent programming requirement is the transfer of variable values between functions. Recollect that local variables are known only within the function in which they are declared. If the need arises to employ the same variable in two functions we are faced with two options. The first is to declare the variable as a global. If the variable is used in several functions this may be the best approach. But frequently the most efficient is to employ pointers to communicate a variable's address to the needed function.

The use of pointers for local variable value transfer is illustrated in Listing 3. This example is similar to Listing 2 except that the arithmetic is now carried out in two dedicated functions. That is, the functions

```
   int sum(addr1,addr2)
   int diff(addr1,addr2)
```

are receptive to input from any function requiring the addition or subtraction of two integer variables. The two integer values are assigned to local variables num1 and num2 by the pointers *addr1 and *addr2. Their sum/difference is returned to the calling function by the respective return statements.

The next listing, 4, carries this concept one step further. In Listing 3 two arithmetic functions, int sum(addr1,addr2) and int diff(addr1,addr2), perform the arithmetic functions of addition and subtraction respectively. In Listing 4 these have been combined into a single function, int arith(addr1,addr2).

The declarations for int arith(addr1,addr2) look the same as those for the sum and difference functions in the preceding example. Inside the braces, however, we see a distinct difference. The return statements of Listing 3 have been replaced by the two pointers, *addr1 and *addr2, for return of the addition and subtraction values. In this example we have taken advantage of pointers to reduce two arithmetic functions to one. Two previous variables, plus and minus, have also been eliminated. Note their replacement by the pointer variables in the closing printf statement.

Compare the length of Listing 4 with Listing 3. This last example illustrates a potential criticism. When pointer usage is carried to an extreme our programs can become very difficult to follow. This becomes true for the creator as well as others. The abundant use of comments is recommended.

This far we have discussed pointers to integers. While pointers to characters follow the same rules the application may differ significantly.

If we have the line DEVICE = ANSI.SYS in our CONFIG.SYS file we can clear the screen and home the cursor in the following manner:

```
char *CLRSCRN = ''\033[2J'';
 printf(''%s'',CLRSCRN);
```

This is included in the example listings. Do not type it in your program if your system lacks ANSI.SYS. You will also note that some examples use the #define CLRSCRN = " \033[2J" instead. Also that

```
 puts(CLRSCRN);
```

functions for either declaration.

Character pointers are typically pointers to strings. A string is identified by the double quotes at the beginning and end of the expression. Strings in C are terminated with the null character, ' \0.' We do not include the null in our declaration as that is performed by the compiler. Pointers to strings offer an impressive range of flexibility to our programming.

Just as Jane, in our initial example program, called Al to learn the date of the party, printf("%s",CLRSCRN); is an instruction to go to the location pointed to by CLRSCRN and print its content on the screen. In this example the content happens to be an instruction to clear the screen and home the cursor. But we can write

```
    char *name = ''Jane'';
    printf(''%s'', name);
```

and see 'Jane' displayed on the screen.

What makes this so great is that throughout our program we can reassign 'name' as often as needed. If 'name' is linked in with a condition, such as scoring in a game, we can write statements on the order of:

```
  if(George_scr > Jane_scr) name = ''George'';
```

We can see this for ourselves when we run the program of Listing 5. This could be part of the scoring for a war game in which the players military rank increases with hits made on the enemy. You might want to add to the ranks right up through five-star general.

The utility of string pointers provides an excellent lead into the array. In the next segment of this series we will learn the declarations:

```
    char *name = ''Jane'';
```

and

```
    char name[] = ''Jane'';
```

are one and the same.

Listing 1

```
/* PTR_EX1.C --------------------------------- **
** Pointer examples get no smaller than this. */
#include <stdio.h>
char *CLRSCRN = ''\033[2J'';  /* MS DOS ANSI screen clear */
/* == Begin program == */
main()
{
int Jane, Joe, *Al;
Joe = 18;
Al = &Joe;
Jane = *Al;
 puts(CLRSCRN);
  printf(''Jane got the party date, May %d, from Joe through Al.\n'',Jane);
   exit(0);
}
```

Listing 2

```
/* PTR_EX2.C  ------------------------- **
** Pointers assigned to keyboard input  */
#include <stdio.h>
char *CLRSCRN = ''\033[2J'';  /* MS DOS ANSI screen clear */
/* == Begin program == */
main()
{
  int entry1, entry2, *addr1, *addr2;
  int num1, num2, add, sub;
   printf(''%s'', CLRSCRN);
    puts(''Enter a number between 10 and 20: '');
     scanf(''%d'', &entry1);
    puts(''Enter a number between 20 and 30: '');
     scanf(''%d'', &entry2);
  /* ** assign pointers addr1, addr2 to entry addresses ** */
    addr1 = &entry1; addr2 = &entry2;
  /* ** assign variables to pointers *addr1, *addr2 ** */
    num1 = *addr1;   num2 = *addr2;
     add = num1 + num2;
     sub = num2 - num1;
      printf(''The sum of the two numbers entered is %d.\n'',add);
      printf(''Their difference is %d.\n'',sub);
       exit(0);
}
```

Listing 3

```
/* PTR_EX3.C  --------------------------- **
** Function calls with pointer variables  */
#include <stdio.h>
int sum();
int diff();
char *CLRSCRN = ''\033[2J'';  /* MS DOS ANSI screen clear */
/* == Begin program == */
main()
{
   int entry1, entry2, *addr1, *addr2;
   int plus, minus;
    printf(''%s'', CLRSCRN);
     puts(''Enter a number between 10 and 20: '');
      scanf(''%d'', &entry1);
     puts(''Enter a number between 20 and 30: '');
      scanf(''%d'', &entry2);
   /* ** assign pointers addr1, addr2 to entry addresses ** */
     addr1 = &entry1; addr2 = &entry2;
      plus = sum(addr1,addr2);
      minus = diff(addr1,addr2);
       printf(''The sum of the two numbers entered is %d.\n'',plus);
       printf(''Their difference is %d.\n'',minus);
        exit(0);
}

/* == Obtain the sum of the entries == */
int sum(addr1,addr2)
int *addr1, *addr2;
{
 int num1, num2;
  /* ** assign variables to pointers *addr1, *addr2 ** */
    num1 = *addr1;   num2 = *addr2;
     return(num1 + num2);
}

/* == Obtain the difference of the entries == */
int diff(addr1,addr2)
int *addr1, *addr2;
{
 int num1, num2;
  /* ** assign variables to pointers *addr1, *addr2 ** */
    num1 = *addr1;   num2 = *addr2;
     return(num2 - num1);
}
```

Listing 4

```
/* PTR_EX4.C  ------------------------------ **
** Pointer only function calls and returns   */
#include <stdio.h>
char *CLRSCRN = ''\033[2J'';  /* MS DOS ANSI screen clear */
/* == Begin program == */
main()
{
  int entry1, entry2, *addr1, *addr2;
   printf(''%s'', CLRSCRN);
    puts(''Enter a number between 10 and 20: '');
     scanf(''%d'', &entry1);
    puts(''Enter a number between 20 and 30: '');
     scanf(''%d'', &entry2);
  /* ** assign pointers addr1, addr2 to entry addresses ** */
    addr1 = &entry1; addr2 = &entry2;
     arith(addr1,addr2);  /* for sum and difference arithmetic */
      printf(''The sum of the two numbers entered is %d.\n'',*addr1);
      printf(''Their difference is %d.\n'',*addr2);
      exit(0);
}

/* == Obtain the sum and difference of the two entries == */
int arith(addr1,addr2)
int *addr1, *addr2;
{
 int num1, num2;
  /* ** assign variables to pointers *addr1, *addr2 ** */
    num1 = *addr1;   num2 = *addr2;
     *addr1 = (num1 + num2);
     *addr2 = (num2 - num1);
      return;
}
```

Listing 5

```
/* PTR_EX5.C ------------------------ **
** String pointers enjoy versatility  */
#include <stdio.h>
char *CLRSCRN = ''\033[2J'';  /* MS DOS ANSI screen clear function */
char *RANK = ''PRIVATE'';
main()
{
 int run = 1, score = 600;
  puts(CLRSCRN);
   printf(''Show us your stuff, %s\n'',RANK);
    do                  {
     new_rank(score);
      printf(''Enjoy your promotion to %s.\n'',RANK);
       score += 600;
        if(score >=3000) run = 0;
                        }
        while(run) ;
        exit(0);
}
/* == increase rank as score adds up == */
int new_rank(up)
int up;
{
      if(up < 1000) { RANK = ''CORPORAL'';   return; }
  else if(up < 1700) { RANK = ''SERGEANT'';   return; }
  else if(up < 2400) { RANK = ''LIEUTENANT''; return; }
  else if(up < 3400) { RANK = ''CAPTAIN'';    return; }
}
```

Summary

In this segment we learned that the unary operator '&' preceding a variable name defines the address of the variable. An '*' preceding a variable name tells us the variable is a pointer to an address containing a value assigned to that address. The '*' is said to be an "indirection" operator. The type of the value at that location is the type of the pointer declaration. Thus:

```
int *addr1;
```

declares the content of what is pointed as type int.

Pointers lend flexibility to our programs and contribute to their efficiency in important respects. They can also contribute to difficulty in following the program's sequence of operations later unless we are careful. Comments are helpful in avoiding this difficulty.

Examples of pointer usage for integer and character variables are provided in several program listings. Compiling and executing the programs will be helpful in grasping the pointer concept. A careful examination of the program listings is recommended.

A valuable reference for further pointer study is:

C Primer Plus by Mitchell Waite, Stephen Prata, and Donald Martin of "The Waite Group." Published by Howard W. Sams and Company. Revised edition 1987. ■

If You Don't Contribute Anything....

....Then Don't Expect Anything

TCJ is User Supported

The ZCPR3 Corner

by Jay Sage

The main topic for this column will be the second installment of the discussion of ZFILER, the Z-System filer shell (Yes, I'm going to fool you all by actually doing as I promised last time!). As usual, there are several other items I would like to discuss briefly first. The original list included the following: (1) a Z-Node update; (2) a hint on patching those hardware-specific utilities provided by computer manufacturers that don't work right under NZ-COM so that they will work; (3) my views on the appropriate way for Z-System programs to be coded for compatibility with various stages of evolution of ZCPR3; (4) an update on making PRL files without a PRL-capable linker; and (5) a suggestion to programmers for how to deal with bad-directory-specification errors under Z-System. As usual, including all this material put TCJ's ink supply at risk, and I had room only for the first two items. Now that I have finished the article and am coming back to hone this section, I also have to add that I did not have room to complete the ZFILER discussion; the topics of customization and configuration will have to wait until another time.

Z-Node Update

As I mentioned in a previous issue, I have been hard at work trying to survey the Z-Node remote access systems (RASs) and to revitalize the network. It was Echelon's creation of that network that first got me started as a Z-System activist, and I continue to feel that it is the single most important source of mutual support for users and developers of the Z-System.

My list of currently active nodes is reproduced in Listing 1. I have added three new columns to Echelon's original format. The one on the far right shows the last date on which operation of the system was verified. The column to its left indicates for nodes accessible by PC-Pursuit the code for the outdial city and the highest bit rate supported for that city.

At this point I have at least attempted (usually several times) to call every North American Z-Node on Echelon's old list. Where contact was made, I requested that the sysops register with Z Systems Associates, and the ones who have done so are designated by an "R" in the leftmost column. For this listing I have retained a number of systems that seemed still to be interested in the Z-System but have not yet registered. However, if I do not hear from them, they will be dropped from the next list. So, if you use one of those nodes (or one of the nodes I have already dropped), please let the sysop know that you want him to continue as a Z-Node, and suggest that he delay no longer in registering. Once we have all the sysops' names and addresses, we can start to think about things like a software distribution chain to make it easier for the nodes to stay current with Z-System software developments. Many of the boards I called had only very old versions of programs.

I would like to extend a special welcome to several new Z-Nodes, and I look forward to doing this in each column as more new nodes come on line. Bob Dean has for some time run the excellent Drexel Hill NorthStar system in Drexel Hill, Pennsylvania, just outside Philadelphia. When I saw what an enthusiastic Z-System supporter he was, I asked Bob if he would like to become a Z-Node. He was delighted and has joined the network as node number 6. Ted Harmon in Minneapolis has been working for some time at getting his node (#80) up, and I hope that he will be in regular operation by the time you read this. So far I have not succeeded in connecting with his node.

Bob Cooper in Ventura, California, is the newest node (#81), and from many voice conversations with him during the past couple of months I know how enthusiastic Bob is. His node is not in full scale operation. Since newly commissioned systems generally have fewer callers than established systems, their sysops would, I am sure, especially appreciate your calls.

Patching Programs for NZ-COM

As I described in an earlier column, NZ-COM creates a Z-System automatically from the host CP/M-2.2 system by setting up a virtual system underneath the original one and forwarding calls presented to the virtual BIOS (basic input/output system, the hardware-specific portion of the operating system code) to the "real" BIOS except for warm boots, which are intercepted to prevent a reloading of the host CP/M system. This produces a software environment that is indistinguishable from a manually installed Z-System, and all programs that adhere to CP/M or Z-System standards should run perfectly.

There is, however, a class of programs that generally do not follow those rules. These are most often utilities supplied by the manufacturer of the computer to perform special operations, such as configuration of the hardware. They usually make assumptions about the internals of the operating system code—in most cases, the BIOS—under which they are running. (Regrettably, they usually take no steps to verify that the environment is what they expect—see Bridger Mitchell's column in TCJ #36.)

Programs of this type generally do not run correctly under NZ-COM, just as they would not run correctly if the user rewrote his

Jay Sage has been an avid ZCPR proponent since version 1, and 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 nearly five years and can be reached there electronically at 617-965-7259 (on PC-Pursuit) or in person at 617-965-3552 or 1435 Centre St., Newton, MA 02159.

Jay is best known for his ARUNZ alias processor, the ZFILER file maintenance shell, and the latest versions 3.3 and 3.4 of ZCPR. He has also played an important role in the architectural design of a number of programs, including NZ-COM and Z3PLUS, the new automatic, universal, dynamic versions of Z-System.

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.*

or her BIOS without taking into account the assumptions the manufacturer made as to the location of certain data structures in the BIOS. (This same problem is less likely to occur, I believe, in a Z3PLUS Z-System running under CP/M-Plus, because Z3PLUS operates as an RSX, which was a fully defined system facility under CP/M-Plus. Manufacturers' configuration utilities are more likely to understand RSXs and operate correctly under them.)

There are two approaches to dealing with this challenge. In many cases the configuration utilities are used only when the system is initially set up (and the newly configured system is then stored on the system tracks of the boot disk). In other cases the configuration utilities are used only when the system is cold booted (i.e., powered up). These situations pose no problem, since the hardware utilities can be run under standard CP/M before the NZCOM command is issued to invoke the Z-System.

In some cases, however, the configuration utilities are needed on a more regular basis. Utilities for setting baud rates, screen attributes, or printer characteristics may fall into this class. These situations can present a considerable nuisance to the computer user, who easily becomes so accustomed to the facilities of Z-System that he or she nearly loses the ability to operate under vanilla CP/M. I can suggest two possible solutions here.

One approach is to put the configuration utility in a directory that is not on the path (or to give it a new name) and invoke it indirectly by way of an alias. The alias would initiate a SUBMIT batch operation, as described in the NZ-COM manual, that would first remove the NZ-COM system using the NZCPM command, then run the configuration utility under vanilla CP/M, and finally reload the standard NZ-COM system. (If you are very clever, you can probably make an ARUNZ alias figure out which of several standard versions of NZ-COM is running and automatically reload it.) This approach will give the appearance of successful operation under NZ-COM of a utility that actually cannot run under it. The main penalty is the extra time it takes to exit from and return to the NZ-COM system. There is also a problem if you have loaded a module (RCP, FCP, NDR, etc.) that is not the one in your standard configuration. It will be lost.

The second approach is to make the utility work properly under NZ-COM. In many cases I have been able to accomplish this without the source code for the utility by using the technique described below. But be forewarned; the technique will not always work.

Most of these BIOS-specific utilities determine the address of the data structures to be modified by adding an offset to the BIOS warm boot entry point whose address is obtained from the warm boot vector (jump instruction) stored at address 0000H in a CP/M system. Usually the instruction LD HL,(0001) is used to load the address into the HL register. The problem is that under NZ-COM this vector points to the NZ-COM virtual BIOS, and offsets from it generally fall right in the middle of one of the Z-System modules. Not only does the utility fail to make the desired change to the machine's real BIOS; it even corrupts some other code, resulting in behavior that ranges from unpredictably bizarre to instantly catastrophic.

The simplest corrective patch consists of replacing the LD HL,(0001) indirect load instruction with a LD HL,WBOOT direct load instruction, where WBOOT is the actual warm boot entry point address of the real BIOS. This kind of patch is performed by using some utility to scan the utility's code for occurrences of the three-byte sequence 2A (load HL indirect immediate), 01, 00 (the immediate address 0001H). ZPATCH is a natural candidate for performing the search, but it unfortunately uses 00 as its string terminator and thus cannot search for a zero byte. Perhaps Steve Cohen will eliminate this minor shortcoming in a future version of ZPATCH (hint, hint—I know you're reading this column, Steve).

The next step is to replace the 2A byte with 21, the direct load opcode. The other two bytes, 01 and 00, are replaced by the BIOS address that you have determined previously (perhaps by looking at the contents of memory location 0001H while running normal CP/M). The low byte is entered first in place of the 01 (it will always be 03). The second byte will be a some relatively large number, almost always with a first hex character of D, E, or F.

Blindly replacing sequences as described above does have its risks. Without careful inspection you cannot be sure that the sequences are being used to perform the assumed function. If you are an experienced coder, you can use a disassembler (such as the one built into debuggers like DDT and DDTZ) to examine the code. The LD HL,(0001) should be followed fairly soon by an ADD HL,DE or ADD HL,BC to add the offset to the BIOS structure to be modified. There is also always the possibility that the utility gets the address it needs in some other way (for example, LD A,(0002) will get the page address of the BIOS).

The procedure I just described "hardwires" the utility to a BIOS at a specific address. This is fine until you someday set up a new CP/M host system with a different BIOS starting address or until you give this modified version to a friend with a different BIOS. By then you will have forgotten all about these patches and will be pulling your hair out trying to figure out why the utility that worked perfectly before is now misbehaving. By then you will also have forgotten exactly what was patched and will not know how to fix the utility.

A more sophisticated patch will allow the program to work with a BIOS at any address. This approach follows Bridger Mitchell's philosophy of "know your environment." The patch checks to see if it is running under NZ-COM and makes the changes only when it is.

Source code for this patch, which can be applied using the MLOAD utility, is given in Listing 2. There are several pieces of information that you will have to determine in advance and enter into the patch code. I have put all that information at the front of the patch using macros where appropriate. If you do not have a macro assembler, you can always put the material directly into the code where the macros are called instead.

First, as before, you have to determine all the addresses at which indirect loads from address 0001 have to be changed to direct loads. These values have to be placed in the patch address table in the patch code. Since the patch will be added to the end of the existing utility code, you will also have to determine that address. You can calculate this from the file size of the COM file in records as displayed either by STAT or by SD with the "C" option. Alternatively, you can read the COM file into a debugger and note the next free address it reports. This address must be entered as the value of the symbol PATCHADDR.

Most of the utility programs I have patched this way start at 100H with a jump to the actual working code. The destination address of that jump must be determined and entered as the value of the symbol STARTADDR. If the utility does not begin with a jump, then you will have to examine the code at 100H and determine the instructions that occupy the first three or more bytes. These instructions should be entered into the REPLACED macro in the patch. The address of the next instruction after the ones replaced should be entered as the value for STARTADDR.

Once you have put all the necessary data into the UTILPAT.Z80 source code, it should be assembled to a HEX file. Then the patch can be added to UTIL.COM to make NEWUTIL.COM by using the following command:

```
MLOAD NEWUTIL=UTIL.COM,UTILPAT
```

Be sure to save the original program, and test the new version carefully. One additional word of caution. Some utilities cannot be expected to work under NZ-COM no matter what you do. For example, a utility that takes the running CP/M system and writes it to the system tracks will fail because under NZ-COM the only part of the CP/M system that is still present is the BIOS. For the same reason, programs that try to patch the BDOS will fail.

ZFILER, Installment 2

Last time we covered most of the built-in functions and had left the macro commands for this time. One built-in function was also deferred, the option command "O", and we will take up that subject first.

The Option Command

When the option command letter "O" is pressed, a special options screen is displayed. Eleven operating characteristics can be changed from a menu with the following appearance (approximately):

```
A. single replace query          Y
B. group replace query           Y
C. archive replace query         N
D. verify query                  Y
E. verify default                Y
F. suppress SYS files            Y
G. sort by file name             N
H. set copied file attributes    Y
I. use dest file attributes      Y
J. archive destination           Y
K. search path for CMD file      N
```

We will explain the meaning of each of these options in a moment. First a few words about the mechanics. While the options menu is displayed, pressing the index letter at the left will cause the setting of the corresponding option to be toggled, and the new state will be shown in the column at the right. The listing above shows the initial state of the options in my personal version of ZFILER. When you are finished toggling options, just press carriage return to return to the main ZFILER menu. These option settings are stored in the ZFILER shell stack entry and will thus continue in effect through all ZFILER operations until the command "X" is used to terminate the shell.

The first three options concern how ZFILER responds when copying (or moving) files and a file of the same name already exists in the destination directory. Item A applies when individual files are copied (commands "C" and "M"); item B applies when a group copy is performed (commands "GC" and "GM"); and item C applies when performing an archiving operation (command "GA"). If the option is "YES", then ZFILER will prompt one before existing files are erased and give one the chance to cancel the operation for that file, leaving the existing file intact. If the option is toggled to "NO", then existing files will be overwritten without even a message.

The next two options affect the verification of the copied file in the destination directory. Item D determines whether or not the user will be asked about verification. If this option is set to "N", then the state of option E will determine whether or not verification is performed on file copies. If this option is set to "Y", then before each copy, move, group copy, or group move, ZFILER will put up the prompt "Verify (Y/N)?".

The next two options affect the way files are displayed on the screen. If item F is set to "Y", then files with the "system" or SYS attribute will be suppressed, that is, not included among the selected files on which ZFILER acts. This is a reasonable choice for this option, since the most common use of the SYS attribute is to make the files disappear from consideration during file maintenance and display operations. Item G on the options menu determines whether files are sorted first by name and then by type or vice versa. Changing this option is presently equivalent to the "A" command from the main ZFILER command menu.

The next three options concern how file attributes are treated when files are copied. One possibility is to create new files with a clean slate of attributes (that is, all attributes reset: not read-only, not SYS, not archived). This is what will happen when option H is set to "N" (but note option J, which may override this). When the attributes of the destination file are to be set, they can be set in two possible ways. If a file of the same name existed in the destination directory, then its file attributes could be used for the

SAGE MICROSYSTEMS EAST

Selling & Supporting the Best in 8-Bit Software

- New Automatic, Dynamic, Universal Z-Systems
  - Z3PLUS: Z-System for CP/M-Plus computers ($69.95)
  - NZ-COM: Z-System for CP/M-2.2 computers ($69.95)
  - ZCPR34 Source Code: if you need to customize ($49.95)
- Plu*Perfect Systems
  - Backgrounder II: switch between two or three running tasks under CP/M-2.2 ($75)
  - DateStamper: stamp CP/M-2.2 files with creation, modification, and last access time/date ($50)
  - DosDisk: Use DOS-format disks in CP/M machines, supports subdirectories, maintains date stamps ($30 – $45 depending on version)
- SLR Systems (The Ultimate Assembly Language Tools)
  - Assembler Mnemonics: Zilog (Z80ASM, Z80ASM+), Hitachi (SLR180, SLR180+), Intel (SLRMAC, SLRMAC+)
  - Linkers: SLRNK, SLRNK+
  - TPA-Based: $49.95; Virtual-Memory: $195.00
- NightOwl (Advanced Telecommunications)
  - MEX-Plus: automated modem operation with scripts ($60)
  - MEX-Pack: remote operation, terminal emulation ($100)

Same-day shipping of most products with modem download and support available. Order by phone, mail, or modem. Shipping and handling $4 per order (USA). Check, VISA, or MasterCard. Specify exact disk format.

Sage Microsystems East
1435 Centre St., Newton Centre, MA 02159-2469
Voice: 617-965-3552 (9:00am – 11:30pm)
Modem: 617-965-7259 (password = DDT)(MABOS on PC-Pursuit)

copy that replaces it. This is what will be done if option I is set to "Y". If option I is set to "N" or if there was no matching file in the destination directory, then the attributes will be set to match those of the source file.

Option J can set a special override for the archive or ARC attribute. If the option is set to "N", then the ARC attribute is treated just like the other attributes according to options H and I. If option J is set to YES, then the destination file always has its ARC attribute set.

There was at one time a great deal of controversy over the way the ARC attribute is handled under ZFILER. At one time it was always reset, so that the destination file would be marked as not backed up. Another school of thought asserted that, on the contrary, the file was backed up, since there was a copy of it on the source disk from which the file was copied. That latter argument made considerable sense in the case of copying files from a master disk to a RAM disk before a work session. Here it was certainly important to start with all files marked with the ARC attribute so that one could easily tell at the end of the session which files had been modified so that they could be copied back to the permanent storage medium.

All in all, I never understood this controversy. Both approaches clearly have merit, and since ZFILER supports both, I saw no reason for all the argument. In a future version of ZFILER, I think I would like to add a flag word that would indicate which drives should automatically set the ARC flag when the J option is set to YES. That way, the option could be made to apply to RAM drives only.

The final item on the option menu, option K, determines how the macro command file ZFILER.CMD (see discussion below) will be located. There are two choices. If option K is set to YES, then ZFILER will look for it first in the currently displayed directory and then along the entire ZCPR3 search path. This option is useful if one wants to have different macro command files that apply to specific directory areas. Alternatively, if option K is set to NO, then ZFILER locates the CMD file without using the path. Depending on how ZFILER is configured (this will be discussed another time), the file will be sought either in the root directory of the path (the last directory specified on the search path) or in a specific drive/user area coded into ZF.COM. This alternative results in faster operation, especially if the specified directory resides on a RAM disk.

The options controlled by the option menu can also be permanently changed in the ZFILER program file using a patching utility like ZPATCH. In the first page of the file, you will see the ASCII string "OPT:". The eleven bytes following this string contain the startup values for the eleven options. Patch a byte to 00 for NO or FF for YES.

Listing 1: Z-NODE list

```
                         Z-Node List #49
                 Sorted by State/Area Code/Exchange


Revised Z-Node list as of December 31, 1988.  An ''R'' in the left column
indicates a node that has registered with Z Systems Associates.  Report
any changes or corrections to Jay Sage at Z-Node #3 in Boston (or by mail
to 1435 Centre St., Newton Centre, MA 02159-2469).


  NODE      SYSOP            CITY          STATE  ZIP    RAS Phone     PCP      Verified

   57 Steve Kitahata    Gardena           CA  90247  213/532-3336  CALAN/24  11/01/88
R   2 Al Hawley         Los Angeles       CA  90056  213/670-9465  CALAN/24  12/11/88
   35 Norman L. Beeler  Sunnyvale         CA  94086, 408/245-1420  CASJO/12  11/01/88
   25 Douglas Thom      San Jose          CA  95129  408/253-1309  CASJO/12  11/01/88
   35 Norman L. Beeler  Sunnyvale         CA  94086, 408/735-0176  CASJO/12
R   9 Roger Warren      San Diego         CA  92109  619/270-3148  CASDI/24  12/11/88
R  66 Dave Vanhorn      Costa Mesa        CA  92696  714/546-5407  CASAN/12  10/30/88
R  81 Robert Cooper     Lancaster         CA  93535  805/949-6404            12/29/88
R  36 Richard Mead      Pasadena          CA  91105  818/799-1632            11/01/88

   27 Charlie Hoffman   Tampa             FL  33629  813/831-7276  FLTAM/24  10/30/88
R  15 Richard Jacobson  Chicago           IL  60606  312/649-1730  ILCHI/24  11/01/88
R   3 Jay Sage          Newton Centre     MA  02159  617/965-7259  MABOS/24  12/31/88
   80 Paul Harmon       Minneapolis       MN  55407  612/560-9122  MNMIN/12    down

R  33 Jim Sands         Enid              OK  73703  405/237-9282            11/01/88
R  58 Kent R. Mason     Oklahoma City     OK  73107  405/943-8638

R   4 Ken Jones         Salem             OR  97305  503/370-7655            09/15/88
   60 Bob Peddicord     Selma             OR  97538  503/597-2852            11/01/88
R  24 Ben Grey          Portland          OR  97229  503/644-4621  ORPOR/12  12/25/88

R   6 Robert Dean       Drexel Hill       PA  19026  215-623-4040  PAPHI/24  12/10/88
R  38 Robert Paddock    Franklin          PA  16323  814/437-5647            11/01/88

   77 Pat Price         Austin            TX  78745  512/444-8691            10/31/88
R  45 Robert K. Reid    Houston           TX  77088  713/937-8886  TXHOU/24  10/03/88

   12 Norm Gregory      Seattle           WA  98122  206/325-1325  WASEA/24  11/01/88
R  78 Gar K. Nelson     Olympia           WA  98502  206/943-4842            09/10/88

R  65 Barron McIntire   Cheyenne          WY  82007  307/638-1917            12/12/88

R   5 Christian Poirier Montreal Quebec    H1G 5G5 CANADA  514/324-9031      12/10/88
R  40 Terry Smythe   Winnipeg    Manitoba  R3N 0T2 CANADA  204/832-4593      11/01/88
   18 Bruce Smith    Mississauga Ontario   L5E 2E5 CANADA  416/823-4521

   26 Robert Kuhmann  Belle Etoile, par St. Martin de la Brasque
                      84760 FRANCE, 11-33-90-77-60-15 (from USA)

   69 R.C. Page      Waltham Abbey, Essex, EN9 3EE, ENGLAND

R  62 Lindsay Allen  Perth, Western AUSTRALIA 6153      61-9-450-0200       12/21/88
   50 Mark Little    Alice Springs, N.T. AUSTRALIA 5750  61-089-528-852
```

ZFILER Macros

Although ZFILER can accomplish many tasks using its built-in functions, its real power comes from the macro facility, which allows it to be extended to include any functions that can be performed using combinations of other programs. This is where ZFILER really makes use of its power as a shell. First I will describe how the macro facility is used, and then I will describe how the user defines the macro functions. As with the built-in functions, macro functions can operate either on single files or on groups of files. The single-file macro facility is well developed and was already present in nearly the same form in VFILER; the group macro facility is new with ZFILER and has not been fully developed yet.

Invoking Macros

One way to initiate a macro operation on the pointed-to file is to press the macro invocation key, which is normally the escape key. A prompt of "Macro:" will appear after the normal ZFILER command prompt. At this point you have several choices. If you know the key corresponding to the macro you want to run, then you can simply press that key. ZFILER will then construct a command line and pass it on to the command processor for execution. If ZFILER is configured for instant macro operation (it generally is), then macros associated with the number keys "0" through "9" can be initiated without the macro invocation key; the number key entered alone at the main ZFILER command prompt will generate the macro function.

If you press the macro invocation key a second time, a user-created help screen will be displayed. This screen generally lists the available macro functions. You can now press the key for the desired function, or you can press carriage return to cancel the macro operation and return to the main ZFILER menu. The help menu screen will also be displayed if you press the "#" key. This is a holdover from VFILER and arises in part because of the structure of the file in which the macros are defined (more on this shortly).

Group macros are invoked in a similar way from the group function command line. After you have tagged a group of files, press the "G" key to enter group mode. The prompt will list only the

Listing 2: Utility patch source code

```
; PROGRAM:  UTILPAT -- Utility Patch
; AUTHOR:   Jay Sage
; DATE:     November 30, 1988

; This patch *may* allow BIOS-specific utilities to
; work under NZ-COM.

; The address below must be set to the first free byte
; after the utility's code.

patchaddr       equ     0000h

; The address below must be set to the address at which
; execution will continue after the patch code is
; executed.

startaddr       equ     0000h

; The following macro should be filled in with any opcodes
; that were at address 100H and following and had to be
; replaced by the jump to the patch code.  If the utility
; began with a JP STARTADDR instruction, then leave this
; macro blank

replaced        macro
                                ; put instructions here
                endm

; The macro below is used to enter the list of addresses
; at which LD HL,(0001) instructions appear that must be
; patched.  Replace the symbol ADDR1 by the first such
; address and insert additional similar lines for any other
; addresses to be patched.
;
addrlist        macro
        dw      addr1           ; Repeat for each address
                endm

ldhl    equ     21h             ; Load-hl-direct opcode
offset  equ     5ah - 3         ; Offset from BIOS warm boot entry
                                ; ..to the NZ-COM signature

; The following code will be patched in at address 100h to
; vector control to the patch code.

        org     100h

        jp      patchaddr

; The actual patch code begins here.

        org     patchaddr

; If NZ-COM is running, HL will now contain the ENV address.
; We will need this later, so we save it.

        ld      (envaddr),hl

; Now we must find out if NZ-COM is running.  This is done
; by looking for its signature ''NZ-COM'' at a specific offset
; from the virtual BIOS warm boot entry point.  If this
; signature is not found, the patch can be skipped.

        ld      hl,(0001)       ; Get possible virtual
                                ; ..BIOS address
        ld      de,offset       ; Offset to signature
        add     hl,de
        ld      de,signature    ; Point to what the
                                ; ..signature should be
        ld      b,sigsize       ; Length of signature
sigloop:
        ld      a,(de)
        cp      (hl)            ; Check character
        inc     de              ; Advance pointers
        inc     hl
        jr      nz,exitpatch    ; Jump if not NZ-COM system
        djnz    sigloop         ; Loop through signature

; We get here if NZ-COM is running.  Now we must make the
; necessary patches to the utility.  First we must determine
; the address of the warm boot entry to the real BIOS.
; NZ-COM keeps the page address at offset 2 into the ENV.

envaddr equ     $ + 1
        ld      hl,$-$          ; Filled in by code above
        inc     hl              ; Advance to CBIOS page
        inc     hl
        ld      a,(hl)          ; Get page of CBIOS
        ld      (cbiospage),a   ; Put it into code below

; Now we patch in the changes to the utility code.

        ld      de,table        ; Point to address table
        ld      b,tablesize     ; Addresses in table
patchloop:
        ld      a,(de)          ; Low byte of address
        inc     de              ; Advance pointer
        ld      l,a             ; Put in low byte of HL
        ld      a,(de)          ; High byte of address
        inc     de              ; Advance pointer
        ld      h,a             ; Put in high byte of HL
        ld      (hl),ldhl       ; Patch in direct-load opcode
        inc     hl
        ld      (hl),03H        ; Low warm boot address
        inc     hl
cbiospage equ   $ + 1
        ld      (hl),0          ; Filled in by code above
        djnz    patchloop       ; Loop through address list

exitpatch:

; Here we have to exit from the patch and resume execution
; of the original utility.  First we execute instructions, if
; any, replaced by the patch intercept code.

        replaced                ; Macro

; Then we jump to the utility code.

        jp      startaddr

; The following is what the NZ-COM signature should look like.

signature:
        db      'NZ-COM'
sigsize equ     $ - signature

; We put the table of addresses to patch here.

table:
        addrlist                ; Macro with address list
tablesize       equ     ( $ - table ) / 2

        end
```

built-in group functions, but if you press the macro invocation key, you can proceed as described above for single-file macro operations, except that the macro function will be performed on each of the tagged files.

The group macro facility works a little differently than the single-file macro facility. Since the command line would generally not be long enough to contain the commands for all the tagged files, the group macro facility works by writing out a batch file for processing by ZEX or SUBMIT. In this way there is virtually no limit to the number of files on which group macros can operate.

There are many configurable options (described below) that are associated with the group macro operation. These include the name of the ZEX or SUB batch file, the directory to which it is written, and the command line that ZFILER generates to initiate the batch operation. The NZ-COM version of ZFILER uses a file called ZFILER.ZEX and the command line "ZEX ZFILER". The Z3PLUS version, under which ZEX will not run, uses a file called ZFILER.SUB and a command line of "SUBMIT ZFILER".

Since macros (and the main menu "Z" function) work by passing commands to the command processor, file tags will be lost in the process, and when ZFILER resumes operation, it starts afresh. In a future version of ZFILER, I hope to preserve the tag information by having it optionally written to a temporary file (the shell stack entry is far too small) and read back in when ZFILER resumes.

Defining Macros—The CMD File

Now let's learn how to define the macro functions we want. As I indicated earlier, the macros are defined in a file called ZFILER.CMD (the ZFILER ComManD file). In the version of ZFILER distributed with NZ-COM and Z3PLUS, the CMD file is searched for in the root directory of the ZCPR3 command search path. As described earlier, the option menu allows the entire path to be used. There are also some additional configurable options that will be discussed another time. You must be sure to put your ZFILER.CMD file in the appropriate directory. If the file cannot be located, you will still get the macro prompt, but, after you have specified a macro key, the error message "ZFILER.CMD NOT Found" will be displayed.

The ZFILER.CMD file is an ordinary text file that you can create with any editor or wordprocessor that can make plain ASCII files (WordStar in nondocument mode, for example). The CMD file has two parts. The first part contains the macro command definitions; the second contains the help screen (described earlier).

In the first part of the CMD file, each line defines a macro. The character in the first column is the key associated with that definition (case does not matter). Macros can be associated with the 10 number keys, 26 letter keys, and all printable special characters except for "#" (explained below). The space character and all control characters are not allowed. Owing to an oversight, the rubout character can be associated with a macro!

After the character that names the macro there can be any number of blanks (including zero). If the first non-blank character is "!", then the "strike any key" (shell-wait) prompt will appear before ZFILER puts up the file display after a macro command is run. This should be used whenever the macro will leave information on the screen that you will want to read. After the "!" there can again be any number of spaces. Any remaining text on the line is taken as the script for the macro command.

The second part of the CMD file starts when a "#" character is found in the first column (hence the exclusion of that character as a macro name). Once that character appears, all remaining text, including text on the line, will be used as the help screen. Since ZFILER will add some information to the display (the name of the pointed-to file and a prompt), you will generally want to keep the help screen to no more than 20 lines, including an extra blank line at the end for spacing. With some experimentation you will get the hang of designing this screen.

Macro Scripts

ZFILER macro scripts are similar to those in ARUNZ and in the other menu shells (MENU, VMENU, FMANAGER) in that parameter expressions can appear. The critical parameters—the ones that implement functions that cannot be achieved any other way—are those that convey information about the directory currently displayed by ZFILER and about the pointed-to file. Parameters consist of a "$" character followed by one of the characters listed below.

```
User prompt parameters
          '    User input prompt
          ''   User input prompt
Parameters for directories
  - currently displayed directory
          C    DIR form
          D    Drive letter
          U    User numbvr
  - home directory (from which ZFILER was invoked)
          H    DU form
          R    Home DIR
Parameters for pointed-to file
          P    Full information (DU:FN.FT)
          F    File name (FN.FT)
          N    File name only
          T    File type only
Special parameters
          !    GO substitution indicator
          $    The dollar character
```

The parameters are listed in a special order above, and we will explain that later. First we will just present the meaning for each parameter.

The parameter expressions $" and $' are used to display a prompt message to the user and to read in a response string. Single and double quotes are equivalent. Once the prompt parameter has been detected, all subsequent characters up to one of the quote characters are displayed as the user prompt. Thus, if I am not mistaken, there is presently no way to put either quote character into the prompt. The end of the line or the end of the file will also terminate the prompt.

No special character interpretation is performed while expanding the prompt. If you want to make fancy screens, you can include escape sequences and some control characters (obviously carriage return won't work). In the future, ZFILER should be enhanced to provide a means to generate all control characters, to allow special characters to invoke screen functions based on the current terminal definition, and to expand directory and file parameters in the prompt.

Now for the directory parameters. Parameters C, D, and U return information about the currently displayed directory, while H and R return information about the home directory, the one from which ZFILER was originally invoked. PLEASE NOTE: macros always operate from the home directory. The reason for this is that ZFILER can display directories with user numbers higher than 15 even when it is not possible to log into these areas. If you want to operate in the displayed directory, then your script must include an explicit directory-change command of the form "DU:" at the beginning (or "$C:" if your system requires the use of named directories) and a command of the form "$H:" (or "$R:") at the end.

One special note about the parameters that return directory names. If the directory has no name, then the string "NONAME" is returned. This will presumably not match any actual name and will lead, one hopes, to a benign error condition. These parameters are included only for systems that do not allow directories to be indicated using the DU form (I hope that few if any systems are set up this way).

Now we come to the four file name parameters. They allow us to generate easily the complete file specification or any part of it. Note that "$F" is not quite the same as "$N.$T". The latter always contains a dot; the former does not if the file has no file type.

Finally, we have two special parameters. "$$" is included to allow a dollar sign character to be entered into the script. "$!" is a control parameter that is used only when a group macro is executed. If it is placed immediately before a token (string of contiguous characters), then that token will be replaced by the string "GO" on all but the first expansion of the script. This allows group macro scripts to operate faster by avoiding repetitive loading of a COM file. It must be used with great care and consideration, however, for reasons that I will not go into here.

Rules for Script Expansion

ZFILER follows a specific sequence of steps when expanding a script, one that gives it a special feature that, I would guess, few users are aware of. The first step in the expansion is to process only the user-input prompt parameters, substituting for the prompt whatever the user entered in response. This results in a modified script that is then processed by the second step in the expansion. Because the expansion is handled this way, the user input **can include ZFILER script parameters!** Thus the script can be used to write a script. You will see an example of this later.

The second step in the expansion is to substitute values for the directory parameters, which are a kind of constant. They do not change as a function of the pointed-to file. Finally, in a third step, the remaining parameters are expanded. For group macros, this final step in the expansion is repeated for each of the tagged files. The file parameters are expanded differently for each file, and, starting with the second tagged file, the "$!" parameter causes "GO" substitution.

```
                    Listing 3: Example ZFILER.CMD file.

Q   ql $p

U ! if $t=?q?;$!sys:uf $p $d$u:;else;$!sys:uncr $p $d$u:;fi

S ! $!sfa $p $'' SFA Options (/o,o.. o=ARC,-ARC,R/O,R/W,SYS,DIR): ''
K ! $d$u:;$!crunch $f $''Destination directory (DU:) -- '';$h:
B   $d$u:;crunch $f $''Destination directory (DU:) -- '';sfa $f /arc;$h:

M ! /move $p $''Destination directory for move: ''

X ! $d$u:;:$n $'' Command Tail: '';$h:
Z ! $d$u:;$'' Command to perform on file: '' $f $'' Tail: '';$h:
Ø ! $''Enter ZFILER macro script: ''
#                   SAMPLE ZFILER MACROS FOR TCJ

Ø. on-line macro        A. set Archive bit                N. NULU
1. LPUT                 B. Backup (cr/sfa)                O.
2. Z8ØASM to COM        C. CRC                            P. Protect
3. Z8ØASM to REL        D. Date display                   Q. QL
4. Compare Files        E. Edit                           R.
5.                      F.                                S. SFA
6.                      G.                                T. Type
7.                      H.                                U. Uncompress
8.                      I.                                V. VLU
9.                      J.                                W.
                        K. Krunch                         X. eXecute
                        L. LDIR                           Y.
                        M. Move                           Z. run command

$!  ZEX 'GO'            $D  DRIVE                $P  DU:FN.FT    $F  FN.FT
$''..''  PROMPT           $U  USER                 $N  FN          $T  FT
$'..'  PROMPT           $H  HOME
```

Macro Examples

Listing 3 shows an example of a ZFILER.CMD file, one designed to illustrate some techniques of macro writing. While writing this article, I discovered that one can include blank lines as shown to make the CMD file easier to read. The help screen part of the listing is taken from my personal script file (which, I have to confess, I have not really worked very hard at). The macro definition part of the listing includes only a few of the definitions.

The macro "Q" is included to illustrate a very simple, but useful, type of macro. It invokes the very powerful file typing program QL (quick look) on the pointed-to file. This is handy when you want more powerful viewing capability than that offered by the built-in "V" command. QL can handle crunched files and libraries, and it can display text or hex forward or backward.

Macro "U" uncompresses a file. It illustrates a more complex script that involves flow control and parameters that extract individual components of the pointed-to file name. It tests the file type to see if the middle letter is "Q" or "Z". In the former case, it unsqueezes the file; in the latter, it uncrunches it. The uncompressed file it put into the source file's directory.

Macros S, K, and B illustrate the use of input prompting. The first one allows the user to specify the file attributes to be set. Note that the prompt includes a helpful reminder of the syntax required by SFA.

Macro K crunches files to a user-specified destination. It also illustrates how one logs into the currently displayed directory. I do this here so that a null answer to the prompt (i.e., just a carriage return) will result in the crunched files being placed in the currently displayed directory rather than in the home directory, as would otherwise be the case (since that is where the macro runs from, remember). As a result, however, this macro will not operate properly in user areas above 15 under BGii or versions of the command processor that do not allow logging into high user areas.

Macro B performs a slightly more complex function. It not only compresses the pointed-to file to a specified destination directory, but it then marks the source file as having been backed up. A combination of the group archive built-in command (to tag files that need backing up) and a group macro B (to perform the backup) gives the ZFILER user a way to back up files in crunched form on the backup disk.

Macro M is included to show that a ZFILER macro, when it needs to do something more complex than it is capable of doing all by itself, can pass the task to an ARUNZ alias. The MOVE alias first determines whether the source and destination are on the same drive. In that case, MOVE.COM is used to perform the move. Otherwise, the source file is copied to the destination and then deleted. What we have, therefore, is a MOVE command that frees the user of the responsibility of worrying about which drives are involved—another example of how Z-System can free you from considerations that need not concern you, that do not require human intelligence to decide.

The final three macro examples are execution macros. Macro X causes the pointed-to file to be executed. A more sophisticated version might check to make sure that the file type is COM. I opted for

(Continued on page 31)

Information Engineering
Basic Concepts
by C. Thomas Hilton

"It helps to know what you want to do before you make the attempt."

Such a statement, as an opening comment, may seem a little silly to some of you. I can assure you it is anything but silly. One of my clients takes great pleasure from assisting others in the development of information systems. The terminology is there, unfortunately, the client has not been able to present any form of definition or guide for his own project, seven months after retaining professional assistance.

I am not belaboring the obvious. A great deal of time and money is wasted by refusing to deal with the obvious. More often than not, the layman can accomplish any project without professional help, if he is willing to give some thought as to what is really needed.

This issue is intended to convey a few rudimentary concepts. The discussions may be a bit pedestrian for some of you. If you find this to be the case, feel free to jump over them, returning to the introductory section when needed.

Field Definition Worksheets

The field definition worksheet is to Information Engineering, (IE), what a flowchart is to large-project programming. Before we can even begin to think of the software development tools we will use in a project, before we think of the hardware to specify for the project, and long before we think about hiring professional help, a study of our project's needs must be done. Knowing what is to be accomplished is the responsibility of the client, not the developer.

"Knowing what is to be accomplished is the responsibility of the client, not the developer.

The first step in any project begins with an idea. The Field Definition Worksheet, (FDW), is nothing more than a simple list of data types, called "fields" that describe a piece of data, which, when combined with other fields, creates a record, or collection of related information.

Figure 1 shows how you might create a Field Definition Worksheet, the first and most important step in any information system.

Figure 1: A field definition worksheet

F I E L D   D E F I N I T I O N   W O R K S H E E T
Project:____________________

Name	Type	Meaning	Source

The Name Field

The first area, shown in Figure 1, is the "Name Field." This area is where you put the name of the data field you are thinking about. At this point clarity is more important than correctness, when deciding upon the name of the field. Select a name that clearly relates to the data this field is to contain. Keep in mind that some software tools cannot tolerate long field names. Some tools require that field names be no more than eight characters in length. Many products allow field names longer than eight characters. But, select a field name that can later be abbreviated if the need arises.

The Type Field

The second area, of the FDW, will describe the data type of the field of information you are thinking of. The basic data types, common to all products, are numerical, character string, and date.

Numerical Data Types

A field declared as numerical should consist only of numbers. Alphabetic characters cannot be used in a numeric data field. A part number such as "A-256" could not be held in a numerical field. If your piece of information requires an alphabetic character, or punctuation of any kind, you will be required to use a character string type of field.

Many products deal only with integer or decimal data types. These are numbers which can be expressed as "whole" numbers, or numbers with a decimal point. These products may not be able to deal with negative numbers such as -3. To deal with these types of numbers you may also be required to use a character string to represent a negative, or less common numerical representation.

String Date Types

A "string" data type may consist of nearly any character you can access from the computer's keyboard. The "alphanumeric," "character," or "string" data type may be considered as a generic or "universal" data type. If there is some confusion over an actual data type, as there may be at this stage of our exploration, use an alphanumeric character string data type. All software products understand this designation.

Date Date Types

The "proper" way to represent a date value is matter of some controversy. The easiest way to get into trouble, and lose data when changing tools, is to use a "date" data type. Nearly every product has its own way of dealing with dates. Dates can be represented by "universal" character strings. If you are confident that you will not be changing database products, then there is little to lose from using date types anytime a date field is required. Most products will be able to convert a date type to an alphanumeric data type for export to another format. When in doubt, check your product's manuals, or use an alphanumeric data type.

The Meaning Field

The "meaning" field of the form will save you many questions, delays, and misunderstandings. Keep in mind that information on the worksheet may appear obvious to you, but not to someone else. The meaning of a piece of data is valuable in deciphering a field name, selecting a final data type, and lends much to the understanding of the purpose of your data.

The Source Field

The final area of the form is where you consider where you will acquire the data. Ask yourself, "where will this data come from?" Everyone has a tendency to be too creative when defining fields for their information system. Even some well meaning manuals tell you to include every field you can think of. This is not wise advise to follow. If you can create the data within your own department, or create it yourself, then you have little to concern yourself about. If you have to look beyond yourself, or your department, then you will have to ask yourself if you are willing, or able, to afford the time and effort to seek out the data from an external source. In a corporate or bureaucratic structure, politics may come into play, denying you access to data. If you cannot consistently acquire data, then it is said that you "cannot support" it. You would be well advised to seek another way of dealing with a desired data concept. In one client's application I was forced to delete over one hundred data fields. The client had neither the desire, nor the wherewithal to acquire the data. The firm rule for this area of the form is, "If you cannot support the data for every record in your database, you cannot include the field!" Do yourself a favor, take these words to heart.

"If you cannnot support the data for every record in your database, you cannot include the field!"

Developers Get Paid For Thinking

The commercial developer is paid for thinking, and of course a little obligatory development work. Most often the professional's task is only to force the client to think! There is little cause to pay someone else to do your thinking for you!

With the FDW worksheet concept fresh in mind, add all the data fields you feel are required in your information system. As you are developing the first version of the worksheet, keep a mental priority tuned to the clarity of field names, and their meanings.

The Demon Called Redundancy

Let us assume you have your basic field list defined. Look now at all the fields together, as opposed to individually. The first pass through the design process required that you be concerned with having enough information to accomplish the task at hand. The next review of your worksheet should be more critical.

Each field in the worksheet represents a portion of an input form which must be prepared, a report that must be constructed, computer processing time, and disk storage space.

Go over your field list and see if one, or more, fields actually represent the same information. Can several fields be combined to convey the desired information? By example, let us deal with the data concept of the type of diploma a person has received, as part of an employment application. The rough draft of the FDW will probably have the questions:

```
20. Did you finish High School (Y/N)?
21. Do you have a High School Diploma (Y/N)?
22. Have you received a G.E.D. (Y/N)?
```

With thought, it may be found that if more time were spent on the question, fewer fields may be required to represent the same information.

Each of the three questions in the example ask for a single character response, either a "Y" or an "N" to represent a positive or negative response. This representation "costs" three characters from three fields. "what-if" the structure of the concept were altered a little bit?

```
20. Tell us about your educational background.
    Have you received:
(D) A High School Diploma
(G) A G.E.D.
    (N) Neither
```

In this version of the concept three fields have been reduced to one, but still return the original information. This type of data design is compatible with optical document scanners. Data format compatibility with optical document scanners will be an important topic we will deal with later in this series.

Another example would be a person's, or product's age. Instead of asking for a birth date and current age, why not ask only the birth date? If the software tool has a means of working with dates, the "age" field can be calculated. If you elect to do calculations with dates, and need the result of the calculation, be sure to select a "date" data type for the source data field on your worksheet. Most systems do not "charge" you, in permanent storage, for calculated fields. They are only displayed in your forms and reports. They are never actually stored on disk. If you feel calculated fields are a clever way to reduce your information "overhead" you would be correct. But, use calculated fields wisely, and be sure to note in the "SOURCE" area of your FDW that the field is "not real" but calculated. Taking the time to be perfectly clear in noting which fields are calculated will save you much grief.

Going Generic

When your final field list is complete, and you are satisfied with it, take a few minutes to go over it one more time. See if there is any way to make things more compact, more useful, and more generic. Generic? What I mean here is to try to plan ahead.

Will your project "grow" to a point where you will need to move it, because the initial software cannot handle the amount of data you have collected?

Will you ever want to do research on your data?

Will your reports require graphic presentations?

Here is where the clever selection of field data types is critical. The primary pitfall of the novice designer is the random use of "date" data types. Do you have any date fields that you will NOT use in calculated fields? The best policy is to use a character string data type for anything that is recorded, but never actually used in calculations, if you are thinking of exporting your data to another database product.

Organizing Your Data Tables

Having defined the data fields you will want to use in your information system, the next step is to organize your data into data tables.

A data table is simply a collection of related information. The concept of the data table has become common in popular software product releases. The purest representation of the data table concept is found in PARADOX. It has been said, and I tend to agree, that all other table based products attempt to imitate the PARADOX system. Our discussions will directly refer to

PARADOX, though the concepts presented may be applied to other table based products.

The Relational Database Concept

The foundation concept of the relational database, and data table structure is simple: keep the data pure, concentrated in focus, and isolated. For example, in dealing with people, we would want to keep information that identifies the individual apart from all other data. While other data may relate to the individual, such as test results, we would not want to include the test results in the same table as the personal information. The relational data table may be compared to a properly designed paper filing system. Personal information goes into the personal file, test results into the test results file. If we just threw everything into a file with a client's name on it, information would be accessible only in relation to the individual. This is a fine system for collecting paper, and filling file cabinets. Yes, I know, "that is the way we have been handling information since dirt was new." This is something I face every day at the office! This way of handling information is also the single most important factor as to why it may take a person hundreds of man hours to answer a simple research question, and why his reports are never on time! The paper data has to be located, pulled from the individual's file, processed, and returned to the file. This is a wasteful process.

With a relational database, it is simple to have the computer run around and collect all the available information on an individual. It is equally as easy to deal with the collection of specific data that does not actually relate to any unique individual. "Paper Think" should become a dead art, never applied to Information Engineering.

By keeping data together and isolated, we can do research on individual data without wading through unrelated information.

How Many Fields Should be in a Data Table?

The general "rule of thumb" is that a data table should be reorganized if it contains more than 15 or 20 fields of closely related data.

It is better to have more tables, with fewer fields, than it is to create large tables of data. My personal preference is to keep data tables small enough so that all the data in a table may be seen on a single screen. This concept is seldom possible, but it is a good goal to strive for.

With products, such as PARADOX, you will discover that you can deal more effectively with tables with fewer fields. PARADOX makes it as easy to deal with many tables, with fewer fields, as it is to deal with a single table with many fields. There is no penalty for storing information about a single subject in many different tables. This is what a relational database is all about!

It should be obvious that if a product does not easily accommodate these concepts, then it not a product you want to work with.

This Issue's Example

The information system we will use as an example in this issue will concern the needs of a Career Development Service, located in a correctional institution. The client has thus far been unable to demonstrate a knowledge of his needs, or provide marginal direction. He has opted to retain professional assistance. All in all, we are left to our own devices. This example represents a true "worst case" scenario for the Information Engineer.

The client has indicated that his project has been funded with public moneys. This requires attention to several other factors which must be entered into the information system's basic design. The portion of the grant specific information which relates to our information system is shown in Figure 2. Other information in the document relates to the saving of copies of public announcements, hence is of little interest to us. What data we do not include in the primary data table will serve to define additional data tables.

The client also has access to data from a variety of test instruments which he wants included in his information system. We will deal with test data at a later time. We have enough to deal with, in this issue, regarding information needs which relate to persons.

Figure 2: Grant Specific Information.

Carl D. Perkins Vocational Education Act
Record Keeping

On-Line Information Required:

1. Minority Status
   Actual status such as Caucasian, NAI, etc.
   Yes or No (Minority)
2. Current Grade Level
   If student, then in what grade
   If not a student, then what level is highest achieved
3. Gender
   Specific M or F indicator is required
4. Handicapped
   Yes or No with field for description
5. Disadvantaged
   Yes or No with field for description (Educational Criteria)

We also know that the client listing will be well over a thousand individuals, once the system has been implemented. Other information gleaned from the client is that he will want to do research, is not very well versed in computer information systems, and we must not forget that the client has little idea what he expects to accomplish.

Selecting the Tool

QUATTRO would be a good tool were the numbers of clients fewer, as would REFLEX. The only problem with these two products is that they keep their data resident in memory. Hence, the size of each data file may not exceed the size of available memory. No matter how we break down our sets of data, eventually we would have as many files of data as we would fields of data in the files. Secondly, we have to accommodate operators of the system who have little experience with computers, or worse, feel they have the experience when they do not. If the operator forgot that every change had to be saved to disk, the session's work would be lost. As we are dealing with people, people's lives and futures, we dare not place such data into the hands of amateurs without caution & restriction.

CLARION would, at first, seem the obvious choice, save for one factor. The client is unable to relate to us what he wants his system to "look like," what it is supposed to do for him, or what functions of research will be desired. While many of you would feel that, ethically, we should ask the client to "get his act together" and call us when he knows what he wants done, we cannot do that. Our client's dysfunction is not that unusual when one is dealing with Information Engineering, as opposed to common data processing. When an application can be clearly defined, and can be "locked" into its general design concept, CLARION is a wondrous tool.

In my experience the only product that will meet the client's specifications, or lack of same, is PARADOX. The prime decision-making concepts I used in this selection are the need for research, an "open data structure" independent of client whims, tangents, and adaptability for use by unskilled operators. We will proceed from this tool selection of PARADOX.

The concepts presented may be indirectly implemented in Rbase, and the new dBase IV. There are some significant product differences which will have to be overcome, however. I have many other reasons for selecting PARADOX which I would be hard-pressed to justify, including a fondness for the product.

Figure 3: The personal information data table

```
F I E L D   D E F I N I T I O N   W O R K S H E E T
Project: CES - PERSONAL DATA
```

Name	Type	Meaning	Source
NUMBER	N	Client Reference Number	Locally Assigned
NAME	A20	Client's Full Name	Job Application
ACTIVE	A1	Is Client Record Active	Locally Assigned
ENTRYDATE	D	Date Of Entry Or Change	Locally Assigned
DOB	D	Stated Date Of Birth	Job Application
GENDER	A1	Client's Stated Gender	Job Application
ETHNICORIG	A5	Stated Ethnic Origin	Job Application
MINORITY	A1	Eligible Minority	Career Counselor
LOCATION	A9	Current Client Location	Available Records
JOB	A38	Current Client Job	Available Records
WAITING	A38	Waiting Lists	Locally Assigned

MOVING?

Make certain that TCJ follows you to your new address. Send both old and new address along with your expiration number that appears on your mailing label to:

THE COMPUTER JOURNAL
190 Sullivan Crossroad
Columbia Falls, MT 59912

If you move and don't notify us, TCJ is not responsible for copies you miss. Please allow six weeks notice. Thanks.

Figure 3 shows the first data table I would create for the client. Prisons are, for all intents and purposes, small townships. Every need and function of a small township is present. For this reason "normal" concepts apply, though they may be abbreviated somewhat. Let us look at the worksheet, pertaining to personal information, in some detail.

The Client Worksheet

It makes little difference as to the type of society where we would be able to apply this system. A client reference number would be required. It is also good design practice to assure that the client number is the first field in every data table. When the numbers of clients exceeds a certain level we find ourselves dealing with more than one "John Jones," and more than one "Jane Smith." In our client's application a client number has already been assigned by the Department Of Corrections. The issue here is the unique identification of the individual.

The NUMBER field uniquely identifies the client from all other clients with the same, or similar name.

There is, when using PARADOX, seldom the need to parse the NAME field into a first and last name. PARADOX will allow any examination of the NAME field in any way the client might require. Additionally, a full NAME field saves a few characters of data space, which contributes to a smaller overall file size. Generally speaking, most applications will not require more than 12 to 20 megabytes of disk storage space when designed intelligently.

We have also added two fields for data and record management. The first, ACTIVE, asks "is the record still active?" As with any project that will do research on its data, there must be a "data trail." We will want to keep test information in the system, but not necessarily the client's personal information. In normal office operation we do not want to clutter the file cabinets with "dead files." This field will allow us to scan for inactive records, and when put in a report, serve to notify the "paper clerks" of which files are to be removed from the "working set." The second field, ENTRYDATE, serves to note when an individual's information, or status has changed.

The DOB field serves the general needs of the client's personal information. Note that the DOB, (date of birth) field is marked as a "D" field, or DATE field. A date data type was used over a string data type as later we may want to perform some "date mathematics" with this field. Further, we see no real need to transfer the application to another product. If the need to export the data should arise, PARADOX easily translates its date data types into string data types using the RESTRUCTURE option.

The GENDER field is required, not only due to the grant specifications of the client, but because female prisoners will also be entered into the system. The female offenders are small in number compared to their male counterparts. They do not warrant a separate information system. While they may be housed in a separate institution, the law states that they must be dealt with in exact equality to male offenders. For the considerations of this data field, the location of the client is of little concern. The concerns of this field can be managed with a simple "M" or "F" entry sequence.

The information required in Figure 2 concerning racial/minority/ethnic concepts, must cause us as Information Engineers, to be a little creative. This field clearly states whether the individual is a racial minority. For the purposes stated by the client, any further "minority" data would be redundant. However, the apparent progress of our society is such that simply being a "minority" is not enough to make a person receive special consideration. In point of fact, every individual that would be entered into the client's system would legally be a minority without regard to their race. They are convicted felons, and our society has many programs dealing with rehabilitated felons. To serve the client's needs, we will provide an alphanumeric field suitable to house a variety of ethnic descriptors. We will also provide a field that will serve to indicate whether the client is actually eligible for any minority programs. Eligibility determination will be the responsibility of the client.

It is interesting to note, when dealing with the LOCATION field, that even though a prison may occupy a remote location, individual buildings within a "compound" actually have different street addresses. This only applies, however, to buildings which do not directly concern prisoners. All buildings in which prisoners are housed have a single mailing address. In this project a conventional address scheme would not work well, as all the individuals would have the same street address. Another interesting fact of our client's location is that every building, while having the same mailing address, has a unique building name. The name of the building, then, will have to serve as the mailing address. In a more standard application, the LOCATION field would be expanded into three alphanumeric lines to represent the individual's unique mailing address. Other than this "hairsplitting" of the concept, the LOCATION field is a reasonable one, suitable for any design. We have made the decision to abbreviate the critical data for reasons of disk storage space alone. Why have fields, that have no meaning, or allow the operator to get creative with data entry? We will deal more with the concept of creative data entry problems, and how to solve them, later in this article.

The JOB field, in the personal information data table, relates to the individual's current work assignment. Every individual is required to have a work, or school assignment. This data field serves the "current vocation" needs of a comparable "civilian" system. In the client's profession it is common to want to know if an individual is actually employed in a trade where they are qualified or were actually interested in their current vocation. Research on this subject is quite interesting, but will have to wait until a later time.

As a last minute addition, the client desired a generic field that could be scanned to indicate individuals waiting to be tested, or for an activity to begin, etc.

Summation of Concepts

The data in this table is directly concerned with the individual. While other information needs of the client are RELATED to an individual, they may not directly describe the individual as a person. We will not deal with this other data in this issue.

Several fields in the PERSONAL data table do not meet the primary design criteria of the preceding paragraph. There is a reason for their inclusion, however. All projects have their "nuisance" segments. The PERSONAL data table serves to deal with, and get out of our way, one of the client's nuisance applications.

Once the PERSONAL data table is built, it will be updated by a daily "Change Sheet," which notifies all departments in the prison of changes in an individual's status. Individuals may change housing units and work assignments often. Better put, when dealing with large numbers of people there are always changes being made. For obvious reasons, a prison environment tends to be quite fanatical about knowing where a prisoner is at any moment of the day.

For Practice

We will continue, briefly, discussing the PERSONAL data table's design and construction. Our discussions are not, however, theoretical in nature. The information system being presented is real. Because it is real, and not theoretical, we cannot "dummy" up data used in the research sections of the series. Remember the minor deity GIGO, "Garbage In, Garbage Out." It would be of little benefit to anyone, if the data were not real. We are required to keep the identity of individuals represented in this system confidential. The PERSONAL data table is the "key" table to an individual's identity. All other tables will have data records referred to only by the individual's "Client Reference Number." For these reasons of confidentiality, the PERSONAL data table, while made apart of the application disks available with this series, will not contain "live data." Qualified professionals in need of data not immediately available through this series may present a statement of need, in care of TCJ. Be sure to include your business telephone number, and where you can otherwise be reached during normal working hours.

Building the Personal Data Table Design

It must be assumed that you have some familiarity with PARADOX, or the table based system of your choice. If this is an incorrect assumption, then the reality of being forced to spend some time with the manuals will have to be attended to.

Create a data table using the fields shown in Figure 3.

Prepare a data entry form using the FORMS menu option.

Exit the FORMS menu option in the normal way.

Select the PARADOX "View" option from the PARADOX main menu. When asked for a table name, press ENTER to be presented with a list of available data tables, place the selector bar on the PERSONAL data table and press ENTER. You may also input PERSONAL, when asked for a data table, and press ENTER. Having entered the "View" mode, with the PERSONAL table, press F9 to enter the PARADOX "EDIT" mode.

Dataentry Validity Checking & Look-Up Tables in Paradox

As you may recall, part of the specification we added to this portion of the project concerned the fact that the system had to accommodate operators with little or no experience with computers, or database managers. This lack of experience tends to also include the desire to be creative in data entry. This is a diplomatic way of saying that the operator may be incapable of performing the same operation, in the same way, twice. In an attempt to thwart creativity in data entry, but without massive program code to "dummy trap" the entire system, simple validity checks need to be installed.

Setting the NUMBER Field

The NUMBER field will always house a five digit number. There are no legal exceptions in the client's reference number, at least in this century. Follow the following sequence.

1. Assure you are in the EDIT mode.
Press F10.
Instead of the main menu, a secondary menu will present available options when editing a table.

2. Select "VALCHECK" from the editing menu options.

3. From the VALCHECK menu select the "DEFINE" option.
Place the cursor on the NUMBER field.

4. When the definition menu appears, select the "PICTURE" option. Because we want PARADOX to accept a five digit number, enter ##### as a picture. This "picture" is to say that only numerical characters can be entered into this field.
The screen will display a message that the picture you have selected has been stored.

Repeat the sequence, steps 1 through 3. When the definition menu appears again, select the "REQUIRED" option. Indicate that this field, must never be left blank. The number field is the "thread" that will link all other data tables together. Without a proper entry in this field there will be no way to easily access an individual's data.

The NAME Field

The name field will be seldom used, other than to present the individual's name in a report of some kind. At this juncture, no special validity checks are required. Later we will present a PARADOX "script" to format the name field.

The ACTIVE Field

Having traveled through the VALCHECK sequence several times now, some familiarity with the process is assumed. For the ACTIVE field we want to assure the field is never left blank, that the "lowest value" is "N", the highest value is "Y", and that the default value is set to "Y". You may note that the ordering of the highest and lowest values are based upon alphabetic order.

Entry DATE and DOB

While we would like to assure that these fields always have data, we need not be fanatical about it. No special processing of these fields needs to be specified, at this time.

GENDER

The GENDER field needs to be somewhat "dummy trapped," so as to only allow the entry of only a single "M" or "F" character. This may be accomplished in the same fashion, in sequence and concept, as the ACTIVE field we did above.

ETHNICORIG, LOCATION, and JOB Look-Up Tables

The ETHNICORIG, LOCATION, and JOB fields may be dealt with together, as their process is identical. In order to use the TABLE LOOK-UP facilities of PARADOX, you must have previously defined and created data tables with the desired data entry field options. Each of the tables used for these functions, (ETHNIC, LOCATION & JOB), have a single field of data defined in them. The records in the respective tables are used to "fill-in" the related look-up fields in the PERSONAL TABLE.

From the DEFINE menu use the TABLELOOKUP option for each of the PERSONAL data table look-up entry fields. Be sure to assign the right table to the proper entry field. From the TABLELOOKUP option series select "JUSTCURRENTFIELD" followed by HELPANDFILL. With these options defined, when the fields are being edited, pressing F1 will display the look-up tables. Placing the cursor on the desired record in the look-up table, and pressing F2 will cause the selected record in the look-up table to be inserted into the entry field in the PERSONAL data table.

To test the look-up functions, assure that you are in the edit mode, whether using a data entry form or table view, and press F1. The look-up table should appear.

MINORITY Field

The minority field requires only a "Y" or "N" entry character. It may be left blank, but we should install a default entry of "N" just to ease the data entry process.

WAITING Field

The waiting field is a "junk" field. The client has great expectations of use for this field, but the actual use of it depends upon the operator scanning it regularly. Unless the client makes visible use of it, it will be deleted at a later date. We often must sate unrealistic whims of clients new to Information Engineering, (sigh).

Cleaning up Other Concepts For this Session

We have not gotten into the "mechanics" of the operation to a point that may totally satisfy you in this issue. There has been a great deal of critical, but fundamental, foundation information that had to be dealt with. Throwing all the basic ideas at you in a first installment of the series gets them out of the way. We will not want to deal with them in later issues. In future installments we will want to deal only with the tasks at hand, assuming the basics are understood.

The client will require a number of reports for this data table alone. PARADOX allows up to 15 individual reports per data table. This was another reason for selecting PARADOX for the client's project. We will not deal with the building of reports in this issue, if at all. The methods of report development vary greatly between products, making the topic unsuitable for a presentation of general information. If you have a particular question dealing with PARADOX reports, however, drop me a line in care of TCJ and we'll see what we can do to help.

In the next installment of this series we will create a job application table, and a table for storing SAT (Stanford Achievement Test Scores). The job application data will be developed for later use. The SAT data table will introduce basic research techniques, basic statistical analysis of data, selectivity functions, and demonstrate a simple method of producing "bar graphs," without the use of graphics or complex programming.

It is hoped that this first installment will get you thinking, and provide you with basic information. Once again, the information presented here is not intended to be a complete tutorial. Further, it will not be repeated in future issues, but built upon with new concepts.

Short Takes

Art is probably the most tactful editor in the world! His way of mentioning that I am a week past deadline is to send me time management software to "look at," and closing the cover letter with our complete production schedule. Well, there went my Christmas downtime! Oh well! If I were efficient no one could stand me!

I am sure most of you have seen the little inserts from POWER UP! which have been included in many of the shrink-wrapped trade magazines. I have never paid a great deal of attention to their products, other than to note that they were "cute." Well, I blew it again! Art sent "down" three selections from their product line for evaluation. While I don't have the space this time to go into detail, there are three products I consider worth your investigation, and investment.

Calendar Creator Plus

From the advertisements CALENDAR CREATOR PLUS looked like just another calendar generation program. It is more sophisticated than I had thought. Oh, it does six or seven different styles of calendars. It also allows you to create your own "overlays" for standard calendars. What they call "overlays" resemble the little holiday notes on commercial calendars. If your work area looks like mine, the calendar is covered with notes, appointments, and even deadlines I keep missing. This calendar program allows you to do everything you would normally do with a pen while talking on the phone, but with a touch of professionalism. Not only can you create functional things to put on the wall, but you can personalize them for your boss. This product has a very high REVENGE factor!

Quick Schedule

On those rare occasions when my boss has an attack of efficiency, and demands "project management software," I normally point to the REFLEX and REFLEX WORKSHOP manuals. REFLEX does the job well, but takes a little time to set up. QUICK SCHEDULE has the ability to do a readable "time chart" which, on first impression, seems to be quite usable. I can think of one application for which QUICK SCHEDULE is very well suited: Class Scheduling. I get real tired of plotting a class starting date and trying to visualize the twelve week's demands upon resources.

Grid Designer

A GRID DESIGNER? Designer? Design what? The darn thing comes with 233 examples on tap. Grids? Yes, it produces grids as well. It has calendars, appointment schedules, score card forms, rules, and more that I haven't had a chance to look at. The first thing that I recalled when looking at the product's "sample" menu was the time my boss was wondering what had happened to the accounting ledger forms that "were here somewhere!" Rather than explain that we threw them away last June, with GRID DESIGNER one needs only ask what style of accounting form he was referring to, printing a supply when he leaves the office for a moment. This product has screen design forms, FORTRAN coding forms, and just all kinds of stuff, and they all can be personalized. Of course one can also design a "grid" or two of their own, should one of the "samples" not suffice.

These POWER UP! programs are the stationary catalogs of the information age. They are very functional, and an asset to any office environment. They are highly recommended. To request a catalog, or place an order call 1-800-851-2917 or 1-800-223-1479 in California.

I can see the need to include many of these POWER UP! products in my Information Engineering toolbox. To keep information moving takes organization. These products are clever enough to get people to notice a project schedule. Getting them to follow it though, is another story. ■

Using ZCPR3's Named Shell Variables
Storing Date Variables
by Rick Charnes

I have really been enjoying myself lately with my Morrow running Z-System. I am slowly but surely developing a sense of myself as what Frank Gaude' used to call a "chipper" (one rung below a "hacker"), and I guess I'd consider myself a proto-programmer in my own right. In any case I am finding a great deal of delight in doing what I'm doing with ZCPR3. What I am producing is providing me with an wonderful feeling of creativity and productivity. I would like here to express my deep gratitude for all the people on the Z-Nodes in the last two years who have been such a strong part of my life and education. Their gentle and wise assistance has been a source of great joy, and I'd like to say how grateful I feel for their help during this exciting period of my life. I can only hope that what I have been sending out over the wires of the Z-Node network can in some small part discharge that debt and stand as testimony to their own good deeds.

It's interesting—I still don't know a "language," in the sense of assembler or Pascal or BASIC, except for the very rudiments; yet ZCPR3 itself provides such a system of tools that as a whole it comprises virtually a language in itself. It is in this programming "language" that I mostly write—and have the most grand time!

As far as tools I use, first and foremost of these I must say right off the top and without any hesitation whatsoever as regard its pre-eminent status among ZCPR3 tools offering a virtual "language," is ARUNZ. This magnificent program of Jay Sage's has given me untold hours of programming delight. Its sophistication, intricacy, power and sheer elegance are unparalleled in the Z world. It acts as a basis and foundation for virtually everything I do. I will not in this column describe its use, as Jay has done so in previous columns; suffice it to say that my ALIAS.CMD file, developed over a period of two years, is now 32k in length and next to my album of family photos would probably be the first inanimate object I would rescue from my house in case of fire.

I fervently hope by now you are all using either ZSDOS or DateStamper. I cannot emphasize enough how much datestamping has added to my computing. There are three datestamping-supporting utilities that have played a particularly important part in my computer use in recent months and have provided a foundation for me to branch off into all sorts of creative projects. They are: Ron Fowler's classic MEXPLUS; Carson Wilson's superbly useful ZSDOS directory program FILEDATE which allows you to sort by date, either most recent to earliest or vice versa; and—of course—ARUNZ. It is ARUNZ' date parameters that have allowed me to complete my latest and extremely fulfilling project, about which I now write.

I have a ZEX script that I use for backing up my hard disk. It's a good friend that I've been maintaining and lovingly looking after for the last year or so. A few days ago, while making some enhancements to it that I have found very exciting and to which I would like to devote my next column, an idea hit me. Whether the sun suddenly came out in heaven, the moon entered Aquarius, or the gods were just in a good mood I can't really know for sure, but I do know that the end result of that light bulb going off in my head has given me a very great deal of delight and I'd like to share it.

One nice thing about using a ZEX script as opposed to a prefab COM file for something like this is that you have complete control over it. The programming ability notwithstanding, you are free to implement any ideas that come to you. I've taken special care to put fancy graphic displays into the ZEX file, something that ZCPR3 programs are often sadly lacking. My idea: I wondered if I could write something that, before doing the actual back-up, would display to me the last date and time any given directory was backed up.

I thought to myself: what an idea! It would be quite amazing to pull something like this off. But—how could it possibly keep track of this information? Where would it be stored? Assuming I could store the information someplace, how could it survive a cold boot, not to mention a power-off? When the idea first came to me I had no idea how I could possibly do it—it seemed like something only an extremely sophisticated computer system could do. Yet I had a feeling somewhere deep inside that I could do it...

By the way, for those who don't get as excited as I about hard disk backups and knowing when the last one was performed, I will mention that I have used the same technique to create a STARTUP alias that displays to me on my screen, in beautiful graphics and reverse video, the last date and time I turned my computer on (or did a cold boot). Furthermore, using the same tools and concepts, I have created an alias that will tell you the named day of week of any date in 1989. If you want to know what day of the week your birthday falls on, this alias will do it. So everything you read here is applicable to these tasks as well.

I should add, by the way, that the first two of these applications display not only the time and date, not only the **name** of the day of the week ("Wednesday," etc.), but also the full name ("August") of the month. This all done without an external "program" written in any of the standard computer "languages," but rather entirely with the basic ZCPR3 utility toolset. I think it's pretty incredible.

One of the things I have always found eminently enjoyable about Z-System is the harmonious way the individual tools work together and their ability to pass messages and information to each other (it's always reminded me of Olympic relay runners passing the torch to each other...), and when I began thinking about how to accomplish this task I knew it was on this feature that any scheme would depend.

Oddly enough when I think about it now, I was first planning to store bytes of date information in the ZCPR3 registers. It's hard for me to imagine how I could have forgotten that the registers are zeroed out when the computer is turned off!

It didn't take me long to sketch out in my mind the broad conceptualization of how I was going to do it, though the actual details took much longer to think through. Those who have digested and assimilated my last article should be thinking already how it could be done. We of course do it with string (shell) variables. Indeed, that is the only way I imagine it would be possible. I'm going to assume here a minimum level of knowledge of the ZCPR3 shell variable system and the relevant tools for manipulating them as discussed in last month's column. Throughout this column I use the phrases "shell variable," "named variable," and "string variable" interchangeably.

The basic concept is to store and then access the dates as named variables inside SH.VAR. Each variable will store the date of last backup of each hard disk directory, one variable to one directory. Since there is no limit to the number of entries inside a VAR file this is no problem; we can easily fit 40 or 50 entries inside a small, 2k (on a floppy) or 4k (on a hard disk) VAR file. The backup procedure will first read and display this definition, then overwrite it with the current date and time, and finally do the actual backup.

I decided to give each string variable name a prefix of "BUDAT", standing for "BackUp DATe." In other words, the string holding the date information about hard disk directory A15: would be named "BUDAT15," and that holding information about my B3: directory would be named "BUDATB3". My idea was that ultimately I would have the contents of SH.VAR looking like:

```
VARIABLE        DEFINITION
NAME

budata0         Sunday, January 27, 1989  3:43 pm
budata15        Sunday, January 27, 1989  4:20 pm
budatb3         Thursday, March 9, 1989  9:17 pm
bjdatb6         Tuesday, March 14, 1989  7:22 pm
 .
 .
 .
```

etc. This is perfectly reasonable. There is nothing about the ZCPR3 shell variable system or the *.VAR file structure that would prohibit such string definitions; strings can be of any (reasonable) length, can contain spaces, numbers and punctuation, etc. The task at hand was: how to do it? What mad and crazy agglomeration of tools, ARUNZ aliases, and/or ZEX files could actually write, update and maintain such information into our humble SH.VAR file? Ah, life's challenges are so sweet...

I should say the most difficult, or at least laborious part of the entire procedure was working out a scheme for getting and displaying the name of the day of the week. I originally spent several hours on the project doing it without this information, and ultimately created the display:

```
LAST TIME THIS A15: DIRECTORY BACKED UP:
DECEMBER 15, 1987  11:03 p.m.
```

I found myself, however, quite frustrated not knowing what day of the week that was. All those numbers without a "Monday" or a "Thursday" in there just didn't feel right; something was missing. I went back to my hilltop and meditated some more. After several days without food or drink, I can now present to the world the entire procedure I developed.

It consists of an ARUNZ alias followed by my abovementioned BACKUP.ZEX script. Though I don't have room in this column to detail the quite lengthy ZEX script and will do so next time, only the first two lines are relevant to our task at hand. If you have any series of commands you use, or could use, to perform your backing up procedures you could simply insert them in a ZEX file after what I give here as the first two lines of my BACKUP.ZEX. If you do not need a hard-disk backup procedure on your system, I hope you can study this universal technique of storing and accessing dates and find another use for it such as displaying the last time your computer was powered on, as was mentioned above.

Without further ado, here's the alias and ZEX script.

```
BACKUP resolve lastback $hb %budat$hb
       shfile $dm$dy
       resolve zex backup budat$hb %$dd %month $dd,
             19$dy $dc:$dn $da
```

The first two lines of BACKUP.ZEX are:

```
shfile sh
shvar $1 $2 $3 $4 $5 $6 $7
```

I'll explain the first line of the alias later, even though this does the "read" of the date and the rest of the procedure does the "write." Although when the alias runs the read happens first, the whole procedure will be much easier to understand if we go through the write first.

RESOLVE, SHFILE and SHVAR are all non-commercial ZCPR3 utilities, available on Z-Nodes everywhere.

Ok, That's what our command scripts look like. Now, to the explication.

First I used SHDEFINE to create 12 different *.VAR files, named 0189.VAR, 0289.VAR, 0389.VAR, 0489.VAR, and so on up to 1289.VAR, one to represent each month of 1989, where 0189 = January 1989, 0289 = February 1989, etc. On a floppy that's only 24K and on a hard disk I'm sure you can find 48k to clear out somewhere. I scarcely notice it now. The purpose of these files is to store information relating day of week with any given date of month. I did this by defining in each file string variables named '01,' '02,' '03,' and so on, up to '30' (or '28' or '31', depending on how many days in that month).

Then, with a calendar in front of me, I defined each string variable (again, approximately 30 in each) in each of the 12 VAR files to be the name of its corresponding day of week. For instance, January 1, 1989 is a Monday. I entered "SHDEFINE 0189" on the command line to create 0189.VAR, got into its Edit mode, and defined a variable "01" to be the string of characters "MONDAY". Then the variable string "02" was assigned the definition "TUESDAY", and so on up to "31" which was defined as "SUNDAY", since January 31 is a Sunday.

Now I was almost ready to save 0189.VAR but before I did there was one last definition to give it. We want ultimately to be able to use this VAR file to return to us the **name** of the month. I did this by simply adding one more variable, calling it "MONTH". Still in SHDEFINE I added a string variable "MONTH" and here defined it to be the string "JANUARY". I then saved and exited, thus creating 0189.VAR on disk with my definitions. I then opened up 0289.VAR with the command 'SHDEFINE 0289' and did the same thing for all the days of February. I repeated this process for each of the remaining months of 1989, still with my calendar in front of me to confirm everything, defining 30 or 31 string variables in each file to be their respective days of the week, and one additional string named "MONTH" in each VAR file to its respective name of month.

It took a half-hour or so, but twelve *.VAR files were now ready to serve me powerfully in many different capacities, for many different purposes.

Now I was faced with the rather interesting challenge of how to get ZCPR3 to know which VAR file to access. Let's assume today is March 11, 1989. How can we possibly get the system to define 0389.VAR as the current shell variable file? Interesting problem, eh?

OK, watch this:

```
SHFILE $dm$dy
```

is our line in ALIAS.CMD that does it. Remember SHFILE from last column? Whatever is given as its parameter becomes the current VAR file. Since we are running ZSDOS/DateStamper, ARUNZ's wonderful date parameters will translate the above line into "SHFILE 0389" and 0389.VAR then becomes our current VAR file.

This is our wonderful "number-as-a-string" technique which I will go into greater detail next column and which comes in very handy in all sorts of situations. There is no reason we cannot use digits in a filename ("0389.VAR"), and here we can see it has served us well. When Jay added date parameters to the ARUNZ parameter set, I'll bet he didn't expect they would ever be used as filenames! But this typifies ZCPR3—there is no end to the inventiveness and creativity it inspires and facilitates.

OK, now we have 0389.VAR as our current file. Now it's time to really get our fingernails dirty, and what better tools for that than a dynamic duo of Dreas Nielsen's superb RESOLVE in concert with Rick Conn's classic SHVAR? Our next, perhaps somewhat cryptic-looking, but powerful, line is:

```
resolve zex backup budat$hb %$dd %month $dd, 19$dy $dc:$dn $da
```

How's that for a mouthful? Let's get out our fine-tooth comb (or should we use a microscope?)

First, let's see what our line will look like when ARUNZ expands its parameters—the various $d date symbols and "$hb" for "home directory"—since that's what will happen first, even before RESOLVE gets a shot at it. Still assuming our date of March 11, 1989, and let's say it's now 8:15 p.m., we will then get:

```
resolve zex backup budata15 %11 %month 11, 1989 08:15 pm
```

Remember, that since we are on the A15: directory, ARUNZ will expand the "$hb" parameter to "a15." "Budat$hb" therefore becomes here "budat15."

Now that ARUNZ has done its job let's see what RESOLVE needs to do. This is in many respects the most interesting part of the whole process, and the magic key to coaxing the shell variable subsystem into doing our work for us. It is what provides the correct parameters to the line in the ZEX script that does the actual writing to the VAR file. Remember that one of the things RESOLVE does is scan its command line for strings beginning with "%". When it finds any such strings it will dip into the currently defined VAR file and see if there are any variables contained within by that name. If there are, it will expand, or "resolve" the string to its definition. So let's see what happens here.

RESOLVE sees two elements on its command line, "%11" and "%MONTH," and recognizes them as strings with which it needs to concern itself. Remember that SHFILE has previously defined 0389.VAR as the current VAR file. RESOLVE then opens up 0389.VAR, takes a peek inside and looks to see for something named "11" and something named "MONTH." Well, what do you know—there they are. Remember when we were creating our 12 *.VAR files we had defined the various dates of month to be their respective names of day? Now we see the fruits of our labors. RESOLVE sees that the string variable named "11" has been defined as the string "SATURDAY." OK, good—that's one. Now it looks for a variable named 'MONTH' ... and finds it! Inside 0389.VAR it's (appropriately) defined to be "MARCH." Wonderful!

So let's see what wonders RESOLVE hath wrought. It's now finished its work and produced the following command line, all ready to send over to our ZEX script:

```
ZEX BACKUP BUDAT15 SATURDAY MARCH 11, 1989 08:15 P.M.
```

Beginning to look like something we can use? Good; it should. The rest is easy. This command line is passed to ZEX which loads up BACKUP.ZEX, and we send the script a command line of seven (7!) parameters! The main part of BACKUP.ZEX, the part that does the actual backing-up, will be described in my column next issue. For now we're only considering the first two lines, those that are relevant to and finish off our task here.

Here's the first two lines of BACKUP.ZEX and the crowning glory of our project, the part that writes the date to SH.VAR.

```
    SHFILE SH
    SHVAR $1 $2 $3 $4 $5 $6 $7
```

The first thing we need to do is reset the active VAR file to SH.VAR, and SHFILE does that perfectly. The storing of the date must always be done to this file, our general-purpose and default file, since all the reading is—and must be!—done from it and not any other.

If after reading my last column you have a good gut feeling of how SHVAR works, you may be getting an outline of how we're going to do this. Remember that the purpose of SHVAR is to write (or overwrite) shell variables inside the currently defined VAR file. It takes its command line in two parts. The very first parameter becomes the name of the variable with which we wish to work, while the entire remaining command line becomes that variable's definition. So quite properly "BUDAT15"—"BackUp DATe for the directory A15:" will be the the variable name which inside SH.VAR we would like to define. The particular definition we would here like to give it is the character string consisting of the current date and time.

It works perfectly. Our command line, the one that finally does the actual writing, complete with parameters ultimately sent to us from ARUNZ, is:

```
SHVAR BUDAT15 SATURDAY MARCH 11, 1989 08:15 P.M.
```

SHVAR opens up the currently defined VAR file, which is now SH.VAR and no longer 0389.VAR, and looks to see if there is a variable by the name of "BUDAT15". If there isn't it will create it; if there is it will overwrite it. In either case, we will then have a variable named BUDAT15 with a definition of the following string of characters:

```
SATURDAY MARCH 11, 1989 08:15 P.M.
```

Simple, but amazing.

We've made a long journey from ARUNZ's date parameters to a string variable definition, taking one thing and transforming it into another. I've always thought ZCPR3 was the most "Buddhist" of operating systems. Here we got a chance to see the beauty of ZCPR3 and the power of its "tool concept" in action.

Now that we've **stored** the date let's finish our task and see how we **read** it, which is actually the first part of the procedure. We need a utility that will accept parameters and display something on the screen, preferably in a flashy and attractive manner, such as:

```
 THE LAST TIME DIRECTORY XXX WAS BACKED UP WAS:
 XXXXXDAY, MONTH DD, 19YY HH:MM X.M.
```

If you'll recall, the first line of our BACKUP alias is:

```
BACKUP resolve lastback $hb %budat$hb
```

Terry Hazen's wonderful PRNTXT15.COM, available on all Z-Nodes inside PRNTXT15.LBR, does what we want perfectly. It makes a COM file out of text, and can display its command line parameters. I often use it for status reporting or "message" screens. Its symbol '$1' represents "the first parameter," and its '$-1 symbol, modeled after ARUNZ, expands to display "the entire command line after the first parameter." This is just what we need. We create a message screen as above and call it LASTBACK.COM. We set it up as follows:

```
  THE LAST TIME DIRECTORY $1 WAS BACKED UP WAS:
  $-1
```

and it will do exactly what we want when fed the correct parameters.

If you don't want to be fancy and use PRNTXT15, simply use ECHO and make two aliases out of it (you will definitely overflow the command buffer otherwise):

```
BACKUP    resolve echo last time directory $hb
          was backed up was: %budat$hb;backup2

BACKUP2   shfile $dm$dy
          resolve zex backup budat$hb %$dd %month $dd,
             19$dy $dc:$dn $da

      .
      .
      .
```

After we have created LASTBACK.COM as above if we are using it, let's now execute our alias. ARUNZ first expands its parameters and produces the command line:

```
RESOLVE LASTBACK A15: %BUDATA15
```

We can see what will happen now. Since this is the first part of our procedure and we haven't yet invoked SHFILE to change the default VAR file, SH.VAR is currently active. RESOLVE then looks inside SH.VAR for a string variable named BUDAT15. Lo and behold—there it is, just as we wrote it the last time we ran the backup procedure. It sees the definition, which of course is the character string consisting of the last date and time we ran the backup, as we did above. We then have the command line:

```
LASTBACK A15: SATURDAY MARCH 11, 1989 08:15 P.M.
```

and if you've set it up as I have, LASTBACK.COM displays on the screen, inside a beautiful reverse video box:

```
        LAST TIME THIS DIRECTORY WAS BACKED UP:
         SATURDAY, MARCH 11, 1989  08:15 P.M.
```

Wow!

Each time the backup procedure is run on the A15: directory it will overwrite with the new date and time whatever had been the current definition of BUDAT15. The beauty of this is that we can do the same for all our hard disk directories, with a different string variable (BUDATA0, BUDATB3, BUDATD8, etc.) assigned the job of keeping track of the date of last backup for each directory. Just run the script on each of your directories and it will automatically create, and ultimately maintain, an appropriately named variable inside SH.VAR, with this variable assigned a definition of the character string consisting of the date and time of last backup. All these string variables are kept inside a single SH.VAR file.

After the date displays, then the remaining part of BACKUP.ZEX runs, which I will describe next issue.

When I got this worked out I stood up and cheered for ZCPR3. I'm proud to be using such an elegant operating system. I hope it inspires similar feelings in you.

Now let's quickly see how we can use the same technique to get our computers to display the date and time of last power-on (or cold boot). Here's the relevant segment of my STARTUP alias, which chains to START2.

```
STARTUP resolve laston %laston
        shfile $dm$dy
        resolve STARTUP2 %$dd %month $dd, 19$dy $dc:$dn $da
STARTUP2  shfile sh
          shvar laston $1 $2 $3 $4 $5 $6
```

It's basically the same procedure, though this time using two aliases rather than an alias and a ZEX file. (There's no reason the backup technique above couldn't be done with all aliases if you don't have a ZEX script; simply follow the syntax used here.) LASTON.COM is PRNTXT15.COM again, this time configured as:

```
    LAST COLD BOOT OR POWER-ON:
    $
```

The '$' symbol returns the entire command line.

Again, 'SHFILE dmdy' sets the current VAR file appropriately for the current month. RESOLVE expands the crucial string variables, '%$DD' and '%MONTH,' and feeds the entire "resolved" command line to STARTUP2. This second alias then, as above, resets the VAR file to SH.VAR and finally feeds the correct parameters, being the current date and time, to SHVAR to do the actual write/store.

Note that here we are using two aliases, just as for the backup procedure we used an alias and a ZEX file. Though it doesn't matter whether one uses two aliases or an alias and a ZEX file, it is vital that the process be split somehow into two parts. The reason for this is that when we do the actual write to file with SHVAR we must write to the specific file named SH.VAR. The parameters we feed to it, however, depend on the expansion of string variables inside 0389.VAR!! SHVAR (unlike SHDEFINE, by the way) can only write to the currently defined VAR file. This two-alias, or alias-and-a-ZEX-script solution, in which the second component resets the VAR file to SH, is our perfectly acceptable workaround.

I have had this procedure as a permanent part of my STARTUP alias for months now. It is an absolute joy and delight to see my dinosaur 1984 CP/M Morrow computer do something as sophisticated and elegant as remembering, storing and displaying to me the time and date of last power-on each time I activate that magic red switch. We ZCPR3 users really do have an awfully lot to be grateful for.

And now for the last interesting alias that uses the 12 *.VAR files we've created. Do you want to know on what day your son's birthday, or your anniversary, or Memorial Day, falls in 1989—without consulting a calendar? I call this alias DAYOFWEEK. What? You didn't know an ARUNZ alias name could consist of more than 8 letters? Shame on you. An alias can be up to 12 possible letters. ARUNZ is not called an EXTENDED command processor for nothing. . .

```
DAYOFWEEK if nu $1
          shfile $dm$dy
          resolve echo today is %$dd
          else
          if ex a15:$1$3.var
          shfile $1$3
          resolve echo %month $2, 19$3 is a %$2
          else
          echo that is beyond my ability
          zif
```

Entered with no parameters it will return with today's day of week. With parameters exactly in the format of

```
DAYOFWEEK MM DD YY
```

it will give you the day of week of the specified date, as long as there is an appropriate VAR file. If there is no VAR file, as would happen if you entered:

```
DAYOFWEEK 06 11 90
```

the alias would humbly inform you of its ignorance. If you shorten your command names (I have ELSE permanently renamed as L, ECHO as E, RESOLVE as RS, etc.) you can easily fit in a syntax statement here as well, such as

```
IF EQ $1 //;ECHO SYNTAX = ''$0 MM DD YY''
```

What I hope I have conveyed with this column is not the mechanics of any particular application but rather the general principles of using the ZCPR3 shell variables to store and later retrieve information. Although I think my use of them here in combination with ZSDOS/DateStamper to store date information for later access is especially enjoyable and intriguing, and is ideally suited to the shell variable feature, the same principles are available for working with many different and varied types of information.

Have lots of fun with these techniques. Please feel free to write. Z you next time. . . ■

Resident Programs
TSR's and How They Work
by Dr. Edwin Thall

Dr. Edwin Thall, Professor of Chemistry at The Wayne General and Technical College of The University of Akron, teaches chemistry and computer programming.

A new approach to writing programs was ushered in when IBM and Microsoft introduced the terminate-but-stay-resident (TSR) function call with the DOS 1.1 version. Software developers quickly caught on and now just about every conceivable utility has been made into a resident program. A preassigned "hot" key allows you to pop up such applications as calculators, calendars, address books, and notepads. As the name suggests, resident programs remain in computer memory while other programs execute. But resident programs are also potential disasters since, without warning, they can lose data, scramble the screen, or crash the system.

The TSR was conceived mainly to facilitate device drivers and peripherals. Unfortunately, DOS does not provide the controls needed to maintain peaceful coexistence between two or more resident programs. Users seem to be divided into two camps: those who swear by TSRs and those who swear at TSRs. The intention of this article is not to defend nor attack the widespread use of TSRs, but rather to thoroughly explain how they work. Two examples, one well-behaved and the other, a "bully," will be presented. Removing a resident program without rebooting the system will also be explored.

Resident Versus Transient Memory

COMMAND.COM is divided into three components: the resident, initialization, and transient portions. The resident portion is loaded in lower memory immediately following IBMDOS.COM. Interrupt vectors 21-24H point to routines within the resident portion. This portion also contains the bulk of the error recovery messages. Once loaded, the resident portion remains in memory until the system is shut down.

The initialization portion of COMMAND.COM is loaded immediately above the resident portion when the system is booted. This portion initially takes control and displays the prompts for the date and time. It also possesses the AUTOEXEC batch file, if one is present. The first program from disk overlays this section of memory.

The transient portion is loaded in the high end of COMMAND.COM. and has the capability to overlay one interim program with another. As resident connotes permanency, transient implies temporary. The responsibilities of the transient portion include displaying the DOS prompt and reading, as well as executing, commands. The transient portion also contains the routines to load .COM and .EXE files into the appropriate memory location for execution.

When you request execution of a program, the transient portion constructs a program segment prefix directly above the resident portion of COMMAND.COM. It then loads the executable program immediately following the program segment prefix, sets the exit addresses, and transfers control to your program. When execution is complete, COMMAND.COM regains control and assigns the same memory locations to your next application. However, if you terminate with one of the terminate-but-stay-resident functions, the program becomes an extension of the resident portion of COMMAND.COM. The area of memory occupied by the program is reserved in the same manner as memory is reserved for DOS. Future applications will not overwrite this section. The only way to eliminate such a program, without the benefit of a resident memory manager utility, is to reboot DOS.

Establishing Residency

We are ready to install a simple program directly above the resident portion of COMMAND.COM. After installation, the DEBUG utility will be used to locate and examine the program. Let's begin by calling Interrupt 27H to incorporate the message "STAY RESIDENT" in the resident portion of memory.

Interrupt 27H terminates the currently executing program and reserves part or all of its memory so that it will not be overlaid by the next transient program. The maximum quantity of memory that can be reserved by this interrupt is 64K bytes. From DOS, load DEBUG and enter the following program (omit comments):

```
A>DEBUG
-A100
DS:0100  JMP   0110              ;BY-PASS DATA
DS:0102  DB    'STAY RESIDENT '  ;DATA
DS:0110  MOV   DX,0110           ;PROTECT UP TO HERE
DS:0113  INT   27                ;TSR
DS:0115  <ENTER>
```

The program's code is 21 bytes (offsets 0100-0114H) and can be viewed by typing:

```
-D100,114
DS:0100  EB 0E 53 54 41 59 20 52-45 53 49 44 45 4E 54 20
DS:0110  BA 10 01 CD 27
```

Later, we will search resident memory to locate this code. Preceding the code is the 256-byte program segment prefix (PSP). The PSP occupies offsets 00-FFH and is made resident along with the code. You can display the PSP with:

```
-D0,FF
```

Here's how the program works. The first instruction (JMP 0100) by-passes the data (DB statement). The next instruction (MOV DX,0110) specifies the program be protected up to, but not including, offset 0110H. When the program terminates with the last instruction (INT 27H), the PSP (offsets 0000-00FFH) and code (offsets 0100-010FH) are made an extension of the resident portion of COMMAND.COM. Save the program as RES27H.COM:

```
-NRES27H.COM
-RCX
CX ????
:0015
-RBX
BX ????
:0000
-W
```

Since Interrupt 27H cannot be invoked from DEBUG, return to DOS and execute the program:

```
-Q
A>RES27H
```

Reload DEBUG and use the "S" command to search the first 64K bytes of memory for the program's code. When searching the initial 64K bytes of computer memory, you must set the data segment (DS) to zero.

```
A>DEBUG
-RDS
DS ????
:0000
-S 0 L FFFF EB 0E 53 54 41 59 20 52
0000:61E0
0000:8F6F
```

Ignore the last address retrieved by the search command. If you are using DOS 2.10, the program's code begins at 0:61E0H with the PSP 100H bytes below at 0:60E0H. To display the PSP and code, enter:

```
-D0:60E0,61EF
```

How is DOS able to keep track of resident memory allocations? The paragraph directly below the PSP (0:60D0H) is a memory control block. Paragraphs originate at an offset ending in zero and are 16 bytes in length. Let's explore the first five locations of this block:

```
-D60D0,60D4
0000:60D0    4D 0E 06 11 00
```

The first location of a memory control block, called the identifier byte, contains either 4DH or 5AH. The value 4DH indicates that the memory directly above belongs to a program or DOS, while 5AH verifies that no more memory control blocks succeed this one. The value 5AH also instructs DOS to install the next TSR here. Bytes 2 and 3 of the memory control block hold the PSP segment (060EH) of the program to follow. If the PSP segment number is zero, then the memory defined by the memory control block is free. Bytes 4 and 5 specify the size (in paragraphs) of the pending memory block. Notice the value is 0011H or 17 paragraphs. Memory control blocks chain from one to the next, and DOS is notified that the next memory control block is 17 plus one or 18 paragraphs above. Between these memory control blocks are the PSP (16 paragraphs) and the program's code (1 paragraph). The last 11 bytes in a memory control block are not used by current DOS versions and may contain remnants from other programs.

To predict the location of the next resident program, search above the PSP for the memory control block beginning with 5AH. The first occurrence here is at 0:6220H. Look at the initial five bytes of this paragraph:

```
-D6220,6224
0000:6220    5A 23 06 DD 79
```

The information tells us to expect the PSP of the next resident program to be loaded at segment 0623H (address 0000:6230H).

Let's install the same program a second time in resident memory but this time by invoking the other TSR function. DOS function 31H was introduced with the 2.00 version. This function is preferred by IBM because it passes a return code and allows more than 64K bytes to remain resident. When using DOS function 31H, the number of bytes made resident are declared in the DX register by paragraph. The value stored in the DX register (0011H) specifies that 17 paragraphs are to be protected. Before entering the program with the Debug "A" command, secure a new DS designation by returning to DOS and reloading DEBUG:

```
-Q
A>DEBUG
-A100
DS:0100  JMP  110                    ;BY-PASS DATA
DS:0102  DB   'STAY RESIDENT '       ;DATA
DS:0110  MOV  AH,31                  ;TSR
DS:0112  MOV  AL,01                  ;RETURN CODE
DS:0114  MOV  DX,0011                ;SAVE 17 PARAGRAPHS
DS:0117  INT  21                     ;CALL DOS
DS:0119  <ENTER>
```

When this program terminates, 16 paragraphs of PSP and one paragraph of code will emerge as part of resident memory. Save the program as RES31H.COM:

```
-NRES31H.COM
-RCX
CX ????
:0019
-RBX
BX ????
:0000
-W
```

Return to DOS and execute RES31H.COM:

```
-Q
A>RES31H
```

To determine where the second resident program has been stored, search with the Debug "S" command:

```
A>DEBUG
-RDS
DS ????
:0000
-S 0 L FFFF EB 0E 53 54 41 59 20 52
0000:61E0
0000:6330
0000:90BF
```

As we anticipated, the new resident code begins at 0:6330H with the PSP at 0:6230H. The memory control block is located at the paragraph preceding the PSP (0:6220H). Display the initial five bytes of this block:

```
-D6220,6224
0000:6220    4D 23 06 11 00
```

The identifier byte (4DH) indicates that a PSP follows at segment 0623H. The length of the PSP/code are 0011H or 17 paragraphs. Every time you execute a TSR function, the protected memory is "stacked" directly above the previous resident program. You can continue stacking resident programs until the available RAM is depleted.

By now, you have probably realized how easy it is to install a program or data in the resident portion of the computer's memory. The not-so-easy part involves activating it after it is resident. This takes us to the subject of interrupt-handlers.

Interrupt Handlers

Interrupts notify the computer's central processing unit to suspend what it is doing and transfer to a an interrupt handler program. The handler quickly takes the appropriate action and then returns control to the original program that was suspended. The 8086/8088/80286 family supports up to 256 interrupts, which are classified as internal hardware, external hardware, or software interrupts.

Internal hardware interrupts are generated by certain events encountered during program operation, such as division by zero. External hardware interrupts are triggered by peripherals such as the keyboard. Software interrupts may be requested by any program executing an interrupt type instruction.

The initial 1,024 bytes of computer memory are known as the interrupt vector table. Each table entry consists of four bytes pointing to the address of its handler. The first 128 bytes of this table for the DOS 2.10 version is provided in Figure 1. Each four-byte position in the table corresponds to an interrupt type (00-FFH) and contains the segment and offset of its interrupt handler. To find a particular interrupt in the table, multiply the interrupt type by four. For example, INT 9H is stored in offsets 36-39 (0024-0027H) and points to address F000:E987H in ROM. Our main programming interest in interrupt vectors is not to read them, but to change them so they point to a new interrupt handling routine.

```
-D0:0
0000:0000  72 30 EB 00 47 01 70 00-C3 E2 00 F0 47 01 70 00
0000:0010  47 01 70 00 54 FF 00 F0-23 FF 00 F0 23 FF 00 F0
0000:0020  A5 FE 00 F0 87 E9 00 F0-23 FF 00 F0 23 FF 00 F0
0000:0030  23 FF 00 F0 55 07 00 C8-57 EF 00 F0 47 01 70 00
0000:0040  65 F0 00 F0 4D F8 00 F0-41 F8 00 F0 51 02 00 C8
0000:0050  39 E7 00 F0 59 F8 00 F0-2E E8 00 F0 D2 EF 00 F0
0000:0060  00 00 00 F6 92 01 00 C8-6E FE 00 F0 40 01 70 00
0000:0070  49 FF 00 F0 A4 F0 00 F0-22 05 00 00 00 00 00 00
```

Figure 1. The first 128 bytes of interrupt vector table (DOS 2.10 version).

Writing interrupt handlers has the reputation of being difficult and best left to experts. Actually, the procedure is straightforward. When an interrupt is invoked, the CPU pushes the flag register, the segment register (CS), and the instruction pointer (IP) onto the stack and disables interrupts. It then uses the interrupt number to fetch the address of the handler from the vector table and resumes execution at that address. The handler will enable interrupts, save registers, and then process the interrupt.

You can readily activate a resident program by modifying an interrupt's vector. Two interrupts especially qualified for this task are the keyboard interrupt (INT 9H) and the user timer interrupt (INT 1CH). Before we attempt to take control of these interrupts, there are a few things you should know. The handler can call DOS, and DOS will perform the function, but the handler cannot be reentered from DOS. The last instruction of a handler must be IRET (IP, CS, and flags registers popped). When an interrupt is invoked, the trap flag is suspended and you cannot trace through a handler to locate errors.

Each time a key is pressed or released, the action is reported to the ROM-BIOS by means of INT 9H. The ROM routine reads port 60H to find out which keystroke was selected. The scan code and ASCII value are stored until the next key is pressed. When you take control of INT 9H, every keystroke is routed through the resident program and then passed along to ROM. Special keystrokes or combinations (hot keys) are used to activate the resident program.

Hot keys are assigned to keystroke combinations not commonly accessed. The special control keys are ideally suited for this role because they do not produce characters of their own, but change the codes generated by other keys. The shift status of the last keystroke can be tested by means of the keyboard I/O interrupt (INT 16H, function 02). This function returns the status of the eight keys listed in Figure 2. For example, if both shift keys were pressed, the AL register returns the value 3. When all eight keys are pressed simultaneously, the value 255 is returned. A resident program can be activated by testing for a definite value. Otherwise, keystrokes are treated in the usual manner.

Bit	specifies	value
0	right shift	1
1	left shift	2
2	Ctrl key	4
3	Alt key	8
4	Scroll Lock	16
5	Num Lock	32
6	Caps Lock	64
7	Insert state	128

Figure 2. The eight shift status codes.

The user timer interrupt is taken 18.2 times per second and is invoked by the timer interrupt (INT 8H). The vector for Interrupt 1CH points to address F000:FF49H in ROM. Use the DEBUG "U" command to look at the first instruction of this handler:

```
-UF000:FF49,FF49
F000:FF49  CF    IRET
```

This is a dummy handler which does nothing but execute an interrupt return. However, control of this interrupt allows you to continuously activate a procedure in resident memory.

Terminate-but-stay-resident programs typically include an initialization procedure, a portion to redefine the interrupt vector table, and a routine to remain resident. Normally, a program will want to leave only part of itself resident, discarding the initialization code. Therefore, you should organize a TSR so that the resident portion is placed at the beginning of the program. To demonstrate how resident programs work, POPUP and RENEGADE are introduced. These programs save the current screen in resident memory, but RENEGADE exerts absolute control over the keyboard interrupt. Once the keyboard is in its grasp, RENEGADE does not relinquish control and excludes all other resident programs from using this interrupt.

Introducing POPUP

The assembly language source code for POPUP is listed in Figure 3. After it is installed in resident memory, the program is activated by hitting both shift keys concurrently. The first entry stores the current screen in resident memory, while subsequent activations pop up the screen. If you have the inclination to run this program, you must engage the Macro Assembler to convert POPUP.ASM to POPUP.COM.

Figure 3. The assembly language source code for POPUP.ASM.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;POPUP.ASM                                                     ;
;Resident program to save screen.                              ;
;Press both shift keys to save first screen or to activate.   ;
;Press Alt/Rt. shift keys to save new screen.                  ;
;<ESC> to exit.                                                ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

IVT     SEGMENT AT 0H           ;INTERRUPT TABLE SEGMENT
        ORG 9H*4
KEYBD   DW 2 DUP (?)            ;INT 9H VECTOR
IVT     ENDS
;--------------------------------------------------------------
CODE    SEGMENT PARA PUBLIC 'CODE'  ;CODE SEGMENT
        ASSUME CS:CODE
        ORG 100H
BEGIN:  JMP  INIT               ;GOTO INITIALIZATION ROUTINE
OLDKB   LABEL DWORD             ;INT 9H ADDRESS IN ROM
OLDKEY  DW 2 DUP (?)            ;STORE OLD INT 9H VECTOR
SCREEN  DB  4000 DUP ('S')      ;STORE ORIGINAL SCREEN
SHIFT   DB  0                   ;SAVE SHIFT STATUS CODE
STATUS  DB  0                   ;CHECK NEW SCREEN STATUS
POPUP   DB  4000 DUP ('P')      ;POP-UP SCREEN
```

(Figure 3 continued)

```
;------------------------------------------------------------
;MAIN is made resident and every keystroke routed here.
;------------------------------------------------------------
MAIN    PROC  NEAR              ;NEW INT 9H VECTOR POINTS HERE
        STI                     ;ENABLE INTERRUPTS
        PUSH  AX                ;SAVE REGISTERS
        PUSH  BX
        PUSH  CX
        PUSH  DX
        PUSH  SI
        PUSH  DI
        PUSH  DS
        PUSH  ES
        PUSHF                   ;SIMULATE INTERRUPT RETURN
        CALL  OLDKB             ;OLD KEYBD ROUTINE IN ROM
        MOV   AH,2              ;RETURN KEYBD FLAGS
        INT   16H
        MOV   SHIFT,AL          ;SAVE SHIFT STATUS
        AND   AL,3
        CMP   AL,3              ;BOTH SHIFT KEYS PRESSED?
        JE    SAVE              ;IF YES, THEN SAVE SCREEN
        MOV   AL,SHIFT          ;RESTORE SHIFT STATUS
        AND   AL,9              ;ALT/RT. SHIFT KEYS PRESSED?
        CMP   AL,9
        JNE   EXIT              ;IF NO, THEN EXIT HANDLER
        MOV   STATUS,Ø          ;EXPECT NEW SCREEN
        JMP   SAVE              ;GET NEW SCREEN

EXIT:   POP   ES                ;RESTORE REGISTERS AND EXIT
        POP   DS
        POP   DI
        POP   SI
        POP   DX
        POP   CX
        POP   BX
        POP   AX
        IRET                    ;RETURN TO PARENT PROGRAM

;------------------------------------------------------------
;This routine activated when both shift keys
;or Alt/Rt shift pressed.
;------------------------------------------------------------
;Save the original cursor position
SAVE:   MOV   AH,3              ;READ CURSOR POSITION
        MOV   BH,Ø              ;SELECT PAGE Ø
        INT   1ØH
        PUSH  DX                ;SAVE CURSOR POSITION
        PUSH  CX                ;SAVE CURSOR SIZE

;Save the original screen
        MOV     AX,ØB8ØØH       ;COLOR GRAPHICS MEMORY
        MOV     DS,AX           ;SOURCE SEGMENT
        MOV     SI,Ø            ;SOURCE OFFSET
        PUSH    PS
        POP     ES              ;DEST. SEG
        MOV     DI,OFFSET SCREEN  ;DEST. OFFSET
        CLD                     ;CLEAR DIRECTIONAL FLAG
        MOV     CX,4ØØØ         ;MOVE 4ØØØ BYTES
        REP     MOVSB           ;FROM VIDEO TO MEMORY

;Determine if new screen
        CMP     STATUS,ØFFH     ;CHECK NEW SCREEN STATUS
        JE      READY           ;IF YES, THEN GOTO READY

;Move the first pop-up screen
        PUSH    CS
        POP     DS              ;SOURCE SEG
        PUSH    CS
        POP     ES              ;DEST. SEG
        MOV     SI,OFFSET SCREEN  ;SOURCE OFFSET
        MOV     DI,OFFSET POPUP   ;DEST. OFFSET
        MOV     CX,4ØØØ         ;MOVE 4ØØØ BYTES
        REP     MOVSB           ;FROM SCREEN TO POPUP
        MOV     STATUS,ØFFH     ;CHANGE STATUS AFTER FIRST RUN
        JMP     POS

;Save new pop-up screen
READY:  MOV     DI,Ø            ;DEST. OFFSET
        MOV     SI,OFFSET POPUP   ;SOURCE OFFSET
        PUSH    CS
        POP     DS              ;SOURCE SEG
        MOV     AX,ØB8ØØH
        MOV     ES,AX           ;DEST. SEG
        CLD
        MOV     CX,4ØØØ         ;MOVE 4ØØØ BYTES
        REP     MOVSB           ;FROM MEMORY TO VIDEO

;Turn cursor off
POS:    MOV   AH,2              ;POSITION CURSOR
        MOV   BH,Ø
        MOV   DL,Ø              ;FIRST COLUMN
        MOV   DH,25             ;ROW OFF VIDEO DISPLAY
        INT   1ØH

;Wait for escape keystroke
WAIT:   MOV   AH,Ø              ;WAIT FOR KEYSTROKE
        INT   16H
        CMP   AL,27             ;ESCAPE KEY?
        JNZ   WAIT

;Restore original cursor
        POP     CX              ;ORIGINAL CURSOR SIZE
        POP     DX              ;ORIGINAL CURSOR POSITION
        MOV     BH,Ø
        MOV     AH,2            ;SET CURSOR
        INT     1ØH

;Restore original screen
        MOV     DI,Ø            ;DEST. OFFSET
        MOV     SI,OFFSET SCREEN  ;SOURCE OFFSET
        PUSH    CS
        POP     DS              ;SOURCE SEGMENT
        MOV     AX,ØB8ØØH
        MOV     ES,AX           ;DEST. SEGMENT
        CLD
        MOV     CX,4ØØØ         ;MOVE 4ØØØ BYTES
        REP     MOVSB           ;FROM MEMORY TO VIDEO
        JMP     EXIT            ;HANDLER IS DONE
MAIN    ENDP

;------------------------------------------------------------
;INIT is executed once and is not made resident.
;------------------------------------------------------------
INIT    PROC NEAR
        MOV AX,IVT              ;SET DS TO INTERRUPT VECTOR
TABLE
        MOV DS,AX
        ASSUME  DS:IVT
        MOV AX,KEYBD            ;SAVE OLD INT 9H VECTOR
        MOV OLDKEY,AX
        MOV AX,KEYBD[2]
        MOV OLDKEY[2],AX
        CLI                     ;DISABLE INTERRUPTS
        MOV KEYBD,OFFSET MAIN   ;INSTALL NEW VECTOR
        MOV KEYBD[2],CS
        STI                     ;ENABLE INTERRUPTS
        MOV DX,OFFSET INIT      ;SAVE TO END OF MAIN PROCEDURE
        INT 27H                 ;TSR
INIT    ENDP
;------------------------------------------------------------
CODE    ENDS
        END BEGIN
```

Here's how POPUP works. The first instruction in the code segment (JMP INIT) by-passes the MAIN procedure and skips to the INIT procedure. The INIT procedure saves the old keyboard vector and then chains INT 9H to the MAIN procedure. The last instruction of the INIT procedure (INT 27H) makes the ultimate sacrifice by allowing itself to be erased. It terminates the entire operation but protects code from the start of the PSP to the end of the MAIN procedure. Note how the DX register points to the start of INIT (the offset one byte beyond the code protected).

Once installed, the MAIN procedure becomes our interrupt handler. Every keystroke is intercepted by this handler and redirected to the old INT 9H in ROM for processing. Our handler then calls INT 16H to determine the last keystrokes. If both shift keys had been pressed simultaneously, a branch takes place to a routine that saves the current screen. This screen will now pop up whenever both shift keys are accessed. To save a new screen, the Alt and right shift keys are pressed. The escape key restores the original screen and returns control to the parent program.

POPUP saves two screens: the application screen to be returned to and the ensuing pop-up screen. Storing a complete screen demands a significant block of computer memory. The storing of a color graphic screen consumes 4000 bytes, whereas the entire POPUP program requires about 8200 bytes. As you can see, the two stored screens account for 98% of the memory allocated to POPUP.

POPUP is an example of a well-behaved resident program; it does not attempt to sabotage other programs. But what if a second resident program is installed, and it also seeks control of the keyboard interrupt? The last program loaded will intercept the keystrokes first, and then chain them to the previously loaded TSR. A problem would arise if both programs attempted to access the same hot keys.

Introducing RENEGADE

Not all resident programs are as well-mannered as POPUP. Some have been labeled thugs, bullies, and outlaws. What have these programs done to warrant such a reputation? When some commercial programs replace the table address of INT 9H, they exclude all others and do not allow concurrent use of a resident program like POPUP. Let's take POPUP and turn it into a bully.

The assembly language source code for RENEGADE, a not so well-behaved resident program, is listed in Figure 4. The MAIN procedure of this program, not presented in its entirety in the figure, is identical to POPUP. RENEGADE performs the same operations as POPUP; however, it takes over INT 1CH as well as INT 9H. As mentioned previously, INT 1CH is taken 18.2 times per second and does nothing except return from ROM. We will use this interrupt to continuously monitor the interrupt vector table.

The initialization portion of RENEGADE takes control of INT

Figure 4. The assembly language source code for RENEGADE.ASM.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;RENEGADE.ASM                                              ;
;Same operation as POPUP.ASM but takes over INT 1CH and    ;
;invokes CHECK procedure 18.2 times per second.            ;
;Recaptures INT 9H and excludes other resident programs.   ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

IVT     SEGMENT AT 0H           ;INTERRUPT TABLE SEGMENT
        ORG 9H*4
KEYBD   DW 2 DUP (?)            ;INT 9H VECTOR
        ORG 1CH*4
TIMER   DW  2 DUP (?)           ;INT 1CH VECTOR
IVT     ENDS
;------------------------------------------------------------
CODE    SEGMENT PARA PUBLIC 'CODE'  ;CODE SEGMENT
        ASSUME CS:CODE
        ORG 100H
BEGIN:  JMP  INIT               ;GOTO INITIALIZATION ROUTINE
OLDKB   LABEL DWORD             ;INT 9H ADDRESS IN ROM
        DB   87H,0E9H,00H,0F0H  ;STORE INT 9H VECTOR
NEWKEY  DW 2 DUP (?)            ;CURRENT INT 9H VECTOR
SCREEN  DB  4000 DUP ('S')      ;STORE ORIGINAL SCREEN
SHIFT   DB  0                   ;SAVE SHIFT STATUS CODE
STATUS  DB  0                   ;CHECK NEW SCREEN STATUS
POPUP   DB  4000 DUP ('P')      ;POP-UP SCREEN
;------------------------------------------------------------
;MAIN is made resident and every keystroke routed here.
;------------------------------------------------------------
MAIN    PROC  NEAR            ;NEW INT 9H VECTOR POINTS HERE
;*****                                      *****
;*****   IDENTICAL TO MAIN PROC IN POPUP.ASM   *****
;*****                                      *****
MAIN    ENDP
;------------------------------------------------------------
;Once installed, CHECK is invoked 18.2 times/sec.
;INT 1CH vector point here.
;------------------------------------------------------------
CHECK   PROC    NEAR
        STI                     ;ENABLE INTERRUPTS
        PUSH    AX              ;SAVE REGISTERS
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    SI
        PUSH    DI
        PUSH    DS
        PUSH    ES
        PUSH    CS
        POP     DS                  ;SOURCE SEGMENT
;Determine if INT 9H vector was changed
        MOV     SI,OFFSET NEWKEY ;SOURCE OFFSET
        MOV     AX,0
        MOV     ES,AX               ;DEST. SEGMENT
        MOV     DI,9H*4             ;DEST. OFFSET
        MOV     CX,2                ;COMPARE 2 WORDS
        REPE    CMPSW
        JE      SKIP                ;IF MATCH, EXIT
;Reinstate INT 9H vector
        MOV     AX,IVT
        MOV     DS,AX
        ASSUME  DS:IVT
        CLI                         ;DISABLE INTERRUPTS
        MOV     KEYBD,OFFSET MAIN ;REINSTATE KEYBD VECTOR
        MOV     KEYBD[2],CS
        STI                         ;ENABLE INTERRUPTS
SKIP:   JMP     EXIT
CHECK   ENDP
;------------------------------------------------------------
;INIT is executed once and is not made resident.
;------------------------------------------------------------
INIT    PROC NEAR
        MOV  AX,IVT         ;SET DS TO INTERRUPT VECTOR TABLE
        MOV  DS,AX
        ASSUME  DS:IVT
        CLI                         ;DISABLE INTERRUPTS
        MOV  KEYBD,OFFSET MAIN   ;INSTALL NEW VECTOR
        MOV  KEYBD[2],CS
        MOV  TIMER,OFFSET CHECK  ;INSTALL NEW TIMER VECTOR
        MOV  TIMER[2],CS
        STI                         ;ENABLE INTERRUPTS
        MOV  DX,OFFSET INIT   ;SAVE TO END OF CHECK PROCEDURE
        INT  27H                    ;TSR
INIT    ENDP
;-----------------------------------------------------------
CODE    ENDS
        END  BEGIN
```

1CH and directs it to the CHECK procedure in resident memory. The old INT 1CH vector need not be saved, since it merely executes an interrupt return. The CHECK procedure, invoked 18.2 times a second, determines whether the keyboard interrupt is chaining directly to the MAIN procedure in resident memory. If the INT 9H vector was taken over by another program, the CHECK procedure recaptures it.

RENEGADE is an example of a "selfish" resident program. It does not permit chaining from one resident program to another. The consequence is that other resident programs are excluded from being activated by way of the keyboard. The worth of RENEGADE is diminished by its lack of compatibility. This program works only with computers storing their keyboard interrupt routines at address F000:E987H.

To observe RENEGADE in action, load the program and display the vector for INT 9H. Before executing, reboot the system to purge any previously installed resident programs.

```
A>RENEGADE
A>DEBUG
-D0:24,27
0000:0024  4D 20 0E 06
```

The vector points to the MAIN procedure at 060E:204DH. This is the address if RENEGADE is the first TSR loaded, and you are using the DOS 2.10 version. Now install POPUP and display the INT 9H vector again:

```
-Q
A>POPUP
A>DEBUG
-D0:24,27
0000:0024  4D 20 0E 06
```

Interrupt 9H should still point to 060E:204DH. Once loaded, RENEGADE assures that its handler has first claim to all keystrokes. It not only recaptures the keyboard interrupt vector, but denies POPUP any opportunity to be activated.

What happens if two resident programs attempt similar strategies to take over INT 1CH? The last program loaded is the one whose handler is invoked 18.2 times per second. In this game, the program loaded last has the advantage. Now you can appreciate why memory resident utilities such as SideKick, SuperKey, and Prokey demand that they be loaded last. I tried loading SideKick, an aggressive user of five interrupts, followed by RENEGADE. The result was chaos with the hot keys giving unpredictable and strange screens, even though, interestingly enough, the system did not crash.

Evicting RENEGADE

Software packages are available which allow you to manipulate resident programs. These utilities, often referred to as managers or organizers, have the capability to remove programs from resident memory without rebooting the system. A short program (EVICT.COM) is introduced to demonstrate how to eliminate the grip RENEGADE has over the keyboard interrupt.

Resident programs are distinct from normal ones in two ways: (1) they do not release memory blocks when terminated, and (2) they chain one or more interrupt vectors to themselves. To remove RENEGADE form resident memory, we must restore the interrupt vectors as they existed prior to its installation, as well as release all memory blocks allocated. DOS function 49H will be called to release memory blocks. The segment to be released is specified in the ES register, while the length of the block is already contained in bytes 4 and 5 of the memory control block. Display the following resident memory locations:

```
-D0:60A0,60EF
0000:60A0  4D 0E 06 02 00 05 C8 00-A3 BD 0B A1 02 00 8C 1E
0000:60B0  50 41 54 48 3D 00 43 4F-4D 53 50 45 43 3D 43 3A
0000:60C0  5C 43 4F 4D 4D 41 4E 44-2E 43 4F 4D 00 00 FF FF
0000:60D0  4D 0E 06 14 02 D3 E8 8C-DA 03 C2 A3 1C 0B B8 00
0000:60E0  CD 20 22 08 00 9A F0 FF-0D F0 8C 02 42 05 99 02
```

The memory locations beginning at 0:60D0H represent the memory control block for RENEGADE's PSP/code. Three paragraphs below is another memory control block. It begins at 0:60A0H and also points to RENEGADE's PSP segment at 060EH. This memory control block identifies the environment. The information stored here includes the path and file name used to load RENEGADE. Removing a resident program requires the release of memory blocks belonging to the environment and the PSP/code.

```
-A1ØØ
DS:Ø1ØØ   CLI                       ;DISABLE INTERRUPTS
DS:Ø1Ø1   MOV    AX,ØØØØ
DS:Ø1Ø4   MOV    ES,AX              ;SOURCE SEGMENT
DS:Ø1Ø6   MOV    SI,Ø12E            ;SOURCE OFFSET
DS:Ø1Ø9   MOV    DI,ØØ24            ;DEST. OFFSET (INT 9H)
DS:Ø1ØC   MOV    CX,ØØØ4            ;MOVE 4 BYTES
DS:Ø1ØF   REPZ
DS:Ø11Ø   MOVSB
DS:Ø111   MOV    DI,ØØ7Ø            ;DEST. OFFSET (INT 1CH)
DS:Ø114   MOV    CX,ØØØ4            ;MOVE 4 BYTES
DS:Ø117   REPZ
DS:Ø118   MOVSB
DS:Ø119   STI                       ;ENABLE INTERRUPTS
DS:Ø11A   MOV    AX,Ø6ØE            ;PSP SEGMENT BLOCK
DS:Ø11D   MOV    ES,AX
DS:Ø11F   MOV    AH,49              ;RELEASE PSP BLOCK
DS:Ø121   INT    21                 ;CALL DOS
DS:Ø123   MOV    AX,Ø6ØB            ;ENVIRONMENT SEGMENT BLOCK
DS:Ø126   MOV    ES,AX
DS:Ø128   MOV    AH,49              ;RELEASE ENVIRONMENT BLOCK
DS:Ø12A   INT    21                 ;CALL DOS
DS:Ø12C   INT    2Ø                 ;RETURN
DS:Ø12E   DB     87,E9,ØØ,FØ        ;ORIGINAL INT 9H  VECTOR
DS:Ø132   DB     49,FF,ØØ,FØ        ;ORIGINAL INT 1CH VECTOR
```

Figure 5. Assembler code for EVICT.COM.

We are ready to execute EVICT (Figure 5), the program to purge RENEGADE from resident memory. EVICT is an unsophisticated program and its sole purpose is to demonstrate how to remove a single resident program from DOS 2.10. For EVICT to work properly, RENEGADE must be the only TSR installed. If necessary, reboot the system. Use the DEBUG "A" command to enter EVICT. The program restores the original vectors for Interrupts 9H and 1CH, and then releases the two memory blocks. Save the program, execute from DOS, and return to DEBUG to display the two memory control blocks:

```
-NEVICT.COM
-RCX
CX ????
:0036
-RBX
BX ????
:0000
-W
-Q
A>EVICT
A>DEBUG
-D0:60A0,60EF
0000:60A0  4D 0E 06 02 00 05 C8 00-A3 BD 0B A1 02 00 8C 1E
0000:60B0  50 41 54 48 3D 00 43 4F-4D 53 50 45 43 3D 43 3A
0000:60C0  5C 43 4F 4D 4D 41 4E 44-2E 43 4F 4D 00 00 FF FF
0000:60D0  5A 0E 06 F2 79 D3 E8 8C-DA 03 C2 A3 1C 0B B8 00
0000:60E0  CD 20 22 08 00 9A F0 FF-0D F0 8C 02 42 05 99 02
```

Note the identifier byte at 0:60D0H contains the value 5AH, the designation that the next memory block is free. If you install a new TSR, it will be stored at segment 060EH.

EVICT was a very easy program to create. We knew exactly where RENEGADE was located in resident memory and which interrupts it controlled. For DOS 2.10, EVICT removes the first TSR installed, providing Interrupts 9H and 1CH were the only ones taken over. The difficult assignment of writing a general utility is to keep track of where each TSR is loaded and the interrupt vector table prior to installation.

Consider the scenario of installing RENEGADE followed by POPUP, and then executing EVICT to remove RENEGADE:

```
A>RENEGADE
A>POPUP
A>EVICT
```

The memory control block defining RENEGADE's PSP is modified and can be viewed with DEBUG:

```
A>DEBUG
-D0000:60D0,60D4
0000:60D0   4D 00 00 14 02
```

The identifier byte prevails as 4DH; however, bytes 2 and 3 are filled with zeros. RENEGADE has been purged from resident memory and its memory blocks released. The next TSR is not stored in this vacated area, but immediately following the memory control block containing 5AH. A "hole" develops in resident memory and all newly installed TSRs are placed above POPUP.

Software packages are available that can disable and, later, enable a resident program. These utilities are resident programs that save entire interrupt vector tables prior to the installation of every TSR. To disable a resident program, all interrupt vector tables, except one, are loaded in the same order they were saved. The address of the disabled program will be excluded from the chaining of one handler to the next. Although the code remains in resident memory, the program has no mechanism for activation. The dormant resident program may be enabled by reinstating, in the proper order, the excluded interrupt vector table.

In Closing

As the number of TSRs continue to multiply, a set of guidelines would be helpful. Since IBM and Microsoft are not providing leadership in setting standards, the major software developers should. Some suggestions for guidelines include the assignment of hot keys, designation of interrupts to be taken over, chaining strategies from one handler to another, and procedure for removal of resident programs.

Will it ever work? It seems unlikely. The technology is changing too fast. By the time a set of standards are proposed and accepted, they will probably be outdated. So until all TSRs are able to coexist peacefully, the best hope is for users to stay informed. ■

ZCPR3 Corner

(Continued from page 14)

the flexibility of pointing, for example, to PROGRAM.Z80 and having PROGRAM.COM run. If there is no COM file with a matching name, the error handler will take care of things. You will note the leading colon before the "$n" parameter. It makes sure that the current directory is searched even if it is not on the path. Prompted input is used to allow a command tail to be included.

The Z macro performs a user-specified function on the pointed-to file. Two separate user prompts allow both the command and a command tail to be given. For example, if you wanted to squeeze the file to A0:, you would enter "SQ" in response to the first prompt and "A0:" in response to the second.

The 0 macro illustrates how the response to a prompt can be used as a ZFILER script. This macro takes care of all those functions we forgot to include in ZFILER.CMD. The whole macro is just prompted input, and whatever we answer will be run as a script. I use this function so often that I put it on a number key so that it can be invoked with a single key rather than the usual pair. Also, as you may have noticed, I include in the macro help screen a list of the parameters that can be used.

The only real limitation of this macro-to-write-a-macro approach is that prompted input cannot be included in the response. As I write this, however, it occurs to me that this limitation could be overcome by recursively parsing the prompt parameters until none remain, and only then going on to the subsequent macro expansion steps.

Well, I was going to discuss patching and configuring ZFILER, but this article is already too long, so that will just have to wait for another time. I hope that this article will help you get more out of ZFILER. See you in the next issue! ■

XED4/5/8 Integrated Editor Cross-Assembler

XED4/5/8 is a fast and convenient method to develop and debug small to medium size programs. For use on Z80 machines running Z-system or CP/M. Companion **XDIS4/5/8** disassembler also available.

Targets: 8021, 8022, 8041, 8042, 8044, 8048, 8051, 8052. 8080, Z80, HD64180, and NS455 TMP.

Documentation: 100 page manual.

Features include:

★ Memory resident text (to about 40 KB) for very fast execution. Recognises Z-system's DIR: DU:. Program re-entry with text intact after exit.

★ Built in mnemonic symbols for all 8044,51,52 SFR and bit registers, NS455 TMP video registers and HD64180 I/O ports.

★ Output to disk in straight binary format. Provision to convert into Intel Hex file. Listing to video or printer. A sorted symbol table with value, location, all references to each symbol.

★ Supports most algebraic, logic, unary, and relational operators. Eight levels of conditional assembly. Labels to 31 significant characters.

★ A versatile built in line editor makes editing of individual lines, inserting, deleting text a breeze. Fast search for labels or strings. 20 function keys are user configurable.

★ Text files are loaded, appended, or written to disk in whole or part, any time, any file name. Switchable format to suit most other editors.

★ The assembler may be invoked during editing. Error correction on the fly during assembly, with detailed error and warning messages displayed.

For further information, contact:

PALMTECH (a division of Palm Mechanical)
**cnr. Moonah & Wills Sts.
BOULIA, QLD. 4829
AUSTRALIA**

Phone: 6177 463-109 Fax: 6177 463-198

Advanced CP/M

by Bridger Mitchell

ZSDOS News

The brand-new CP/M disk operating system—ZSDOS—that I announced in this column last fall is meeting an enthusiastic reception. At that time I wrote that the quality of the design and testing that have gone into this project means we are unlikely to see a long series of revision numbers to fix bugs. But I didn't mean to imply that a major upgrade, ZSDOS 2.0, might never appear! Indeed, I should have gone on to say that we may expect further contributions from the design team.

In the first of a multi-part article in this issue of TCJ, two of the ZSDOS authors—Hal Bower and Cameron Cotrill—take you behind the scenes of the many innovations in the new DOS. And several of you have asked about porting ZSDOS to banked-memory (HD64180, Z80) systems. Well, early discussions are afoot to set specifications for a banked-memory version with well-defined memory-management services.

Meanwhile Carson Wilson, the third member of the team, has been avidly turning out system utilities with nifty new features and is at work on a Z-System version of a popular public-domain memory-based editor.

If you haven't already ordered it, ZSDOS is available from Plu*Perfect Systems and Sage Microsystems East.

Feedback Loop

With the demise of yet another magazine (Profiles) that provided some coverage of CP/M topics, TCJ takes on greater prominence as a continuing source of high-quality CP/M information. Art Carlson and we regular columnists need your feedback and suggestions to keep expanding and broadening TCJ's material.

I've appreciated the cards and BBS messages several of you have sent. They indicate that you find these columns worthwhile, although not always fully digestible in one sitting! I will continue to focus on more advanced technical topics relating to the CP/M operating system, aiming to get the core concepts and details into print. I fully expect readers to extend, expand, revise and critique these pieces—that's how our hobby progresses!

I'd especially like to receive suggestions for topics for future columns. One early candidate is methods to make the Z-System external environment address available to applications that have not been coded as a Z-System tool, including compiler-generated COM files.

I'd also welcome information about adding 3.5" and high-density 1.2MB 5.25" drives to CP/M systems. I've recently customized DosDisk for an OEM to handle the AT-style 1.2MB floppy disk format. The DosDisk software could be similarly extended to handle the 3.5" MS-DOS format, but in order for this to be usable the BIOS must talk to the drive. Some of you have doubtless done this, or at least thought it through. It would make a nice TCJ article!

Unit-Record Input/Output

A processor is isolated and quite useless until it can talk to the "outside world". Input and output are essential—they supply the processor with data and enable it to report results. Memory chips provide the fastest I/O. After that come hard disks, floppy disks, magnetic tape, and serial channels at decreasing data rates.

Bridger Mitchell is a co-founder of Plu*Perfect Systems. He's the author of the widely used DateStamper (an automatic, portable file time stamping system for CP/M 2.2); Backgrounder (for Kaypros); BackGrounder ii, a windowing task-switching system for Z80 CP/M 2.2 systems; JetFind, a high-speed string-search utility; DosDisk, an MS-DOS disk emulator that lets CP/M systems use pc disks without file copying; and most recently Z3PLUS, the ZCPR version 3.4 system for CP/M Plus computers.

Bridger can be reached at Plu*Perfect Systems, 410 23rd St., Santa Monica CA 90402, or at (213)-393-6105 (evenings).

In TCJ #35 we covered file systems. Input and output to files is done in blocks (physical sectors) of many bytes, and file storage devices (floppy disks, hard disks, tape drives, ram disks) are sometimes called block-devices.

Our interest in this column is input/output to/from *character* devices—devices that normally supply or accept one byte at a time, such as a terminal, printer or modem. Single-byte devices are sometimes called unit-record devices; they can be thought of as special block devices with a record length of one.

For many purposes it's useful to think of the computer's software environment as a series of rings. At the outer ring are the application programs. Just inside are the high-level languages, and inside that is the operating system. Its outermost layer is the BDOS, providing a standardized, hardware-independent set of high-level services for access to the file structure and bundled input/output services to various devices.

The next ring is the BIOS, providing all of the primitive input/output services needed by the BDOS. It is the border land between a standard system and the specific computer. At the BIOS jump table the interface is completely standardized, but within the BIOS the system designer must program down to bare metal, coding routines that know the precise conditions of the hardware—disk drive, video display, printer handshaking conditions.

In general, applications will be more portable and easier to write if they confine their operating system access to the outer ring—BDOS calls. Yet there are good reasons for using BIOS calls in some applications, those that require highest performance or services unavailable from the BDOS. And in a few cases, an application must forego portability and itself directly access the hardware, because no BIOS service is available; for example, to read a video terminal's screen or use a modem port.

The CP/M 2.2 BIOS provides character-device services for the basic device needed to command the system (the console), an auxiliary device, and a printer. These services are:

CONSTAT	Console Input Status
CONIN	Console Input
CONOUT	Console Output
READER	Auxiliary Input
PUNCH	Auxiliary Output
LISTSTAT	Printer Output Status
LIST	Printer Output

Each input or output service returns or sends a single byte. For input, the routine waits until a byte is ready before it returns; for output, it waits until the device can accept the byte.

Strangely, only one BIOS input device and one output device has a status call function available to an application. The BDOS or an application can determine, by calling CONSTAT, whether a character is waiting in the input (a key has been pressed). Similarly, it can call LISTSTAT to see whether the printer is idle and can accept a character.

But there is no portable way, in CP/M 2.2, to determine whether the console device is ready to *accept* a character. All you can do is call CONOUT to send the character and wait, hoping that the device will eventually be ready. This might seem all right (how would you run a CP/M system if you couldn't see its console output?). But, consider an application that wants to keep the processor running at full efficiency (perhaps a video game, or just a smart display utility). It would like to send a character to the console only when it knows that it will be processed immediately.

Internally, however, the BIOS must have a routine to determine the input and output status of every device. In order to obtain a valid byte of input it must not access the physical device (a parallel port, an asynchronous receiver chip) until the device signals, by some type of status report, that a byte is ready. And similarly, the BIOS must not output a byte to a physical device (video ram, serial transmitter chip, parallel port) until the device signals that its buffer is empty and ready to receive a byte.

Cooked Input

The BDOS provides standardized services to applications, hiding some of the tedious details of communicating with the input and output devices. For the console device the BDOS provides *cooked* (processed) input and output services, sparing the programmer the overhead of including this code in almost every application. For applications needing raw console input and output, the BDOS also provides a raw (uncooked) function #6.

In CP/M 2.2, console single-character input function (#1) provides:

- echo to console output
- flow control
- abort control
- tab expansion

In addition, the console line-input function (#10) provides limited line-editing and printer controls:

- delete last-character (backspace)
- cancel line (^X or ^U)
- retype line (^R)
- list device output control

(Other function #10 editing controls—delete and echo, and end physical line—existed to serve paper-output teletypes. BDOS patches and replacements such as ZSDOS have eliminated them.)

Both the console single-character output function (#2) and the string output function (#9) provide:

- flow control

Flow Control and Lookahead

Flow control is the process of starting and stopping the flow of bytes over an input/output channel. Our concern here is the control of bytes to the console device.

The BDOS is designed so that the user can "freeze" a screen of messages by typing a Control-S—the standard XOFF character. Output will resume by typing Control-Q—the standard XON character. Actually, output resumes when any other character (except Control-C—the abort character) is typed, but it's a good habit to use Control-Q to keep your fingers conditioned for systems, such as UNIX, that use the standard control characters.

In order for flow control to work, the application must print its messages using BDOS functions #2 and #9.

Flow control requires something that many—including compiler authors and BDOS hackers—have found astonishing: the BDOS console-output functions must call the BIOS console *input* functions in order to perform a *lookahead* function. After all, how else could the BDOS know that the user had typed a Control-S to suspend output?

It works like this. Before the BDOS sends a character to the console, it checks the console input status. If no key has been pressed, the character is sent.

But suppose a key has been typed. In this case the BDOS calls the BIOS console input function to get the character. From this moment on, the character is no longer in the BIOS. The BDOS then tests whether the character is a Control-S. If it is, the BDOS waits for the *next* keypress and only then sends the output character. If it is *not* Control-S (or Control-C, discussed below) the BDOS saves the character (say 'A') in a one-character buffer and sends the output character.

If the next operation is to print another character on the console, the BDOS test for flow control will become ineffective. The BDOS has only the one-character buffer, which is now full (it's holding the 'A'), so it cannot check the next keypress for Control-S; if it did, and the character were anything else, it would have to throw away one of the input characters.

The lookahead function might perhaps have been better implemented by providing a "peek" subfunction to the BIOS CONIN—return but retain the pending next character.

The key result is that the next console input character is moved from the BIOS into the BDOS one-character buffer as a result of any BDOS function #2 or #9 output. As a consequence, any application that uses the *BIOS* to obtain console input will sometimes "lose" a typed character, only to have it emerge when the BDOS is next used for input (function #1 or #10)!

Coping With Missing Characters

The simplest rule I can give you for coping with missing characters is to keep all console input/output at *one* level of the operating system—all BDOS or all BIOS—within a single application. For example, don't mix BDOS line input (function #10) and BIOS CONIN.

It's fairly common for applications to use the BIOS functions for console I/O, in order to speed up output and to get every possible keyboard character. The Z3LIB and VLIB routines used in many Z-System utilities do so. At the start of such an application you may need to check the BDOS, using function #11, to see if a fast keypress has already been sucked into the BDOS one-character buffer. If it returns non-zero, you can get the character with function #1 (but that will echo). In order to use function #6 successfully to get the character without echo, you need to have installed the Plu*Perfect Systems patch (described later).

Jay Sage has used the following method of obtaining input (a named-directory password) from BDOS function #10 with echoing shut off. First, save the first byte of the BIOS CONOUT jump vector (it should be the JP opcode) and replace it with a RET opcode. Next, call BDOS function #10. When the BDOS calls the BIOS CONOUT to echo the character, the BIOS will return at once. Then, immediately following the BDOS call, restore the first byte of the BIOS CONOUT jump. (Note that he follows the sound principle of saving and restoring the environment, by saving and restoring the byte in the BIOS "jump vec-

Plu*Perfect Systems == World-Class Software

BackGrounder ii $75

Task-switching ZCPR34. Run 2 programs, cut/paste screen data. Use calculator, notepad, screendump, directory in background. CP/M 2.2 only. Upgrade licensed version for $20.

Z-System $69.95

Auto-install Z-System (ZCPR v 3.4). Dynamically change memory use. Order **Z3PLUS** for CP/M Plus, or **NZ-COM** for CP/M 2.2.

JetLDR $20

Z-System segment loader for ZRL and absolute files. (included with Z3PLUS and NZ-COM)

ZSDOS ....$75, for ZRDOS users just $60

Built-in file DateStamping. Fast hard-disk warmboots. Menu-guided installation. Enhanced time and date utilities. CP/M 2.2 only.

DosDisk .... $30 - $45

Use MS-DOS disks without copying files. Subdirectories too. Kaypro w/TurboRom, Kaypro w/KayPLUS, MD3, MD11, Xerox 820-I w/Plus 2, ON!, C128 w/1571 -- $30. SB180 w/XBIOS -- $35. Kit -- $45. Kit requires assembly language expertise and BIOS source code.

MULTICPY $45

Fast format and copy 90+ 5.25" disk formats. Use disks in foreign formats. Includes DosDisk. Requires Kaypro w/TurboRom.

JetFind $50

Fastest possible text search, even in LBR, squeezed, crunched files. Also output to file or printer. Regular expressions.

To order: Specify product, operating system, computer, 5 1/4" disk format. Enclose **check**, adding $3 shipping ($5 foreign) + 6.5% tax in CA. Enclose invoice if upgrading BGii or ZRDOS.

Plu*Perfect Systems
410 23rd St.
Santa Monica, CA 90402

```
    Figure 1.  Corrected CP/M 2.2 BDOS Function #6 Routine

Authors: Derek McKay, Bridger Mitchell (Plu*Perfect Systems)

 ØØØØ  bdos       equ  ØØØØh          ; base of CP/M 2.2 BDOS
 ØØB7    abort    equ  bdos+ØØB7h     ; ''jp ØØØØ''
 ØØFB    getchar  equ  bdos+ØØFBh     ; get next console input char.
 Ø3Ø1    setretval equ bdos+Ø3Ø1h     ; set return value in A
 Ø3ØA    charbuf  equ  bdos+Ø3ØAh     ; 1-character input buffer
 ØD91    exit     equ  bdos+ØD91h     ; BDOS exit routine

 ØEØØ  bios       equ  bdos+ØeØØh     ; base of BIOS
 ØEØ6    constat  equ  bios+6         ; console status
 ØEØ9    conin    equ  bios+9         ; console input
 ØEØC    conout   equ  bios+ØCh       ; console output

; -- original (8Ø8Ø) Direct Console I/O Routine --
;    malfuncting code marked with ''***''

 Ø2D4            org     bdos  + 2D4h

 Ø2D4  79        fn6:    ld      a,c
 Ø2D5  3C                inc     a
 Ø2D6  CA Ø2EØ           jp      z,fn6in
 Ø2D9  3C                inc     a
 Ø2DA  CA ØEØ6  fn6s:    jp      z,constat      ; ***
 Ø2DD  C3 ØEØC           jp      conout
 Ø2EØ  CD ØEØ6  fn6in:   call    constat        ; ***
 Ø2E3  B7                or      a,a
 Ø2E4  CA ØD91           jp      z,exit
 Ø2E7  CD ØEØ9           call    conin          ; ***
 Ø2EA  C3 Ø3Ø1           jp      setretval

; -- corrected (z8Ø) routine --

 Ø2D4            org     bdos  + 2D4h
                 ;
 Ø2D4  79        fn6:    ld      a,c            ; if c == FF
 Ø2D5  3C                inc     a
 Ø2D6  28 Ø8             jr      z,fn6in        ; ..get character
 Ø2D8  3C                inc     a              ; if c == FE
 Ø2D9  CA Ø137           jp      z,ckstat       ; ..get input status
                                                ;   of buffer & bios
 Ø2DC  C3 ØEØC           jp      conout         ; ..else output char
                fn6in:   call    ckstat         ; check both buffer
                                                ;   and bios
 Ø2DF  CA ØD91           jp      z,exit         ; ..no char waiting,
                                                ;   return Ø status
 Ø2E2  CD ØØFB           call    getchar        ; get char from buffer
                                                ;   or bios
 Ø2E5  18 1A             jr      setretval      ; and return it
                 ;
```

tor''. he doesn't simply assume it is a JP. It's possible that other code—perhaps in an RSX—has already patched this location.)

This trick is handy, but should be used only where no other solution is available. In Jay's case, there was insufficient room in the Z34 command processor to collect a password with function #6. The difficulty with this approach is that during the time that the BIOS CONOUT is patched out it is possible that other processes would be generating console output. What other processes could there be in CP/M? If BackGrounder ii is loaded, a press of the <SUSPEND> key would temporarily suspend the current task and prompt for user input, but the prompt would be invisible! Or an interrupt-driven task could generate a screen message that would be lost.

Abort Control

A Control-C will cause the BDOS to abort the current application, jumping directly to 0000, when it is:

- the *first* character typed after a Control-S has halted output from function #2 or #9.
- the *first* character typed to line-input (function #10)

The abort control feature of the CP/M 2.2 BDOS is a mixed blessing at best. It gives the user a handy way to kill a job that is scrolling unwanted output to the screen. But it limits the use of edited BDOS line input to applications that can tolerate abrupt termination if the user happens to hit Control-C. As a result, most well-written applications must incorporate their own line editor in order to retain control to avoid being canceled with unclosed files, open modem connections, or whatever.

Cooked Output

In addition to flow control, which is a feature of cooked console input that controls the flow of output, the BDOS alters the raw output to the console by special processing of tabs and by creating a parallel stream of output for the printer.

Tab Expansion

The BDOS expands the horizontal tab character (09h) to the number of spaces required to reach the next logical tab stop (every eight characters). To do this it keeps a current-column count for all output to functions #2 and #10, resetting it to 0 on each carriage return.

This is a handy cooked-output service. But to work successfully, all output on the line must go through these BDOS functions. Avoid mixing BDOS and BIOS console output on the same line.

Echoing to the Printer

The BDOS maintains a flag that, when set, causes function #2 and #9 output to be echoed to the BIOS list device as well as the console output. The flag is toggled when a Control-P is typed to function #10—the line-input function.

Control-P is very handy for getting a quick, selective printed record of some console output. It can also mysteriously freeze your system when the printer is not ready. When your computer locks up, make a habit of checking the attached external devices (printer, modem) before you resign yourself to pressing the reset button!

The Case of the Missing Character

If you've used a number of CP/M systems, you've probably had the puzzling and quite annoying experience of occasionally ''losing'' one character you have typed when running a program, only to have it pop up unexpectedly much later, perhaps at the next command prompt. It's a difficult bug to reproduce, and occurs only on some systems. This spooky gremlin is so perplexing that a user can begin to believe his computer is truly haunted!

Has CP/M been visited by the supernatural? Probably not. We've already seen how mixing BDOS and BIOS console functions can cause an input character to become stuck in the BDOS one-character buffer when subsequent input is obtained by BIOS calls.

Several years ago, Derek McKay, my partner at Plu*Perfect Systems, spotted another cause of missing characters—a bug in Digital Research's original design of the CP/M 2.2 BDOS that used faulty logic in the handling of "raw" console input with BDOS function #6. Moreover, Derek developed a Z80 patch that corrects the problem and fits in the original BDOS space. This is an important improvement, because without it there is no totally reliable way to mix cooked BDOS console I/O with any type of raw I/O, either BIOS or BDOS.

We included the patch in the CP/M Enhancements that Plu*Perfect originally published for Kaypro systems. More recently, the authors of ZSDOS have incorporated the same logic into their excellent new DOS. So, on these systems, the missing character doesn't manifest itself.

Raw Console Input

To understand how a character can disappear, and then reappear, we first need to examine the BDOS's raw console input function.

BDOS function #6 was intended to provide absolutely raw console input and output functions accessible by a BDOS call, with no input flow control and no output processing. An application would use this function, for example, when it wanted to get a character without necessarily echoing it to the terminal.

DRI attempted to squeeze input, input status, and output into a single BDOS function (probably to save 8080 code space) and in doing so somewhat limited the usefulness of this service. To use function #6, set C = 6 and

Figure 2. Console Look-Ahead Routines

```
; -- original (8080) console input look-ahead routine --

 Ø123          org     bdos + Ø123h

 Ø123  3A Ø30A lkahead:ld      a,(charbuf)
 Ø126  B7              or      a,a
 Ø127  C2 Ø145         jp      nz,return1
 Ø12A  CD ØEØ6         call    constat
 Ø12D  E6 Ø1           and     1b
 Ø12F  C8              ret     z
 Ø13Ø  CD ØEØ9         call    conin
 Ø133  FE 13           cp      'S'-'@'
 Ø135  C2 Ø142         jp      nz,savechar
 Ø138  CD ØEØ9         call    conin
 Ø13B  FE Ø3           cp      'C'-'@'
 Ø13D  CA ØØØØ         jp      z,ØØØØ
 Ø14Ø  AF              xor     a,a
 Ø141  C9              ret

 Ø142  32 Ø3ØA savechar:ld     (charbuf),a      ; save input char
                                                ; in buffer
 Ø145  3E Ø1   return1:ld      a,1              ; return a non-zero
 Ø147  C9              ret                      ; character

; -- shorter replacement (z8Ø) routine --

 Ø123          org     bdos + Ø123h

 Ø123  CD Ø137 lkahead:call    ckstat           ; if no char waiting
 Ø126  C8              ret     z                ; ..return
 Ø127  DC ØEØ9         call    c,conin          ; if no char in buffer,
                                                ;   call bios
 Ø12A  FE 13           cp      'S'-'@'          ; if not ^S
 Ø12C  20 14           jr      nz,savechar      ; ..return the char
 Ø12E  CD ØEØ9         call    conin            ; ^S, so get next char
 Ø131  FE Ø3           cp      'C'-'@'          ; if ^C
 Ø133  28 82           jr      z,abort          ; ..abort
 Ø135  AF              xor     a,a              ; else return false
 Ø136  C9              ret                      ;    status

; new check-console-status (z8Ø) routine

 Ø137  3A Ø3ØA ckstat: ld      a,(charbuf)      ; if buffered char waiting
 Ø13A  B7              or      a,a              ; ..clear CY and return NZ
 Ø13B  CØ              ret     nz
 Ø13C  CD ØEØ6         call    constat          ; else check bios for a char
 Ø13F  B7              or      a,a              ; if char waiting there
 Ø14Ø  ØF              rrca                     ; ..set CY and set NZ
 Ø141  C9              ret

; resume original (8Ø8Ø) code at Ø142h
```

```
E= OFFh        to get a character, if ready
E= OFEh        to get console input status
(E=OFDh        to wait for a character)
E= 0...OFCh    to output the value in E to the console
```

When used for input, function #6 returns a 1-byte value in A. If A is 0, no character is waiting; a non-zero value is the input character. Thus it is impossible to enter a NUL character (Control-@ on most keyboards) when function #6 is used. (This defect is significant for editors, which must therefore use BIOS functions for console I/O.)

When used for output, function #6 is limited to values 0h to 0FCh. Usually ok, this restriction makes some 8-bit coded graphics characters unprintable on a few terminals. (The subfunction code 0FDh is used by CP/M Plus and ZSDOS to wait for the next character and return it.)

But the real bug in function #6 is its internal check for input status. The original BDOS code (figure 1) calls the BIOS CONSTAT routine to determine if a character is waiting. This is fine, except that another BDOS function may have called the lookahead routine to test for flow control and left the tested character in the lookahead buffer. When that situation exists, function #6 will return A = 0 (no character waiting) until a key is typed, and then return the next typed character, not the one last typed and still in the buffer!

Meanwhile, the tested character continues to sit in the buffer. Eventually, someone—either the application program or the command processor—will call a BDOS function that does check the lookahead buffer before returning a character. It will find the character still there, and return the missing character!

Code

Figure 2 contains the replacement routine. It calls a new "ckstat" routine to determine input status. The new routine just fits into the space made available by rewriting the "lkahead"

(Continued on page 38)

Real Computing
The National Semiconductor NS32032
by Richard Rodman

First, let me apologize to everyone who tried to call my BBS and found it disconnected. I've moved to Manassas (of battlefield and shopping mall fame). The new number is listed below.

Floating point

When it comes to floating point, there are three kinds of people. First, there are the reasonably balanced people, who don't know much about it but use it from time to time, say, to balance their checkbook. Second, there are the hard-core scientists and statisticians who write Fortran programs with COMPLEX*16 variables and judge computers only by their array-processing MFLOPS. Lastly, there are the hands-dirty assembler and real-time programmers who will go to any fixed point length to avoid floating point, and who view floating point as the ultimate expression of sloth and wastefulness.

I have to admit that I fit into the last category. I take some pride in the fact that, in the last five years or so, I haven't written a single program that used floating point. However, I will concede that this is an irrational point of view, and that there are many legitimate uses for floating point. Space flight, for example: Neil Koozer has written, in NS32 assembler, a highly detailed and accurate moon flight simulation. It makes use of the NS32 floating point unit to provide fast and highly accurate floating point math.

The 32081 Floating Point Unit (FPU)

The 32081 FPU uses the 16-bit slave processor protocol, so it can be used with the 32016, 32032, 32008 or 32332 CPUs. It has also been used with 68000 and Z8000 family chips. When used with an NS32 CPU, however, it becomes integrated right into the machine's instruction set. Thus, programmers don't need to worry about its address or any other implementation considerations.

The 32081 has 8 general-purpose floating point registers called F0 to F7, each of which is 32 bits long, and a floating point status register (FSR). Registers can be combined in pairs, called L0, L2, L4, and L6, for 64-bit operations. The floating point storage follows the IEEE standard.

Floating point operands are used in regular instructions just like any other ones. Here are some examples:

```
MOVF    FUEL,F1         Load FUEL into F1
MOVF    FLOW(R0),F1     Load value indexed by R0 into F1
ADDF    F1,F2           Add F1 to F2
SUBF    F1,BALANCE      Subtract F1 from BALANCE
SUBF    CHECK,BALANCE   Again, no need to go through
                        registers. Operands can be used
                        directly to and from memory.
```

The FPU implements add, subtract, multiply, divide, negate, absolute value and compare instructions. It doesn't do the functions such as sine, tangent and hyperbolic sine.

Why not? Well, here arises the philosophical difference between the National approach and the Intel/Motorola approach. The National FPU implements primitives only, so that the implementor can optimize the processing for his specific application as well as detecting problems in intermediate values during the calculation of an advanced function. It is easy to code, for example, a sine given these primitives, and having the CPU involved allows each intermediate value to be tested for convergence according to the programmer's criteria. Further, the CPU is not left inactive for long periods of time, only to get a result that has unknown precision because the intermediate values could not be checked.

The 32381 Floating Point Unit (FPU)

The 32381 floating point unit can use either the 16-bit or the 32-bit slave processor protocols, and thus can be used with any NS32 CPU. It is fully software-compatible with the 32081, but faster (especially when the 32-bit protocol is used) with a 32332 or 32532. A few new instructions have been added, which allow easy testing and manipulation of the floating point exponent and mantissa. These instructions are intended to provide authors of floating point software the tools to more quickly detect underflows, overflows or other conditions involving potential loss of precision.

The 32580 Floating Point Unit (FPU)

The 32580 is National's highest performance floating point unit, which embodies a new approach. Rather than perform the operations itself, it manipulates the operands and uses a high-speed Weitek floating point chip set to do the actual math. This device only uses the 32-bit slave processor protocol and can only be used with the 32332 or 32532. It is intended as a high-end product for use in workstations and other systems where cost is not as much a factor as floating point speed.

Neil has pointed out that in 32016 and 32032 systems, the primary time-consumer is not the math, but the loading and storing of floating point operands in main memory. The 32081 is thus "fast enough" for these CPUs, and the 32381 is "fast enough" for the 32332. With the 32532, however, this was no longer the case, and the new approach of the 32580 became feasible.

The 32832 Memory Management Unit (MMU)

Last time, I discussed the 32082 MMU at length and included a sample program. I neglected to discuss the new 32-bit MMU, the 32382. This MMU supports the full 32-bit address range of the 32332. Since the 32532 has an MMU built-in, the 32382 is usable only with the 32332. However, I believe the 32532's built-in MMU to be substantially the same as the 32382.

The 32-bit addressing range gives an addressing range of 4 gigabytes. Because of this extremely large addressing space, the page size of the MMU has been increased to 4096 bytes. The level-1 and level-2 structure is analogous to that discussed for the 32082, if not a little more symmetrical. Each level-1 page entry (a "superpage") controls 1024 level-2 page entries, or 4 megabytes.

Now, if the operating system uses a common level-2 page table with a separate level-1 page table for each task, you might say, "Gee, I'll have to have 8MB of RAM for a single task!"

However, remember that these are **virtual** addresses, and the mapping of virtual to physical addresses is controlled by the MMU. Any page of physical memory can be assigned to any page, even multiple pages, of virtual memory. Naturally, there are certain pages that will have to be "locked" in place (such as the MMU support code), so there will be a minimum amount of RAM you will have to have in the system—more than just 4K.

At any rate, the operating system software will have to be easily configurable to either page size. Even further, the swapping page size should be configurable independently from the MMU page size.

New coprocessor boards

There are some new coprocessor boards available for those who want to run Unix on their PC-type system. All of the 32332-and-up boards require an AT-clone with its 16-bit bus for I/O.

The Zaiaz coprocessor boards are available from Amazz Computers. There are both 32032 and 32332 boards available. They come with Unix system V, you just plug them in and go.

Opus Systems has coprocessors for the 32032 and the 32332, and just recently announced a 32532 coprocessor board. This board sells for $7000. With that and a nice graphics board, you could just plug together a workstation with more power than a Sun 4. Yes, you could "extinguish the SPARC."

There is a designer's kit for the 32532, but it is a little out of the hobbyist range, costing about $1000. It comes with a 32532, a printed circuit board, monitor PROMs, PALs and documentation.

The free operating system

Those who wish to participate in the free OS project may obtain the pre-release version of Metal, version 0.2, which appears to be basically functional. It is single-tasking and very crude. Send me a diskette and return postage in a reusable mailer. Or, you may obtain it from the BBS. I'll try to put together some documentation, too.

Next time

Art Carlson has thrown out the question of what the personal computers of the 90s will be like. Last time, I discussed how these computers would be set apart from what went before, by virtual memory. Next time, I'll go into more particulars about connectivity, compatibility, and computational power of these new systems (as I envision them). These machines are being designed today, but don't look for them on the drawing boards of Apple or IBM. The designs could be as close as your own desk!

Where to write

John Dodd, Amazz Computers
506E Clinton Avenue
Huntsville AL 35801
(205) 534-6823

Opus Systems Inc.
20863 Stevens Creek Boulevard
Bulding 400
Cupertino CA 95014

Richard Rodman
8329 Ivy Glen Court
Manassas VA 22110
BBS: 703-330-9049

■

Advanced CP/M

(Continued from page 36)

routine just above it in z80 code.

Note the exact logic of the ckstat routine. By clever coding it returns two flag values—nonzero and carry not set when the next character should be obtained from the BDOS and nonzero and carry set when the next character should be obtained from the BIOS.

With this new routine, the lkahead routine can determine from calling ckstat whether to call the BIOS CONIN.

Patching Your BDOS

If you are running the original CP/M 2.2 BDOS you can upgrade it with the function #6 patch. Using a debugger, first check that the original 8080 code is exactly as shown in the figures. Then assemble just the patch code with the BDOS equate set to the base value for your system, and output a hex file.

The hardest part is getting the patch installed in your system. You can load it with a debugger, and then check memory to see that it is installed. But if your system reloads the BDOS on a warm boot, the patch will be gone when the next program runs. If that's the case, you will need get the patch into the SYSGEN.COM image of the BDOS.

Load SYSGEN.COM with a debugger. On most systems, the BDOS image begins at 1200h. Compare the bytes there and in the running BDOS in high memory and then compare the bytes at the patch locations in the image (by adding 1200h to the addresses in the figures here) and in high memory. If all matches up, load the hex patch into the high BDOS, compare again, and then move just the patched bytes of the two upgraded routines to their corresponding location in the overlay:

```
MBDOS+0123,BDOS+0141,1200+0123
MBDOS+02D4,BDOS+02EC,1200+02D4
```

Then save the appropriate number of pages of the modified XSYSGEN.COM. Run XSYSGEN and place the system on the boot tracks of a scratch disk. Boot the disk, test the system for normal operation, and then with a debugger check the high BDOS to see that the patches are indeed in place.

Note that this patch will not work with ZRDOS or other replacement BDOSes. It may be possible to write a functionally equivalent ZRDOS patch, if you can find enough free space in a BDOS that is already in Z80 code. ■

Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these registered 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 marks 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, DosDisk; Plu*Perfect Systems; Clipper, Nantucket; Nantucket, Inc. dBase, dBase II, dBase III, dBase III Plus; Ashton-Tate, Inc. MBASIC, MS-DOS; Microsoft. WordStar; MicroPro International Corp. IBM-PC, XT, and AT, PC-DOS; IBM Corporation. Z80, Z280; Zilog Corporation. Turbo Pascal, Turbo C; Borland International. HD64180; Hitachi America, Ltd. SB180 Micromint, Inc.

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

ZSDOS
Anatomy of an Operating System
by Harold F. Bower and Cameron W. Cotrill

Harold F. Bower, Major, US Army Signal Corps. BSEE, MSCIS, Ham (WA5JAY), avid homebuilder (starting with 8008 running SCELBAL).

Cameron W. Cotrill, Vice President, Advanced Multiware Systems; specialist in "impossible" real-time hardware and software systems.

It has been many years since Digital Research released CP/M 2.2, and much has changed since that time in the speed, capability and flexibility of 8-bit general purpose microcomputers. CP/M is one of the few operating systems that runs on a wide variety of hardware, and is widely supported. Other than the current popularity of MS/PCDOS, UNIX and its derivatives have been the closest competitors to the universality of CP/M, with only the UNIX family being hosted on more hardware types. This hardware independence is perhaps the single strongest point of CP/M, and we venture to guess that more than a few PCDOS users have wished for this kind of flexibility. In recent years, however, better and faster 8-bit hardware has been more available, and has fostered vastly improved software such as BackGrounder ii, DateStamper and the ZCPR 3.X series of Console Command Processors. On such systems, CP/M 2.2 has been weighed in the scales and found wanting. The only widely accepted replacement for CP/M 2.2 is the ZRDOS family which added only slight improvements to the basic Digital Research structure. CP/M 3.0 (also known as CP/M Plus) added some new features, but was not easily retrofitted to existing systems, and did not achieve the popularity level of CP/M 2.2.

ZSDOS is our answer to what we perceive as a stagnation forced on the 8-bit computer community by a restrictive suite of Operating Systems. ZSDOS is a new BDOS replacement written in the spirit of the popular ZCPR command processor. Its roots come from P2DOS Version 2.1 by HAJ Ten Brugge of the Netherlands. P2DOS made significant strides in overcoming some of the limitations of CP/M 2.2 with such features as larger disk and file sizes, built-in CP/M Plus compatible file date stamping, and a Read-Only Path within the DOS for opening files. Use of attribute bits was expanded to include a modified recognition of the Plu*Perfect PUBlic bit and support for the Archive bit.

Despite the number of nice features adopted by P2DOS, several other desirable ones were not included. We and several others including Benjamin Ho (SUPRBDOS), C. B. Falconer (DOS+25), Carson Wilson (Z80DOS, ZDDOS), have sought to extend the functions of P2DOS by adding features and fixing bugs.

When BackGrounder ii with its task-swapping and cut and paste features was released, we found that even the modified P2DOS derivatives were unusable due to the very tight linkage needed between Backgrounder ii and the operating system. In independent contacts with Bridger Mitchell concerning the possibility of recognizing P2DOS as an "authorized" DOS for BackGrounder, we finally came to the realization that P2DOS was not stable enough. Seemingly everyone had source code and was changing it. Additionally, the added features were not much more beneficial than those offered by the ZRDOS family of operating systems.

Until Bridger suggested that we pool our resources, each of us were embarked on different paths. Cameron's primary thrust was directed at refining the fundamental P2DOS architecture and adding compatibility with Plu*Perfect's tools. Hal concentrated on correcting bugs, optimizing for speed and building tools to support the P2DOS (CP/M Plus compatible) file date stamping method. Carson Wilson joined the group because of his efforts to embed the entire DateStamper module into Z80DOS. ZSDOS was born of our decision to join forces and develop a complete DOS. The result was a pair of compatible DOSes; a full ZSDOS with complete capabilities, and ZDDOS which sacrifices some features for the integral DateStamper support.

ZSDOS was designed from the beginning to be compatible with existing applications. This sounds wonderful in theory but is no easy trick. To follow accepted standards, they must first be found, then understood to a degree sufficient to foil Murphy. Where no standards existed, a great deal of thought was given to future growth directions for CP/M compatible systems and how these would affect added functions. The first problem encountered was that there are no less that three BDOS standards; CP/M 2.2, CP/M 3.0 and ZRDOS. With much of the community using ZRDOS 1.7, we decided that it would be the final arbitrator of compatibility from a DOS call perspective. Functions not present in ZRDOS that resembled CP/M 3.0 functions would be similar in calling parameters and use the same function numbers. Much of the effort in ZSDOS involved defining the "real" standards. In the end, we arrived at what we hope are intelligent additions that will allow further evolution of CP/M compatible systems rather than hindering future growth.

In compatibility, ZSDOS rates very highly, though not perfect. First, ZSDOS requires that the system's BIOS preserve the IX register. This may present problems for some Osborne and Bondwell owners. Secondly, ZSDOS forces applications programs to "play by the rules" when dealing with file control blocks. Unlike CP/M 2.2 and ZRDOS, but like CP/M Plus, ZSDOS does not create or open new extents until they are really needed. Programs that attempt to second guess the DOS by looking at the record count field of the FCB and opening their own extents can get into trouble due to the 80H value present after the last record in the extent is read or written. In over a year of extensive testing with hundreds of CP/M and Z System applications, only two such programs were found; TDL's linker and MicroPro's ReportStar 1.1. In defense of ZSDOS, neither of these programs will run under CP/M Plus either.

When anything is altered within a DOS, there is a good chance of introducing more bugs than features. In applications programs, bugs are normally an annoyance. In a DOS they are a disaster (witness version X.0 of any Microsoft DOS if you doubt this assertion). DOS calls, therefore, must work as advertised, and any side effects of the calls must be properly documented.

The DOS features should also be logically consistent. One of the problems with P2DOS is that it attempted to alter long-standing conventions pioneered by Digital Research and Plu*Perfect Systems. While there were good arguments in favor of the conventions used in P2DOS, they are inconsistent with what we viewed as a cohesive set of DOS standards.

Several key philosophical concepts forming the basis for ZSDOS have already been mentioned—compatibility with existing systems and enhancements that make sense. The overall goal of ZSDOS is that the DOS must not "get in the way" of a user unless absolutely necessary. By getting in the way we mean that the DOS does not respond in the desired manner, or prevents seemingly logical actions from being properly executed. Any time a command file (e.g. editor, assembler, etc.) is executed from one drive/ user area and not another, DOS prevents it by getting in the way. When DOS requires a user to tell it that a disk has been changed, DOS is getting in the way. When a user can't tell which file is the latest version, DOS is getting in the way.

The second guiding concept in ZSDOS is that the operating system is a tool. To this end, ZSDOS has a multitude of options never before available to the user. These options can be configured on the fly from the command line, command scripts, or interactively. In all cases, ZSDOS responds instantly to these changes.

The following sections provide details on changes that ZSDOS brings to the user of 8-bit computer systems. Unless explicitly stated, references to ZSDOS also include ZDDOS. The topics assume a passing familiarity with CP/M 2.2 and ZRDOS.

Automatic Relog of Changed Disks

One of the first areas tackled in the development was automatic accommodation of changed disks. Benjamin Ho was one of the first to attempt a fix to P2DOS in his SUPRBDOS which implemented a changed disk detection method consisting of a flag which was set when the directory checksum of a disk didn't match the previous value. Relogging was then performed at a later time. This approach has been used in several systems (Z80DOS, SUPRBDOS, and P2DOS+) and seems to work as intended. Cameron had a problem with this method—the DOS relinquishes control without the problem being corrected. This put the DOS in an invalid state, and breaks a guiding principle of real-time control systems—**NEVER** leave the system in an undefined or invalid state!

ZSDOS uses a more difficult approach that relogs disks at once when a change is detected. After slogging through the existing code, it became apparent why no one else has succeeded in finding a better method—the control flow is very difficult to follow, and it is nearly impossible to retrace the code, relog, then pick up where the change was detected. Cameron, having accomplished "impossible" programming tasks before, once again displayed his stubborn streak and succeeded in correcting this flaw by using a tool normally reserved for high-level language programmers: recursion. This code was one of the first major modifications to P2DOS and has now been running successfully for nearly two years.

While developing ZSDOS, our original goal was to give users a warning that a changed disk had been detected **only** if ZSDOS was ready to write to an already opened file. It turns out that many programs successfully open more files than they close. Thus, counting opens and closes was not a viable solution. ZSDOS would have to track every FCB used by user programs in order to stand a fighting chance of sorting out whether or not to warn. This sloppy coding style seems endemic to the entire Personal Computer programmer community, and causes severe problems, particularly in multiuser and multitasking systems. ZSDOS consequently warns the user every time it detects a disk change. In the spirit of not "getting in the way", however, the message can easily be suppressed with the disk being automatically relogged. A little extra head activity on the drive will be the only noticeable effect.

Time and Date Stamping

Probably the most desired, yet least standardized, DOS feature is time and date stamping of files. Within the 8-bit CP/M community there are currently three known methods, all mutually incompatible. The closest to a universal standard at this point is probably Plu*Perfect's DateStamper® . This is an add-on time/date stamping system which offers no DOS level functions, but can be installed on practically any CP/M compatible machine, with or without a real-time clock. It is also the most complete stamper available featuring times and dates of file creation, modification, and last access. DateStamper is well supported with file copy programs, directory and file utilities, disk catalog programs, etc. Finally, it consumes only one directory entry while the other two stamp methods consume ¼ of your available directory space. On the negative side, it is slower than the other methods due to the need to open, update, and close the datestamp file when storing the stamp information.

The other two known methods, P2DOS style (compatible with CP/M 3.0) and DOS+25, are very similar. Both use the fourth entry in each directory sector to hold the time and date information for the three file entries in the sector. Right away, one-fourth of the directory is consumed. While this may not be a problem for diskettes or hard drives that have 128 or more possible entries, it is definitely a problem with the 64 entry limit of systems such as the Kaypro. These two stamp methods are quite fast and add very little to DOS overhead because the directory entry is already in position when stamps are applied. The two methods differ only in the format of the stamps. The CP/M 3.0 type supports times and dates for Create and Modify only, while DOS+25 supports times and dates for Last Access and Modify, and dates only for Create. A disadvantage is that both of these schemes require an additional BIOS vector to function.

While ZSDOS can, theoretically, manage nearly any type of time and date stamp construct, we have only implemented DateStamper and P2DOS schemes. This versatility is accomplished by placing method specific and clock code in BIOS or other protected memory, and by vectoring all DOS time stamp functions through a table in the configuration area of ZSDOS. By inserting addresses of the method specific service routines in this table, ZSDOS maintains control of program flow, calling stamp routines as needed. Vectors are currently provided for Get/Set Time, Get Stamp, Put Stamp, Stamp Create, Stamp Access, and Stamp Modify. Only the routines needed to support the required stamp method need be implemented. For example, all six vectors must be set in a full DateStamper system with ZSDOS, while only the Get/Set Time is required in the ZDDOS version. All but the Stamp Last Access vector, which is disabled, are needed in a P2DOS stamping system.

ZSDOS does most of the required housekeeping in all stamp related calls to simplify the implementation. When external time stamp functions are called, ZSDOS has already loaded the directory buffer with the record containing the required entry. If a directory write is required, the drive has already been tested for Read/Write status. ZSDOS passes all necessary information to the external routines which are responsible for getting the time (if required), converting the stamp between universal format and target format, and placing the stamp where it belongs. Only in the case of stamp last access or put stamp is the BIOS actually required to call the write directory record routine. The external routines all return a status code to notify applications/users of the operation success or failure. When portions of the stamping scheme have been disabled, or have not been loaded, an error status is returned from dummy routines addressed by the unimplemented vectors in ZSDOS.

To support file stamping in a coherent manner, ZSDOS has adopted four new function calls. Functions 98 and 99 are Get Time and Set Time respectively. A six-byte packed BCD construct is used to pass two-digit year (arbitrarily decided as 1978 to 2077), month, and day as well as hours, minutes and seconds values. Functions 102 and 103 are the Get Stamp and Put Stamp calls respectively. Key to the way ZSDOS handles these functions is that we have defined a format that is independent of the stamping method and can handle all present, and probably any future stamping method. Stamps are defined as three fields of five bytes each. Each field is comprised of packed BCD digits for Year, Month, Day, Hour and Minute, in the same manner as the Time returned from the clock, less the Seconds byte. Any conversions needed between this "Universal" format and the native stamp format is performed in the external stamping module. Since this is the native DateStamper format, ZDDOS operates directly on the data. We hope that this format will provide a meeting ground for the diverse methods and provide a badly needed standard applications interface.

The six byte BCD format was adopted (after considerable debate) because it simplifies the code for applications that deal with the stamps. The order of the stamp lends itself readily to sorting by date. The BCD format makes conversion to Ascii display values a snap. The fact that it is the same format as the de-facto CP/M 2.2 standard, DateStamper, was also a key factor in the decision.

Enhanced Error Handling

Error handling in most CP/M compatible operating systems left much to be desired. CP/M 2.2 has been infamous for cryptic error messages. While ZRDOS attempted to be somewhat less so, it required users to look up error numbers in the manual (or via an online utility) to decipher the meaning. CP/M 3.0 for the first time provided truly useful error messages, reporting errors in something approaching plain English. Feedback includes the type of error, the function number which triggered the error, the drive in question, and the name of the file (if one was used). P2DOS and its derivatives adopted these messages but in P2DOS even Bad Sector errors which could be ignored under CP/M 2.2, caused an immediate warm boot. ZSDOS adopts the CP/M 2.2 recovery methodology added in most of the enhanced versions of P2DOS and continues the concept of plain English error messages.

CP/M 3.0 also has a another very desirable feature which was never incorporated in CP/M 2.2 compatible BDOSes. With a new function 45, all errors can be returned to the application program including Select errors and Read Only errors. In this mode, errors return a unique code revealing the exact cause. Applications can even tell CP/M 3.0 to suppress all DOS error messages. This seemed to be a highly desirable feature, so we added it to ZSDOS.

ZSDOS also reverts to strict CP/M compatibility in another obscure area, not following the path taken by ZRDOS. ZSDOS maintains a vector table for errors in the same location used by CP/M 2.2. This allows the few programs that patch the table, such as bad sector lockout programs, to operate properly.

Function 37 Fixed

Probably the most troublesome of all functions provided by CP/M 2.2 was function 37. Many wrongly assumed that function 37 would relog a selected disk after performing the stated function of resetting (unlogging) all disks. It didn't! This anomaly caused many blasted disks and infuriated users. In accordance with the ZSDOS spirit, it seemed reasonable that if the default drive was logged out, it should be automatically relogged after a function 37 call. This is now the case with ZSDOS.

Global Availability of Files

One of the most frustrating aspects of most microcomputer operating systems is context sensitivity. The classic model for computer operating system file management—that of the ordinary filing cabinet—neglects one very important point. The phone, the pencils and pens, and the myriad of other tools found on most desks are not kept in the filing cabinet! Yet, on a vast majority of our file management systems they are. Clearly, some distinction should be made between tools and data if the operation of the system is to approach the intuitive level. One should be able to log into an area where data resides and have all the system tools available— without the need to either remember where they are or inform each and every application where it lives.

The ZCPR family of Console Command Processor replacements made tremendous strides in freeing user dependence on strict drive and user area specification by creating a Search Path that the Command Processor uses to find Command files. Users were thereby able to use files spread across the computer's complement of drives with the privacy of user areas by specifying a Path consisting of a list of drive and user numbers. Unfortunately, the Path became inactive once command programs began executing unless the application program is specifically written to locate and use the Path. Programs using overlays (including most of the programs for which we bought computers in the first place) printed nasty messages about missing overlays and dumped us unceremoniously back to the Command Processor when we relied on Path access. Thus, Path alone wasn't the answer to context sensitivity.

In 1984, Bridger Mitchell and Derek McKay of Plu*Perfect systems published the PUBlic patch for CP/M in Dr Dobb's Journal. With this patch, any file having the F2 bit (most significant bit of the second character of the file name) set could be accessed from any user area on the disk from BDOS, provided that it was specified with an unambiguous name. This solved the overlay problem when logged on the same disk as the overlays. It was also fast with only one scan of the directory required to locate the file! However, PUBlic files disappeared from the directory—at least according to directory utilities that didn't know how to look for them. Thus PUBlic was a major step forward but again was not the total solution.

P2DOS was the first real attempt at a coherent solution. Path was implemented, along with a modified form of PUBlic from within the BDOS. By setting F2, a file was made Public, just as with the CP/M patch. By setting the system bit, Path was allowed to find the file. Thus, the user had complete control over the method of accessing files. Both Public and Path accesses were read only, and Path only worked for file opens. Public files even appeared in directories! Both limitations were reasonable, but there were bugs. The DOS routine that checked for file read only status had several serious bugs that prevented erasure of files when it should have been allowed by the Plu*Perfect definition. Also, due to the way the erase routine was implemented, erased Public files continued to be found on directory scans! Third, strange file R/O errors would randomly occur if the last file in a directory was declared Public. Finally, if ZCPR's Path was used, the entire Path would be walked N^2 times if the file was not found!

ZRDOS attempted a different solution by providing a mechanism for declaring complete directories Public. A map is maintained of disks containing Public directories and user numbers declared Public. This arrangement is best visualized as a grid, with user numbers on one axis and directories on the other. Wherever the Public declarations for drive and user intersect on the grid, all files in the associated directory are Public. As with Plu*Perfect PUBlic, all Public files disappear from directory listings. Since this means everything in the directory, entire directories appear empty! Since access is R/W, accidental erasure or overwriting of files can be a real problem. From a performance aspect, it appears that one disk directory scan is required for each user area declared Public. Thus, considerable time overhead may be required in finding Public files if more than one user area is declared Public. The fact that a PUBlic patch for ZRDOS has been developed bespeaks the fact that ZRDOS hasn't found the magic combination either.

Our approach in ZSDOS was to build on P2DOS's framework and remove the idiosyncrasies and bugs. The P2DOS approach had several very strong points. The access mode of the files was totally under the control of the user on a per file basis (unlike the per directory basis of ZRDOS), with only one directory scan per drive required, and P2DOS used two de-facto standard structures: Path and PUBlic. After many gyrations, we finally settled on the "best" definitions for Public and Path. For a start, we allow run-time configuration of the ZSDOS access modes—of which there are five in ZSDOS and two in ZDDOS (Path being sacrificed for the integral DateStamper)! Though these five modes are somewhat contrived, they serve to provide a way of understanding the options offered by ZSDOS which may be set on installation and altered on line at any time.

The first access mode is the default mode. This is the way a standard CP/M 2.2 system accesses files. If the file is not found in the logged drive/user area, an error is returned. No other attempts are made to find the file. ZSDOS always tries this access mode first regardless of the access options enabled. This prevents files or entire directories from "disappearing" on the unsuspecting user. Wildcard file specifications are only allowed for this mode, not for any other.

The second mode is the Path Directory Access mode featured only in the full ZSDOS. A search Path (which may be a ZCPR3 Path), an internal three element Path or a user defined Path is scanned to find the file. In this mode, all files in any directory along the Path may be located by using an unambiguous file name. This mode is similar to the ZRDOS Public mode.

The third access mode is the Path File Access mode, again only contained in the full ZSDOS. As in the Path Directory mode, the DOS Path is used to search for files. This mode, however, requires that the SYStem attribute bit be set for the file to be found. In the Path File Access mode, the user has more control over how particular files will be accessed. Entire directories need not be made globally available, just those files with the SYStem attribute bit set.

The fourth method is the Public access mode featured in both the ZSDOS and ZDDOS versions. This mode follows the Plu*Perfect PUBlic definition and requires the F2 attribute to be set to find the file. Unlike the original Plu*Perfect patch to CP/M 2.2, Public files are "visible" if you are logged into the same drive and user area as the file. The Public Access mode is very efficient for finding unambiguously specified files, correctly locating them on the first disk directory scan. One precaution to be observed with Public, however, is that only one file of the same name is permitted for any given disk. ZSDOS comes with appropriate support tools to enforce this rule.

The fifth method is called the Combined Access mode. As in the other modes requiring a Path, it exists only in the ZSDOS version. Rather than being a distinct mode, it is selected by simultaneously setting options so that either Public or Path can access files. ZSDOS first uses Path to select a drive, then locates Public files on that disk in the first directory scan, regardless of the user area indicated in the Path. File access time is thereby significantly reduced for systems where files are spread over several drives and user areas.

One of the thorny questions we had to deal with was whether to restrict access of Public and Path files to read only. The original Plu*Perfect definition of PUBlic allowed writes, while P2DOS did not. After many discussions with Bridger Mitchell and others, we decided to allow read/write access to both Path and Public files, but we didn't stop there. Many users are understandably wary of the accidental file erasure or overwriting possible with both Path and Public. To accommodate these users (we number ourselves among them), ZSDOS allows user selection between Read/Write and Read Only file accesses. As with most ZSDOS features, this selection can be set or changed at any time. The restriction to Read Only does not affect access to files in the currently logged drive/user area.

Path access is necessarily restricted to the Open File function in order to prevent global chaos. Once successfully opened, the file will always be found again if the application doesn't alter FCB + 0 and FCB + 13. Actually implementing a Writable Path turned out to be one of the biggest compatibility problems we faced in the design of ZSDOS, and it was one of the last major features to be added. Formally defining a use for the byte at FCB + 13 marks a departure from the accepted definitions of "undefined" and "reserved" in previous DOSes, but provides an upward compatibility with new system components such as ZCPR Version 3.3 and later. Also, providing the user with access to the F7 attribute bit to determine whether files were found by Path and/or Public is another deliberate step in providing software developers with additional tools to write responsive programs.

User Number Storage in the FCB

In making the Path Writable, we had to break new ground with ZSDOS. While most ZSDOS features are refinements of existing ideas from other DOSses, the concept of user number storage in the FCB is borrowed from ZCPR Version 3 and is new for a DOS. As explained above, Path is only active in ZSDOS on file Opens, and only resolves the drive containing the file. Unless the Public bit on the file is set or we are logged in the right user area, subsequent accesses to the file will fail. Our answer to this dilemma is to have ZSDOS keep track of the user area in which the file was found using a trick from ZCPR3.

Version 3.0 of ZCPR began to use the previously "reserved" byte at FCB + 13 to store the user number of files several years ago. This byte, known as S1 in CP/M 2.2 and ZRDOS, was listed as "reserved" but unused until ZCPR began using it when parsing file specifications. ZSDOS also uses this byte to store the user number for the file. Simple? Far from it! Problem #1—how do you distinguish between a valid user 0 and an FCB from an application that knows nothing of users and sets this byte to zero because the Digital Research documentation said to? The answer is fairly simple; use a bit as a user valid flag since only 5 bits are required to hold user numbers. Just check this bit. If it's clear, DOS has only to place the current user at this location, set the bit and the problems go away. . right? If only life were this easy!

There is a little known corollary to Murphy's law which states that Murphy is an optimist! We can attest to its validity after solving this problem. Digital Research did not specify that the S1 byte should be initialized prior to file opens. Consequently, many programs reuse FCB's without bothering to clear the S1 byte. When these recycled FCB's are used to call ZSDOS, the (usually wrong) user number is already resolved in the FCB!

After much head scratching and experimentation, only one method provided the necessary backward compatibility. The user number in S1 is ignored on file opens **UNLESS** an application explicitly says that it knows about user numbers, and wants to work with them. We call these **ZSDOS application programs**, and they signal their presence by setting the BDOS error mode before executing file access commands. Pseudo code for this modified operation is:

```
SELECT DRIVE FROM FCB:
        IF there's a ''?'' in FCB+0
                GOTO function 14 (FCB+13 not altered or examined)
        ELSE
                SELECT drive from FCB
                IF D7 of FCB+13 is clear
                        Get default user
                        OR with 80H
                        Place in FCB+13
                ENDIF
                GET FCB+13
                AND with 7FH
                Store in FCB+0 for SEARCH
        ENDIF

SEARCH: (does not use FCB+13 - user is in FCB+0 at this point)
        IF file not found AND it's NOT open a new write extent
                Clear bit 7 of FCB+13
        ENDIF
```

So long as an application doesn't fiddle with FCB+0 to FCB+13, after a file open, this method works very well. For applications that do, logging in where your data is and using explicit DU: specifications to grab data from other drive user areas (just like you've been doing all along) will make even these perform correctly. A shining example of a program that is ill-behaved in this manner is MLOAD.

Note that the access modes in ZSDOS are primarily designed to load applications and overlays—not find data files. By now, the reasons for this design should be clear. While there is nothing in the design of Path and Public as implemented in ZSDOS to prevent fetching data, our primary concerns were in backwards compatibility with programs that insist on altering FCB's once files have been opened.

Fast Relog of Hard Disks

Another enhancement provided by ZSDOS is fast relog of fixed disks. This again is simple in concept, but if all potential problems with relog are considered, things get more complex. In a normal CP/M system, every time a disk is reset, DOS clears its bit map showing what blocks on the disk are allocated. When the disk is logged in, the allocation vector is rebuilt by scanning the entire directory. This takes time, particularly on large hard disks that have 1,024 or more directory entries. When a non-removable media drive such as a hard disk is present, the allocations do not change. Therefore, clearing the allocation vectors and scanning the directory can be avoided after the initial selection.

DOS has all the information it needs to determine whether a disk contains removable media or not. The disk parameter header returned to DOS after a BIOS select contains a pointer to a scratchpad area known as the Work Area for Changed Diskettes (WACD). Each disk with removable media must have a WACD. If this pointer contains 0, the drive by definition contains non-removable media. So far, so good. All DOS has to do is build a vector defining which drives in the system are fixed disks and avoid relogging those drives.

Now for another Murphyism. First, it is not always true that allocations won't change without DOS's knowledge. Any time the BIOS is directly manipulated, it can change disk allocations without giving DOS a clue. Examples of programs that do this are PACK, PUTDS, and PUTBG. Clearly, a method of signaling DOS to rebuild the drive allocation is required. Fortunately, ZRDOS (starting with V1.5) defined such a method. When a drive is logged out using function 37, the hard disk login vector is also cleared for that drive, forcing the allocation vector to be rebuilt.

Now for the worst problem with the fast relog. Some BIOSes such as the Advent TURBOROM BIOS for Kaypro and the AMPRO BIOS support remapping of logical drives. For example, assume that at boot time, disks A through D are floppies and F through I are hard disks. On one of the BIOSes mentioned above, you could tell the BIOS to swap drives so that A through D are the hard disk and F through I are floppies. Since this is a BIOS function, DOS has no idea what has happened behind its back. If BIOS swap programs would reset these drives with a function 37 call, all would be well. Unfortunately, they don't.

Now the compatibility issue raises its head again. We can't expect all the swap utilities to be rewritten or patched, nor do we want to depend on the user invoking custom programs such as RELOG or DSKRST. If drives can be swapped, DOS needs some method of discovering this and coping with it once discovered. The method chosen, while not perfect, provides protection in all but one rather unusual case. Since DOS knows that a WACD pointer can tell which drives are fixed and which are not, this information is combined with the state of the hard disk login vector to tell if BIOS has played games behind our back. Pseudo code for this procedure is:

```
IF WACD indicates hard disk
        IF it's already logged in as a hard disk
                EXIT
        ELSE
                Log in as hard disk and build allocation
        ENDIF
ELSE it's removable so
        IF it was logged as a hard disk before
                Clear all hard disk allocations
        ENDIF
        Build allocation vector
ENDIF
```

Those who looked over this carefully have probably discovered the one problem ZSDOS can't detect; the case of two hard disks being swapped. In all of our head scratching, we couldn't think of a simple, reliable method of doing this (if you know of one, let us know!). However, this is a rare situation and ample warning is given in the ZSDOS documentation. To be compatible with ZRDOS function conventions, the hard disk login vector can be examined using function 39.

Run Time Configuration

The FLAGS byte originated in P2DOS has been extended in ZSDOS to allow more complete user control over the DOS. Currently, bits in the FLAGS variable control whether Path and Public are enabled, what Public mode is selected, whether the Fast Fixed Disk Relog is used, and whether the disk write protect buffer is flushed by a warm boot. In addition to setting default conditions for these parameters during installation, they can be dynamically altered with a supplied utility, or by applications programs using the new functions 100 and 101. These two functions Get and Set the FLAGS respectively in a standard and published way. While FLAGS is currently a BYTE value, it is defined and passed as a 16-bit WORD value with the upper byte set to zero. Future expansion is thereby accommodated with minimum changes to defined parameters. There should be no more need to use undocumented patches so commonly found with CP/M 2.2 and ZRDOS.

Expanded Attribute Support

Attribute bits are the Most Significant Bits in the name and type of files used in file specifications. They are denoted by "F" for file name, and "T" for file type, and a number for the specific byte in the field. Where CP/M recognized only two attribute bits, ZSDOS supports six of the possible eight. The Read Only (T1) and SYStem (T2) bits are retained with their original definitions. The popular Archive bit (T3) is supported as well. The Archive bit is automatically cleared by DOS when files are created and written to, and set by many file utilities such as ARCHIVE and BU. Attribute F2 is used to identify Public files in accordance with the Plu*Perfect standard covered above. Additionally, attribute F3 is used with the DateStamper file stamp method to tell ZSDOS not to update the Last Accessed time field. Finally, attribute F8 is used to show that the file is restricted to users with Wheel access privileges.

Enhanced Write Protection

Protection against inadvertent writes has been increased in both areas of disk and file. Function 28 may be used to declare an entire disk as Read Only. In both CP/M and ZRDOS, this status reverts to Read/Write on a Warm Boot. Under ZSDOS, an option exists to declare the Read Only vector as permanent. In this mode, Read-Only Sustain, only reloading the DOS (or turning off the R/O sustain bit in FLAGS, then warm booting) will restore the drive to Read/Write status.

File write protection has also been strengthened by methods related to the added ZSDOS features as well as retaining the old Read Only attribute (T1) bit. Prohibiting writes to Path and

Public files is another method previously described. The final method to protect files is by using another concept from ZCPR, the Wheel Byte. We added a vector to the configuration area of ZSDOS which may be set to any arbitrary address, including a ZCPR Wheel Byte to control file writes. ZSDOS follows the same conventions as ZCPR where a null (zero) value in the Wheel byte means that the user is not a "Wheel" and cannot perform certain privileged activities, including erasing or writing to files marked with the Wheel Attribute (F8) bit. Disabling the Wheel protect feature of ZSDOS has the effect of granting full access to the user by setting the vector to a non-zero value.

Revised Console I/O

CP/M originated at a time when the Teletype Model 33 was a mainstay console for computers. The vast majority of terminals are now video based, and do not need the teletype editing sequences. We therefore decided to follow the lead of ZRDOS and many P2DOS hackers, and change the interface. Thus, ZSDOS no longer supports Control-E. The Rubout (7FH) and Backspace (08H) characters are both treated as destructive backspaces in function 10, Read Console Buffer. This was done in a way that does not require patching of such programs as WordStar. For the benefit of bulletin board operators, Control-U is synonymous with Control-X since the latter is a special character in many protocols. Also for remote operators, Control-R (retype line) has been retained in the full ZSDOS. ZDDOS sacrifices this feature to provide space for the embedded DateStamper.

Function 6, Direct Console I/O, was also enhanced with the addition of a **Get Console Character** function. This operates as function 1 except that no character checking (such as Control-C Warm Boot trapping) is performed. The code was also revised to use a one character type-ahead buffer in a manner that permits free mixing of function 1 and 6 calls in a totally reliable manner.

Other Features and Tradeoffs

In addition to the features already mentioned, ZSDOS supports the ZRDOS return DMA pointer function (function 47), and a ZSDOS return version call (function 48) which is a logical extension of the ZRDOS call. ZSDOS is also compatible with the reentrancy requirements of ZRDOS 1.7 for rudimentary multi-tasking support such as that required in some Input-Output Packages for ZCPR 3.

ZSDOS features larger disk and file sizes by logically and transparently extending the original CP/M constructs. Files may now be as large as 32 Megabytes with random access files consisting of as many as 262,144 logical records. Disks may contain 1,048,576 kilobytes, or one gigabyte!

True to the original goals, ZSDOS is fully compatible with the current suite of advanced tools such as BackGrounder ii. While an overlay is provided for existing versions, newer revisions will automatically recognize ZSDOS and permit direct loading. We also concentrated on providing a complete repertoire of support tools and utilities to take control of the various features. Installation tools and procedures are provided for overlaying ZSDOS onto a MOVCPM image, installing ZSDOS with NZ-COM, XBIOS, and Plu*Perfect's JetLDR. Tools are provided for: Directory listing of files with date stamps added by DateStamper, P2DOS, and DosDisk; file copying and archiving with date stamp preservation; Path manipulation; and run-time tailoring of all ZSDOS options. Additionally, the full set of over 40 clocks recognized in the Plu*Perfect library (as well as new drivers for the Ampro Little Board, several SB180 clocks, and the Oneac ON!) are supported with easy-to-use interactive installation tools.

Now, how did we get all this into a DOS that originally had only 7 bytes left? Needless to say, nearly every trick in the book was used (and some we have **NEVER** seen in any book!). This gave us some room, though much of the effort was still slogging line by line through the code in tedious manual optimization. Both of us are somewhat proficient in code crunching (Hal having done a thesis on optimizing compilers and Cameron having used Z80's for a number of years in real time control systems). It took the best efforts of both of us to chop things down to size. What we missed, Bridger Mitchell, Joe Wright, and Carson Wilson found in their review of the code—but it wasn't much. Some details of this effort will be covered in the next part of this article along with some performance results and code fragments to show how to use the advanced capabilities of ZSDOS. ■

Editor

(Continued from page 3)

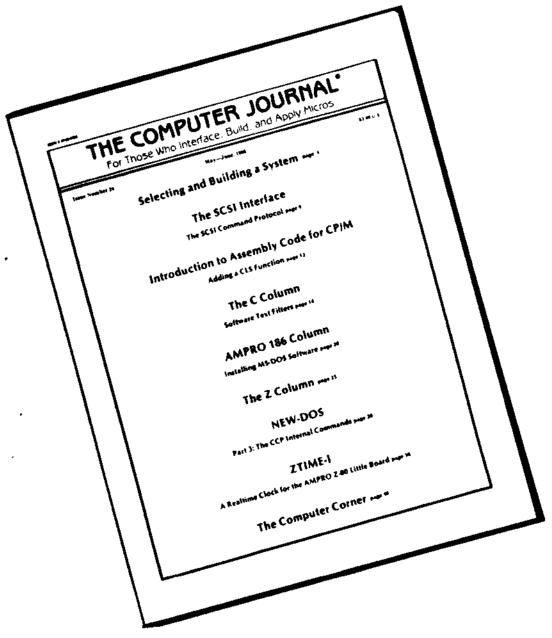
dealers may have to become vertical market VARs (Value Added Reseller) and consulting houses which only deal with the customers who are intelligent enough to value (and pay for) their services.

What are your experiences and opinions on this?

What is CAD?

The term CAD is currently being used for at least three functions which are similar but different. They are Computer Aided Drafting, Drawing, and Design. Software which is best for one of these functions will perform the other two functions poorly, if at all. Some publications have directly compared drafting programs with design programs in a design application which made the drafting program look bad. This is like trying to use a pipe wrench and a socket wrench for the same job. It is wrong because they are different tools designed for different applications.

An upcoming article will contrast the three functions with an example of a software package for each function. ■



THE COMPUTER JOURNAL
For Those Who Interface, Build, and Apply Micros

Selecting and Building a System

The SCSI Interface
The SCSI Command Protocol

Introduction to Assembly Code for CP/M
Adding a CLS function

The C Column
Software Text Filters

AMPRO 186 Column
Installing MS-DOS Software

The Z Column

NEW-DOS
Part 3: The CCP Internal Commands

ZTIME-1
A Realtime Clock for the AMPRO Z-80 Little Board

The Computer Corner

THE COMPUTER JOURNAL

Back Issues

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 19:

- Using The Extensibility of Forth
- Extended CBIOS
- A $500 Superbrain Computer
- BASE: Part Seven
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 2
- Multitasking and Windows with CP/M: A Review of MTBASIC

Issue Number 20:

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

Issue Number 21:

- Extending Turbo Pascal: Customize with Procedures and Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition and 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 Index
- The AMPRO Little Board Column

Issue Number 24:

- Selecting and Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assembly 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 Realtime Clock for the AMPRO Z-80 Little Board

Issue Number 25:

- Repairing & Modifying Printed Circuits
- Z-Com vs Hacker Version of Z-System
- Exploring Single Linked Lists in C
- Adding Serial Port to Ampro L.B.
- Building a SCSI Adapter
- New-Dos: CCP Internal Commands
- Ampro '186 Networking with SuperDUO
- ZSIG Column

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: The CCP Commands Continued
- 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 and 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 for 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 L.B.• 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 L.B., part one.
- 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 L.B.
- 3200 Hacker's Language
- MDISK: 1 Meg RAM disk for Ampro LB, 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 and their variations.
- XBIOS: A replacement BIOS for the SB180.
- K-OS ONE and the SAGE: Demystifing 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 O.S. 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: Part 2, 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, and how to write self relocating Z80 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 and extnded DOS services.
- ZCPR3 Corner: ARUNZ, Shells, and patching WordStar 4.0

Issue Number 34:

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

Issue Number 35:

- All This & Modula-2: A Pascal-like alternative with scope and parameter passing.
- A Short Course in Source Code Generation: Disassembling 8086 software to produce modifiable assem. souce code.
- Real Computing: The National Semiconductor NS32032 is an attractive alternative to the Intel and Motorola CPUs.
- S-100 Eprom Burner: a 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.
- ZCPR3 Corner: How shells work, cracking code, and remaking WordStar 4.0.

Issue Number 36:

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

Special Close Out Sale on these back issues only

Issue Number 1:
- RS-232 Interface Part One
- 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 Two
- 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 the 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 6:
- Build High Resolution S-100 Graphics Board: Part 1
- System Integration, Part 1: Selecting System Components
- Optronics, Part 3: Fiber Optics
- Controlling DC Motors
- Multi-User: Local Area Networks
- DC Motor Applications

Issue Number 8:
- Build VIC-20 EPROM Programmer
- Multi-User: CP/Net
- Build High Resolution S-100 Graphics Board: Part 3
- System Integration, Part 3: CP/M 3.0
- Linear Optimization with Micros

Issue Number 16:
- Debugging 8087 Code
- Using the Apple Game Port
- BASE: Part Four
- Using the S-100 Bus and the 68008 CPU
- Interfacing Tips & Troubles: Build a "Jellybean" Logic-to-RS232 Converter

Issues—1, 2, 3, 4, 6, 8, and 16
3 or more, $1.50 each postpaid in the U.S.
Outside of the U.S., 3 or more, $2.50 each postpaid surface.
Other back issues are available at the regular price.

TCJ ORDER FORM

Subscriptions		U.S.	Canada	Surface Foreign	Total
6 issues per year ☐ New ☐ Renewal	1 year	$16.00	$22.00	$24.00	
	2 years	$28.00	$42.00		
Back Issues		$3.50 ea.	$3.50 ea.	$4.75 ea.	
Six or more		$3.00 ea.	$3.00 ea	$4.25 ea.	
#'s					
				Total Enclosed	

All funds must be in U.S. dollars on a U.S. bank.

☐ Check enclosed ☐ VISA ☐ MasterCard Card # ____________

Expiration date__________ Signature ____________

Name ____________

Address ____________

City____________State________ZIP ________

THE COMPUTER JOURNAL
190 Sullivan Crossroad, Columbia Falls, MT 59912 Phone (406) 257-9119

#37

(Continued from page 48)

only employment section. It is a good way to get Forth help faster than the mail or magazines.

Lastly is go to a Forth Convention, there is always a lot to learn there (more later on the latest meeting). Let's not forget local Forth interest groups. Our group in Sacramento is small but friendly and very helpful. Well worth the once a month we meet and discuss events. It is in these meeting where you find out which devices are best and who has what software to solve the problems.

Again, thank you for your letter and keep us informed of your experiences with Forth.

The Forth Real Time Convention

This year the Forth community held its convention in Los Angeles. The main theme was Real Time use of Forth, as well as Real Time activities in general. I was unable to go, but have talked with one of the other Forth members to get his opinion of the affair. My main interest was who won the real time contest. There was a $1,000 prize for the first person or group that could make their computer talk to and control a GIZMO. The actual gizmo was kept a secret so nobody could do any preplanning.

The gizmo turned out to be a type of metronome. A solenoid controlled the swing of a hack saw blade, while a LED display could scroll out the text you would sing along with. There were 4 teams and 6 individuals trying to win. Each had their own unit and the winners (a team of two) took 110 minutes to make it work. It was done in Forth on an Amiga (any language could be used), and my friend said it was quite exciting watching people as they got more and more functions to work.

Seminar Reports

A number of topics were discussed in the seminars and one item that is currently hot is multi-tasking. The current issue is that most want to do pre-emptive and not task switching. The difference as I understand it, requires a major change in how the kernel is actually put together. A rather large number of Forth main features must be given up to use pre-emptive multi-tasking. My feelings and those of others are based on why you use Forth. It is just those features which need to be given up that make Forth so great. So one can see rather quickly that pre-emptive multi-tasking will be a big problem for some time. I am not sure I understand all there is to know about multi-tasking, so I will try and explain it in detail next time around.

A talk by Ray Duncan, a Doctor turned Forth user, explained how he got started using Forth. Seems he spent 6 or more years writing a program in assembler, and then saw Forth demonstrated. That demo convinced him to go Forth. One of his complaints (and mine) is that too many people become convinced Forth is the only language. I might add that others like "C" as the only true language. The main point Ray made, and which I strongly support, is that you need to adjust your choice of language to the time frame to complete the job, the number of people involved in the project, your own programming style, cost, functionality, tools, and to what level you need to reach in the task. Sometimes Forth may win out, some times other languages. This doesn't make one better than the other, just more appropriate for the task at hand.

Charles Moore (the inventor if you will of Forth) gave a "fire side chat" as usual, and my friend noticed one interesting comment. He noticed Chuck (Charles Moore likes being called this) commented on the need for memory. He feels your program should take as little space as possible, leaving the rest for data storage. Those are good ideas to consider, as I am sure more than once, we have all loaded a program only to find not enough memory left to do any form of work. What good is a program that doesn't leave any room for your application, not very good!

Hardware Vendors

I understand that there were not a lot of new vendors present. Silicon Composers did talk about a few of their new products. Silicon Composers currently makes PC plug in boards for the Novix and RTX2000. They have also produced a product called SC/FOX. This is a standalone RTX2000 system for $1,195, that includes software. It still seems a bit high in price, but is cheaper than the $4,000 they charge for the RTX plug in board. Most of the problems start with the high price of the RTX CPU. They also plan on producing a board with the WISC chip as soon as production of the device is sufficient.

Harris announced the availability of their RTX2001. This cheaper version is smaller and does not contain the MMU (memory management unit). In quantity it is suppose to be around $50 each. Now that is still much higher than a Z80, or even the Z180/64180 which have many of the other RTX features like serial ports and I/O ports. What you do gain is direct Forth interpretation and incredible speed from the RTX/Novix.

Some Final Words

I hadn't intended to talk only about Forth this time around, but once I started... My work with WordStar 5.0® and Wordperfect 5.0® is still going on. I had to remove WS 5.0 for my wife, it was just too many changes too fast for her to handle at once. I think given some time, without a deadline to meet, and she would like some of the changes. I have found some bugs in the printer operation and don't like some of the changes which make it closer to Wordperfect operation. An example is the justification on/off option. Using it now puts a .OJ in the text as you type, compared to the before where it was a screen type option. I prefer the old way, thank you anyway WordStar.

Well I promised Art this article early because of the holidays and so any more comments are just going to have to wait till later. Hope all your holiday activities were safe and pleasant.

For more information and help on Forth try these sources:

Forth Interest Group
PO Box 8231
San Jose, CA 95155
(408)277-0668
Dues: $30/yr.

Genie Network
FIG section #700
Over 1200 Forth files on line
(800)638-9636
Special subscription fees for FIG members

Micro Cornucopia
PO Box 223
Bend, OR 97709
(503)382-5060

The engines mentioned:

New Micros, Inc.
808 Dalworth
Grand Prairie, TX 75050
(214)642-5494

Bryte Computers, Inc.
PO Box 46
Augusta, ME 04330
(207)547-3218

Silicon Composers, Inc.
210 California Ave, Suite K
Palo Alto, CA 94306
(415) 322-8763

■

THE COMPUTER CORNER

by Bill Kibler

The Holiday season should be over by the time you read this, and I hope all went well. Things here are starting to pick up as the holidays actually get closer. A number of interesting things have been going on.

The Number One CPU

A curious thing about the computer industry is which actual CPU device is the most popular. I have been working as a consultant lately, changing code on a real time machine. As I worked on the code it reminded me that the Z80 is still the most used device around. Now I know that most would think the PC clone based 8088 from Intel would be tops, but there are too many variations of that device for anyone of them to be used extensively. The other factor is cost, the Z80 is cheap.

The machine I worked on controls a camera mechanism and provides some user interface. There is about $50 worth of components including the card. Many of the newer CPUs, especially 16 and 32 bit versions, can cost 4 or 5 times that amount, just for the CPU. On top of that I love to do Z80 assembly. It is simple and straight forward, only bettered by the 68000. For real simple applications, the 68000 is overkill and so another reason to use the Z80.

This leads me to comment on a letter I received:

Dear Mr. Kibler

I read about your interest in Forth engines in the "The Computer Corner" of *The Computer Journal* issue #35.

I do a fair amount of assembly language programming and am getting interested in Forth. My concept is similar to yours in that I think Forth would make a nice platform for a microcontroller SBC. The difference is that I think the Novix is too expensive for simple tasks (e.g. control lights, motors or read transducers). What do you think of a project based on a Z80, 6809, or 68000 with Forth in EPROM.

Heath used to make a computer called a H89. It was a two board system. One for the terminal and the other a Z80 system with 48K RAM. The terminal & CPU board connection used a RS232 line. The lower 8K was for the ROM monitor. The CPU board can be found for $20 at hamfests. I thought this might be a start—replacing the monitor with Forth.

I would be interested in your thoughts and if "The Computer Corner" could be a forum to develop such a home-brew project.

Thanks for listening.
J.O.
Bremen, IN

Well, Mr. O, Thanks for the letter. Forth in memory restricted systems is ideal and far more capable than any debug type monitor. For a new type "SBC" the price of the NOVIX is definitely too high. I have hopes that Harris and their RTX2000 (a Novix plus MMU) will find a high volume user that will bring the price to less that $50 each. There is a product called WISC, a Writable Instruction Set Computer that will have a Forth instruction set. The design and manufacturing cost are to be less than Novix ($50) while only being 10 to 20% slower. I'll have more to say about these later.

For cheap engines like you suggested in your letter, Rockwell has a 6502 with internal Forth ROM, called the R65F11. I have one of the units and have found some ways around the design bugs. Unfortunately Rockwell is not actively supporting the device so help from them is almost nonexistent. Newer Forths include the M68HC11 from New Micros Inc. based on the Motorola device. There is also an Intel 8031 Forth from Bryte Computers Inc. I haven't played with either of these two yet, but understand they have real possibilities.

For beginners to Forth like yourself, buying a Forth engine might be too much too soon. I have used Xerox 820 boards (cheap like your H89 boards) under an Idaho Forth ROM. The code for the ROM is available from Micro Cornucopia as their 8 inch disk B18. The information to make changes is not available as the project was based on a commercial product. When I ran the Idaho Forth it proved a bit cumbersome and lacked many features I wanted. I think a better way to go would be using the public domain F83 and modify it for your machine. The reason is the availability of all the source code. You can use it on any CP/M or MSDOS and get familiar with it before you put it in ROM. Several people have done just that and the recent issue of Forth Dimensions (The Forth Interest Groups Newsletter) explains metacompiling (using itself to make a new you) to a standalone application.

As you can see there are a lot of ways to go in putting Forth into or on a computer system. My original use for computers was controlling energy systems and I have decided that Forth is most ideal for applications of that nature. I am very much interested in hearing from you (or anyone doing Forth engine work) as to the problems you encounter and your solutions. The Computer Journal is definitely interested in making not only my Computer Corner available as a forum on using small computers (with or without Forth) but the entire magazine.

Your letter and my recent work on a Z80 controller make me feel that the Z80 version of F83 could stand some commenting on. I was thinking how nice it might have been to make changes using Forth and not a cross assembler. I know you mentioned a lot of other chips, and they are being used under Forth to control real operations. The availability of Z80 based units to modify and play with however makes it a far better option. What is needed is articles, not only mine, on just what is involved in bringing up Forth. What also needs commenting on is what is the real advantage you gain by going to Forth. I plan on doing development in both assembly and then Forth to show the difference.

I might suggest you join both FIG (Forth Interest Group) and GENIE network. The FIG membership will get you Forth Dimensions which has lots of articles and companies selling engines. On GENIE there are 1200 Forth files to be down loaded, as well as round table discussions on all sorts of topics. You might even find me there at times. FIG members have a special subscription price to get started with, not to mention our FIG members

(Continued on page 47)