ISBN # 0748-9331

# THE COMPUTER JOURNAL®

For Those Who Interface, Build, and Apply Micros

Issue Number 21 November—December, 1985 $2.50 US


## Extending Turbo Pascal

**Customize With Procedures and Functions** page 2

## Unsoldering: The Arcane Art

**Second in a Series** page 8

## Analog Data Acquisition and Control

**Connecting Your Computer to the Real World** page 19

## Build the Circuit Designer 1 MPB

**Part 2: Programming the SBC** page 35

## The Computer Corner page 52

**THE COMPUTER JOURNAL**
*190 Sullivan Crossroad*
*Columbia Falls, Montana*
*59912*
*406-257-9119*

***Editor/Publisher***
*Art Carlson*

***Production Assistant***
*Judie Overbeek*

***Circulation***
*Donna Carlson*

***Technical Editor***
*Lance Rose*

***Contributing Editor***
*Ernie Brooner*

***Contributing Editor***
*Neil Bungard*

***Contributing Editor***
*Bill Kibler*

---

*The Computer Journal® is a bimonthly magazine for those who interface, build, and apply microcomputers.*

*The subscription rate is $14 for one year (6 issues), or $24 for two years (12 issues) in the U.S. Foreign rates on request.*

*Entire contents copyright © 1985 by The Computer Journal.*

*Advertising rates available upon request.*

*To indicate a change of address, please send your old label and new address.*

*Postmaster: Send address changes to: The Computer Journal, 190 Sullivan Crossroad, Columbis Falls, Montana, 59912.*

*Address all editorial, advertising and subscription inquiries to: The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912.*

# Editor's Page

**Mid-Level Languages Being Developed**

When the first hobbyists built their microcomputers they had to code their programs directly in machine language, but they soon wrote assemblers which enabled them to use easier to remember mnemonics for their source code. Language development continued and BASIC became the familiar language for the new micro owner. Today there is an almost overwhelming assortment of languages to choose from with something for every purpose, and it seems that there should be no reason to work on language development – but there are those who are not satisfied with what exists, and who feel the need to build on the foundation which has been established. Telling a sofware hacker not to waste his time on language development because everything has already been done is like telling an author not to write another novel because everthing has already been written about, or telling an artist not to bother painting another picture because every kind of picture has already been painted!

While it IS true that there is a suitable language for almost every purpose, it is also true that there are some applications for which there is not an optimum language. As I have stated before, I contend that we should all be comfortable working in at least two or three languages and familiar with two or three more so that we can choose the best language for each application. There isn't (and never will be) a perfect language which is optimum for every use! When we learn our first language we think that we can do everything with it, and although it is cumbersome for some uses we plod along because we don't want to give up our hard won skills and start over again with a new language. But, learning the second language is a lot easier than the first one, and it enables us to see how clunky the first one was in some areas (and also how we miss some of its features in the new one).

If you haven't already read Kibler's Computer Corner turn to the back page and read it now, because Bill's thoughts about languages have been voiced by others several times during the past two months.

Assembly is a great programming language and the small ROMable programs run fast – but coding can be tedious for large programs, and I/O, screen-handling, menus and floating point math routines can be a real bitch to code. Developing a non-trivial assembly language program is awkward because you have to write the souce code with an editor (I use WordStar), assemble to a HEX file, and load that to a .COM file for a trial run. Then back to the editor for any revisions and through the whole routine again. An assembly language program is also VERY difficult to understand and update in the future, even if you wrote it yourself, and can be almost impossible to decipher if written by someone else.

**"There isn't, and never will be, a perfect language which is optimum for every use."**

A program properly written in a structured high level language such as C or Turbo Pascal takes less time to write and is much easier to maintain or modify if your requirements change, but some sections may be too slow for time-critical real world control applications or computer bound portions. Some language implementations also have a large run-time code which results in large files for even small programs. I really like assembly language for the high speed routines and for initiating system port drivers, but I don't like the time and effort involved in non-critical areas such as screen messages and menus.

The experts say that you should fully define the objectives and design your program before writing any code instead of hacking away at it and making

*(Continued on page 44)*

# Extending Turbo Pascal

## Use Procedures and Functions to Develop Your Own Version

by Jerry Houston

Because Turbo Pascal's® procedures and functions allow the use of local variables that don't conflict with other parts of the program, and because parameters can be passed to them, it is really easy to define new keywords, in effect. That means that a version of Turbo Pascal can be developed that meets the individual and personal needs of a particular programmer, whether he or she is a writer of business report programs or of economics simulations or engineering number-crunchers.

For example, when printing business forms like an invoice, I wanted a simple command that would establish a print position on the printer, such as the standard Turbo Pascal procedure GOTOXY(h,v) which places the cursor at a specified horizontal/vertical position on the CRT screen. A simple procedure can be written to do this, effectively adding another command to the language. Anyone can extend the language with keywords that are familiar from another language, if they want. Thus, a Turbo Pascal instruction could be written such as:

```
HTAB(20); VTAB(10); WRITE('Hi, there!");
```

which would do exactly what an old Applesoft® programmer would expect it to, except in this case, with the printer!

So that the runtime library isn't overly large, little-used commands are not built into Turbo Pascal, but left up to the user. Exponentiation isn't accomodated directly by any version of Pascal or C that I know of, but can be added easily, using the LOG function, which is available. Thus, the MBASIC® statement:

```
100 C = A ^ B
```

could be implemented in Turbo Pascal such as:

```
C := POWER(A,B)
```

once the function has been written and tested. Significantly, it never has to be written or tested again, and can be used whenever needed.

Because of the block-read and block-write features of the Turbo editor (it works just like WordStar's® CTRL-K R and CTRL-K W), it's easy to save procedures and functions separately on a disk for future use in other programs. The {$I} (include) compiler directive will even merge a procedure with a source as it is being compiled, making it unnecessary even to read it in with the editor. A whole personal library of often-needed procedures and functions can be read into the source code with just a few keystrokes, and seldom-used ones can be added individually at compile-time.

The proper construction of procedures and functions, and the proper use of local variables with them, is an important topic for anyone who's starting out with Turbo Pascal, because it helps them to exploit the full power and versatility of this beautiful language. Procedures and functions in Pascal share many qualities—for convenience, I'll refer to both of them as "procedures" from now on, until the time comes to make a distinction.

**Pascal vs. Basic**

For those who are just now being introduced to Pascal (due to the amazing popularity of Turbo), I should pause to explain two of the major differences between Pascal and BASIC, a language familiar to nearly everyone. The first of these differences lies in how variables are used, the second in how programs themselves are structured (or not).

Disorganized program design is easy for anyone, using languages that do not lend themselves to good structure. BASIC is a very unfortunate choice for a beginner for this reason, but it's the one to which most of us were first introduced. Not only does BASIC—when used without the benefit of training in structured design—encourage pointless jumping from one place to another within the program, but it allows new variable names to be "invented" at any time. Misspell a variable somewhere in that maze of instructions, and BASIC will go along with it without question, even though it means disaster for the program. BASIC has singlehandedly brought a whole new meaning to the term "debugging".

Pascal, on the other hand, requires that variables be defined before they're used, and that they be declared according to their intended use. There are many reasons for this requirement—all of them excellent. For example, if a string variable is used in BASIC, it's stored in memory the first time it is given a value. The next time a value is assigned to the same string variable, it can't be stored at the same place, as its length as well as its contents might change. So a different chunk of memory is used, and a table is updated to point to the NEW contents of that string variable.

In a program that sorts a large list of string variables, it's likely that a particular variable like COMPARE$ might get thousands of different values stored in it before the sort is finished. Each time this happens, more memory is used, until BASIC finally runs out of variable storage space and has to stop to GARBAGE COLLECT, the usual term for reclaiming the space used by all the PREVIOUS CONTENTS of a string. This process can actually stop the execution of a program for minutes at a time, depending on the computer.

Pascal, on the other hand, assigns ONE AND ONLY ONE area of memory to store the contents of any variable, including a string. It does this by requiring that the user declare not only the NAME of a string, but also its maximum LENGTH. Since the same storage location is re-used each time a new value is assigned, there's no such thing as GARBAGE COLLECTION in Pascal.

Since all variables are declared before they're used, it is a simple matter for the Pascal compiler to call errors to the programmer's attention. If a variable name is misspelled part way

thorugh a program, the compiler will, in effect, say "What's this?" when it sees the error.

Turbo Pascal sets aside the amounts of memory shown in Figure 1 for the particular types of variables, and that same space is ALWAYS used to contain the current contents of that variable.

Using the keyword ABSOLUTE, a variable can be assigned a memory location of the programmer's choice, or Turbo can be allowed to keep them in a data storage section.

Like variables, procedures and functions must be declared before they are used. When the compiler reads through the list of source code to convert it to machine language, each time it sees a reference to a procedure or a function, it must already know what that operation requires. Thus, Pascal programs are said to be designed from the top down, and written from the bottom up. In the design stage, the overall job is split into tasks that need to be done. Each task is split further, into smaller tasks, until finally procedures can be written easily to accomplish those tasks. Writing the program is simple then—just write the code needed to execute each of the tasks, then write the MAIN LOGIC that asks for them to be done as needed.

Some excellent programmers take a different approach, preferring to design the program by writing the main logic itself first. Using this method, the main logic is written referring to procedures that haven't yet been coded, with no thought given to HOW each procedure will be written. Once the overall logic is complete, the programmer goes back and writes the procedures in detail.

Because it's so important to writing PORTABLE sections of code—ones that can be used in program after program—the concept of GLOBAL and LOCAL variables needs to be understood here.

GLOBAL variables are defined at the beginning of the entire program, and they exist to all parts of the program—the main logic and all the procedures. This is the familiar approach used in languages such as BASIC. LOCAL variables are defined at the beginning of each of the individual procedures instead, and have meaning only to that procedure and to other procedures that it calls. This concept is called SCOPE.

That means that I can define a variable called COUNT in several procedures, and use it as a counter in each of them, and never have to worry about whether one part of the program will change the value of a variable used in another part! Each version of COUNT will be stored separately, and will exist ONLY to the procedure in which it is declared, an example of local scope. If, however, I declare a variable called COUNT at the beginning of the program, then it's fair game for each procedure in the program. Any of them can change its value, and only ONE version of it exists in memory. It's easy to see that there are good uses for local variables, just as there are good reasons to use global variables, depending on the logic of the program.

| TYPE | USE | STORAGE |
|---|---|---|
| Boolean | Takes on value of True or False only. Often used as a "flag" to indicate whether a condition exists. | 1 Byte |
| Byte | Integer-type variable, can contain whole numbers from 0 to +255. Most efficient for counters, where the number of iterations won't exceed this range. | 1 Byte |
| Integer | Can contain whole numbers from -32768 to +32767. Faster numeric operations than Real Numbers, and require only 1/3 their storage space. Ideal choice for numerics that won't take on any fractional values. | 2 Bytes |
| Real | Often called Floating-Point variables, as they can express a numeric value with a decimal point in it. Can store values from 1E-38 to 1E+38. Turbo Reals contain up to 11 significant digits. | 6 Bytes |
| Char | Short version of a string variable. Can take on any value of a character in the ASCII character set, that is with an ordinal value from 0 to 255. | 1 Byte |
| String | As usually thought of in other languages, like BASIC. Strings can contain up to 255 characters. They are stored in an area of memory that contains one byte for the length, then one byte for each character the string is defined to hold. | 1 Byte + Length of String |

Figure 1

**Procedures**

All implementations of Pascal, including Turbo Pascal, depend heavily upon procedures, which are sections of code that are written to accomplish a particular task. One of the greatest strengths of Pascal as a language is that it encourages the thoughtful design of a program through top-down structuring techniques, as mentioned above. Another strength is that procedures, once properly written, can be used again and again in programs that require the same simple task.

Since procedures are called when needed just by mentioning their name, the language itself can be extended in any way the programmer desires. Consider the case of GOTOXY(). It's a function that's provided with Pascal, one the user doesn't have to write, and is part of the Pascal Library (runtime package) that becomes a part of each program when it's compiled. GOTOXY() positions the cursor on the screen, so that a non-scrolling display can be written. The horizontal position is the first parameter in the parentheses following the procedure name (GOTOXY), and the vertical position is the second parameter.

Consider a program that needs to ask a large number of questions, and get a lot of answers from the operator. Instead of displaying questions one at a time and prompting individually for the answers, the programmer might prefer to display all the questions on the screen at one time, then move the cursor to the position following each question and ask for the answers, one at a time. To put the cursor in the tenth horizontal position of the fifth screen line, the procedure would be called by code such as:

```
GOTOXY(10,5); Read(Answer);
```

Consider now that an invoice is to be written, and you need the PRINT

HEAD, not the cursor, to go to a particular position on the paper before printing. The answer is to write a suitable procedure, then save it on a disk for future use whenever you need the same feature. The following example (which works with the Kaypro/Juki printer) can be modified easily for printers that use other control codes for print head movement:

```
Procedure PrintAt(X,Y:Byte);
Begin
  Write(lst,#27#9,chr(X));
  Write(lst,#27#11,chr(Y));
End;
```

The first line identifies the code as a procedure, and names it PrintAt. The parameter list following the name shows that two parameters will be passed to this procedure, and locally calls them X and Y. Each of these parameters is identified as a Byte, so they can take on values from 0 to 255, quite appropriate for the purpose. The BEGIN and the END show the compiler what code is actually part of PrintAt.

The procedure PrintAt() calls another built-in procedure, called WRITE. The first parameter for WRITE is "lst", which tells Pascal that this output is to be sent to the LST: device, the printer. (When a WRITE procedure is used to send data to the CRT screen—the most common case—the first parameter is unnecessary, as the default is CON:). The printer is sent the values which represent ESC CTRL-I for a horizontal tab, and the character which represents the position wanted, CHR(X). Then it's sent the values which represent ESC CTRL-K, for a vertical tab, and the character which represents the vertical position wanted, CHR(Y).

Suppose now that your invoice form needs the variables Name and Address printed at column 20 on lines 5 and 6, and the variables City, State, and Zip printed starting at column 20 on line 7, with a comma between City and State, and two spaces between State and Zip. The code in the main logic to accomplish that would simply be:

```
PrintAt(20,5); Write(lst,Name);
PrintAt(20,6); Write(lst,Address);
PrintAt(20,7); Write(lst,City,', ',State,'  ',Zip);
```

Like the built-in procedures Read, Write, and GotoXY, you now have a brand-new—and incidentally, very useful—procedure called PrintAt at your disposal. All that's needed is to include it into each source code as you write your programs.

Such pieces of source code can be included automatically, thanks to the compiler directive {$I}, the INCLUDE directive for the Turbo Pascal compiler. To include the procedure PrintAt with another later program (assuming that PrintAt has been saved to disk), only the instruction:

```
{$I PrintAt}
```

is needed in the source, before PrintAt is referenced by name. The compiler will stop reading from the source code file that it's working with, and read in PrintAt from the disk, then continue with what it was doing.

Alternately, the Turbo editor can be used like WordStar to include such sections of code with the source. Place the cursor where you'd like the procedure to go, then type

CTRL-K R

The editor will respond by asking what file name to read from, and you'll type

PrintAt

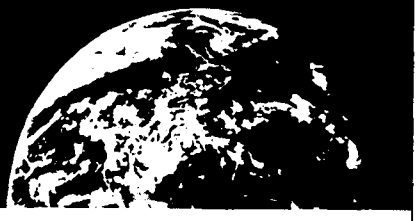
An extension of .PAS is assumed when using the editor to read or write files, and will be added automatically unless you add a dot to indicate no extension, or add a dot and a different extension. Then your file name and extension will be taken literally.

Upon finding out the name of the file you want to include, the Turbo editor will read it in, and highlight the text if your terminal is capable of that. (To get rid of the highlighting, CTRL-K H works fine, again just like WordStar.)

This technique can be used to include a whole library of needed procedures, and the user is free to develop several libraries for special purposes. When additional procedures are stored together in a single file, as a library, less disk storage overhead is required, and the language can be extended considerably with only a few keystrokes.

It's probably obvious by now, but the reverse of this routine is useful to save a procedure to disk. If you've just written a terrific procedure that will simplify all your future programming, take the time to mark the top of the block

GIVE YOUR COMPUTER THE ABILITY TO INTERACT WITH THE REAL WORLD



MONITOR AND CONTROL TEMPERATURES

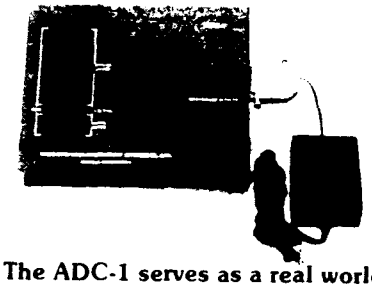
MANAGE INDUSTRIAL PROCESSES

MEASURE ENERGY CONSUMPTION

CONTROL LAMPS AND APPLIANCES

PROVIDE SECURITY PROTECTION

PERFORM SCIENTIFIC DATA COLLECTION



**The ADC-1 serves as a real world interface for any computer or modem with a RS-232 serial port.**

This sophisticated yet easy-to-operate data acquisition and control system includes:

- 16 Analog to Digital Inputs - 12 bits provide 0.1mV resolution over ± 0.4V.
- 4 Digital Inputs for security and rotary encoder sensors.
- 6 Switched Outputs for relays and low voltage device control.
- AC Line Carrier Transmitter - controls 32 BSR X-10 type remote modules.
- Owner's Manual with detailed programming examples.

**Sensors available from Remote Measurement Systems** include: light, temperature, humidity, wind, sound, soil moisture, ultrasonic ranging, energy consumption and security.

**The ADC-1 – an exceptional purchase at $449.**

**REMOTE MEASUREMENT SYSTEMS, INC.**
2633 Eastlake Ave. E., Suite 206
Seattle, Washington 98102
(206) 328-2255
Send for complete specifications
Telephone, Visa and Mastercard orders welcome.

with a CTRL-K B and the bottom with a CTRL-K K, then type CTRL-K W and give the editor a file name when it asks. Just like you-know-what.

**Functions**

All the manipulation techniques described so far apply equally to procedures and functions, but the two are somewhat different in use. A procedure is like a tiny program that accomplishes a single useful task, whereas a function is like a variable that takes on a particular value according to specified operations.

Since all the built-in procedures and functions of Turbo Pascal are included as part of the runtime library that becomes a part of each compiled program, routines that are not often used are not built-in for reasons of space. These built-in features take up nearly 10 K of memory, meaning that the very smallest of programs, when compiled, will be at least 10 K long. It's amazing what can be accomplished with that 10 K of library, but that's not the point.

Everyone will sooner or later miss a favorite keyword from another language. It's inevitable, so let's expand on an example mentioned briefly before. BASIC offers easy exponentiation using the ^ operator. To find the cube of the variable A, for example, one can express it as A^3. Such features are not part of Pascal, because the code required to execute them would become part of every compiled program, including the thousands of programs that never have to perform such tasks.

It's at times like this that the extensibility of Pascal really comes through. Here's a function called POWER that will provide the same feature in Pascal.

```
Function Power(X,Y:Integer) : Integer;
  Var R : Real;
  Begin
    R := Exp(Ln(X), Y);
    Power := Round(R);
  End;
```

The first line names the function, just as we named a procedure in the PrintAt example. In the parentheses are defined variables representing the two values that will be passed to this function (the parameters). Variables X and Y are to be integers, as is the function 'Power', itself.

The local variable R (for result) is assigned as a Real. R is arrived at using the EXP (exponent) and LN (natural logarithm) functions, which are built-in. Then R is rounded off and stored in the function Power. This function was written for the purpose of setting and clearing bits within a byte, so it was designed as an integer function. It could easily be rewritten as a real function, for floating-point math.

Again, the BASIC code

```
A = B^C
```

can be duplicated in Pascal as

```
A := Power(B,C)
```

and once more the language has been extended with the addition of a new keyword.

**Portability**

In this case, portability refers to the quality that lets a function or procedure be used with any program, not just the one for which it was originally written. To be truly portable, procedures and functions should be written using parameters when needed, and using local variables. These local variables should not be used as global variables within any of the programs. An easy way to accomplish that is to use single-character variable names for local variables, and variable names of more than one character for global variables. This isn't a rule of Turbo Pascal, but one I find useful for myself.

Note for those who are REALLY new to Turbo: All identifiers, such as variable names, procedure names, and function names, can be up to 127 characters long, with each character significant, and case is ignored. That means that a variable that refers to net profit before taxes can be called:

NetProfitBeforeTaxes

if you really want to, and a procedure that calculates a standard deviation can be called:

CalculateStandardDeviation

The compiler tokenizes all these names, so being descriptive doesn't cost anything at all, except maybe a little extra typing. Well written, a Pascal program can be darn near self-documenting, and a joy to modify at a later date, if that ever becomes necessary.

Needless to say, since Pascal programs are generally made up of many procedures, some of those will certainly NOT be of use in other programs. Using the concept of top-down design, a single large task is divided into two or more intermediate tasks, which are each further broken down into two or more smaller tasks.

```
PROGRAM          ;

{                                                    }

CONST
          ReverseOn        :   String[3]  =  #27'B0'  ;
          ReverseOff       :   String[3]  =  #27'C0'  ;
          HalfIntOn        :   String[3]  =  #27'B1'  ;
          HalfIntOff       :   String[3]  =  #27'C1'  ;
          FlashOn          :   String[3]  =  #27'B2'  ;
          FlashOff         :   String[3]  =  #27'C2'  ;
          UnderlineOn      :   String[3]  =  #27'B3'  ;
          UnderlineOff     :   String[3]  =  #27'C3'  ;
          CursorOn         :   String[3]  =  #27'B4'  ;
          CursorOff        :   String[3]  =  #27'C4'  ;
          VideoOn          :   String[3]  =  #27'B5'  ;
          VideoOff         :   String[3]  =  #27'C5'  ;
          MemorizeCursor   :   String[3]  =  #27'B6'  ;
          ReturnCursor     :   String[3]  =  #27'C6'  ;
          StatusLineOn     :   String[3]  =  #27'B7'  ;
          StatusLineOff    :   String[3]  =  #27'C7'  ;
          DeleteLine       :   Char       =  ^R       ;
          DeleteToEnd      :   Char       =  ^W       ;
          Bell             :   Char       =  ^G       ;
          ClearScreen      :   Char       =  ^Z       ;
          Home             :   Char       =  ^^       ;

     SerialPort  = 4 ;          { assigns serial port number }
     StatusPort  = 6 ;          { assigns status port number }
     Mask        = 1 ;          { assigns status mast value }
     BaudPort    = 0 ;          { port number to set baud rate }
     BaudRate    = 14 ;         { this value sets baud = 9600 }
```

Listing 1

Soon these bottom-level tasks are small enough, and simple enough, so that they're easily coded into procedures. It is these bottom-level tasks that are usually worth saving for future use. It makes good sense to set aside a procedure like PrintAt, since it would almost certainly be useful in many kinds of programs. A procedure that gathers answers to a series of prompts used in one particular program, though, wouldn't be very useful elsewhere.

**Examples**

Some examples of procedures and functions that I've found useful in many programs are shown in Figure 2. In each case, I've identified the purpose, but I won't necessarily point out all the details of Turbo Pascal syntax, as this part of the article is intended for programmers who have and use Turbo Pascal.

It should be obvious that certain escape sequences, especially, are system-dependent, and that they will need to be modified a little (or maybe a lot) to work with a different terminal, printer, or other peripheral. Where this is not obvious, I'll point it out. In these particular examples, specific escape sequences are appropriate for the computer I use most often, a Kaypro 4-84® and the Kaypro (Juki 6100) letter-quality printer.

Those who happen to use the same equipment should be able to copy these routines directly. Those with other gear should not find it difficult to substitute small portions of code that will work for them.

Many readers will find these examples immediately useful, but that's not the real reason for sharing them. These have been short, easy to understand procedures and functions, but the same principal can be applied to complex—but routine—procedures. The IMPORTANT concept is that nearly every command you've ever wanted in a language can be developed as an extension to Turbo Pascal, making it one of the most powerful languages that was ever this easy to use.

I want to leave one last recommendation that will save a lot of time and trouble, too, although it doesn't have a thing to do with procedures or functions. It DOES provide another way to define more keywords.

Many programs can make use of the terminal capabilities of a computer, such as commands that clear the screen, home the cursor, print reverse images, highlight, or even turn off the video display when a key is not pressed for a long time. In some cases, these terminal attributes make use of some of the ASCII characters above the normal 0-127 range used for printing and ordinary control characters. In other cases, such as with the Kaypro, these are gained by printing an ESCape

```
Procedure HTAB(X:Byte);
Begin
   Write(lst,#27,#11,chr(X));
End;

Performs a printer horizontal tab to the position given in X.


Procedure VTAB(X:Byte);
Begin
   Write(lst,#27,#9,chr(X));
End;

Performs a printer vertical tab to the line given in X.


Function ACD(X:Byte) : Byte;
Begin
   ACD :=$5 ((X Div 16) * 10) + (X - (16 * (X Div 16)));
End;

Converts BCD (binary-coded decimal) values to ASCII decimal
values. Used in the procedures below that read from a
clock/calendar card, as the usual output from these clocks is in
BCD.


Procedure GetDate(Y,M,D:Byte);
Begin
   Port[34] := 15;            (make up Kaypro clock card)
   Port[32] := 9;             (ask for year)
   Y := ACD(Port[36]);        (get year)
   Port[32] := 7;             (ask for month)
   M := ACD(Port[36]);        (get month)
   Port[32] := 6;             (ask for day)
   D := ACD(Port[36]);        (get day)
End;

Gets the date from the Kaypro 4-84 clock card. In a program,
code such as the following might be used to write a formatted
date:

         GetDate(Year,Month,Day);
         Write(Month,'/',Day,'/',Year);


Procedure GetTime(H,M,S:Byte);
Begin
   Port[34] := 15;            (make up Kaypro clock card)
   Port[32] := 4;             (ask for hour)
   Y := ACD(Port[36]);        (get hour)
   Port[32] := 3;             (ask for minute)
   M := ACD(Port[36]);        (get minute)
   Port[32] := 2;             (ask for second)
   D := ACD(Port[36]);        (get second)
End;

Gets the time from the Kaypro 4-84 clock card. In a program,
code such as the following might be used to write a formatted
time:

         GetTime(Hour,Minute,Second);
         Write(Hour,':',Minute,':',Second);


Procedure DrawBox(V1,V2,W1,W2 : Byte);
Begin
   Write(#27+'L'+chr(28+v1*4)+chr(31+w1*2)+chr(28+v1*4)+chr(31+w2*2));
   Write(#27+'L'+chr(28+v1*4)+chr(31+w2*2)+chr(32+v2*4)+chr(31+w2*2));
   Write(#27+'L'+chr(32+v2*4)+chr(31+w2*2)+chr(32+v2*4)+chr(31+w1*2));
   Write(#27+'L'+chr(32+v2*4)+chr(31+w1*2)+chr(28+v1*4)+chr(31+w1*2));
End;

This is not code you'll want to type often! For drawing outlines
on the screen using the Kaypro graphics, this draws the four
lines needed. The four parameters are the beginning and ending
vertical positions (lines) on the screen, and the beginning and
ending horizontal positions (columns). This procedure is written
to use values pertaining to cursor positions, and to translate
those into the graphics positions required by the line-drawing
escape sequences. To draw a box around the entire screen, one
would use:

         DrawBox(1,24,1,80)
```

Figure 2

sequence that the terminal recognizes as a special attribute request.

As easy as reading a procedure from a disk file, a text file can be read that contains constant assignments for all these attributes, with names that are easily remembered and used in one program after another. This file is, actually, the beginning of a Pascal program, with all these constants assigned. If space is at a premium (hardly ever the case, with Turbo's virtual memory capability), any that are not used for a program can be deleted when the programming is finished. It usually doesn't hurt anything to leave them, though, and they'll be available if you decide later to "spruce up" the output.

During the actual coding of the program, these attributes can be called upon by "Writing" them. If a particular action is required, like turning off the cursor during graphics routines—to make a cleaner display—it can be handled separately, as in:

```
Write(CursorOff);
```

If an attribute has something to do with a value that's going to be written anyway, it can simply preceed it or follow it, like any other variable, such as:

```
Write(FlashOn,'This message flashes...',FlashOff);
```

I use a file called STARTUP.PAS to begin nearly every program I write. A version of it is shown in Listing 1. Again, this file will be of special interest to Kaypro '84-or-later owners, but owners of other equipment should appreciate the IDEA.

On work disks that I use for a special purpose, such as analog/digital data acquisition programs, I keep a special version of STARTUP.PAS that contains, in addition, functions and procedures for operating an ADC-1 Data Acquisition and Control System. On disks I use for business programs, I keep a copy of STARTUP.PAS that includes procedures and functions for getting dates and times from the clock card, printing datelines, and so on. ■

*The following are Registered Trademarks: Turbo Pascal, Borland International; MBASIC, Microsoft; Applesoft, Apple Computer Company; WordStar, MicroPro International.*


*CP/M-80 C Programmers . . .*

# Save time

. . . with the BDS C Compiler. Compile, link and execute *faster* than you ever thought possible!

If you're a C language programmer whose patience is wearing thin, who wants to spend your valuable time *programming* instead of twiddling your thumbs waiting for slow compilers, who just wants to work *fast*, then it's time you programmed with the BDS C Compiler.

BDS C is designed for CP/M-80 and provides users with quick, clean software development with emphasis on systems programming. BDS C features include:

- Ultra-fast compilation, linkage and execution that produce directly executable 8080/Z80 CP/M command files.
- A comprehensive debugger that traces program execution and interactively displays both local and external variables by name and proper type.
- Dynamic overlays that allow for run-time segmentation of programs too large to fit into memory.
- A 120-function library written in both C and assembly language with full source code.

Plus . . .

- A thorough, easy-to-read, 181-page user's manual complete with tutorials, hints, error messages and an easy-to-use index — it's the perfect manual for the beginner *and* the seasoned professional.
- An attractive selection of sample programs, including MODEM-compatible telecommunications, CP/M system utilities, games and more.
- A nationwide BDS C User's Group ($10 membership fee — application included with package) that offers a newsletter, BDS C updates and access to public domain C utilities.

Reviewers everywhere have praised BDS C for its elegant operation and optimal use of CP/M resources. Above all, BDS C has been hailed for it's remarkable *speed.*

"I recommend both the language and the implementation by BDS very highly."
Tim Pugh, Jr.
in *Infoworld*

"Performance: *Excellent.*
Documentation: *Excellent.*
Ease of Use: *Excellent.*"
*InfoWorld*
Software Report Card

". . . a superior buy . . ."
Van Court Hare
in *Lifelines/The Software Magazine*

*BYTE* Magazine placed BDS C ahead of all other 8080/Z80 C compilers tested for fastest object-code execution with all available speed-up options in use. In addition, BDS C's speed of compilation was almost *twice* as fast as its closet competitor (benchmark for this test was the Sieve of Eratosthenes).

***Don't waste another minute on a slow language processor. Order your BDS C Compiler today!***

Complete Package (two 8" SSDD disks, 181-page manual): **$150**
Free shipping on prepaid orders inside USA.
VISA/MC, COD's, rush orders accepted.
Call for information on other disk formats.

BDS Software, Inc.

BDS C is designed for use with CP/M-80 operating systems, version 2.2, or higher. It is not currently available for CP/M-86 or MS-DOS.

BD Software, Inc.
P.O. Box 2368
Cambridge, MA 02238
(617) 576-3828

# Unsoldering: The Arcane Art

## Second in a Series on Soldering, Unsoldering, and PC Board Repair

by James O'Connor

Why do you think they put erasers on the end of pencils? Who hasn't heard that old chestnut, and the answer is because people make misteaks, mistakes, eras, errors, whatever. The spelling checker program will blow a binary fuse when it tries to digest that sentence.

Just as things go wrong when using a pencil, they also go awry when soldering. From the moment you begin to solder things together, inevitably you'll solder the wrong thing in the right place, or the right thing in the wrong place. And if you get really involved in electronics you'll want to unsolder components because they are under suspicion of malfunctioning.

Having determined that you must unsolder something you might naturally consult your reference material for some helpful hints, and if your referenece library is similiar to mine you may find a glaring lack of any help. When I recently reorganized all my literature I could find practically nothing on unsoldering among the dozens of features on soldering. Had I stumbled upon some dark and mysterious art known only to a select few? Some ancient ritual marked by obscure incantations, oaths, and secret hand clasps? Yes! At least the part about odd hand clasps, more about that later, and as to the mumbled oaths almost any will do if they benefit the process.

But seriously, the true purpose of this article is to delve into the intricacies and problems of unsoldering as fully as space will allow. In the initial article in this series on soldering it was stated that soldering is mostly science and partly art. By science is meant hard knowledge about the process and what makes it work, by art is meant the experienced judgement that comes from actual practice. Unsoldering on the other hand, hands that is (more than two are very useful), is more like 50/50 science and art. Or to phrase it another way, knowing what to do is equally as important as knowing when to do it and when not to.

We begin by reviewing the theory of soldering as it applies to unsoldering, then we ramble into the jungle of unsoldering tools including the techniques of using them. Next we'll probe the art part, applying the right tool and techniques. Followed by the main skirmish, unsoldering a DIP IC which leads us to all the things that can go wrong trying to unsolder a DIP. We finish by discussing construction methods and how they affect the whole problem of unsoldering.

**Theory**

In the first article we began by talking about welding – separating things that have been welded together is very straightforward, since welding makes one object out of two the only way to reverse the process is to simply cut the one object into two pieces. We talked about welding to emphasize that soldering is different, we proposed the analogy that soldering is more like gluing things together, that solder can be thought of as a heat activated metallic 'glue'. Glues stick things together so if the glue can be unstuck, the parts can be separated.

Glues come in several different types and the properties of each type affects how to unstick them. Some glues dry very brittle and can be separated by fracturing them, the old hide glue used on early furniture behaved this way, it would fracture with age which is why old furniture often has wobbly joints. Some glues never really harden, they just stay sticky, these can be defeated either by using a solvent to disolve them or in some cases by heating the glue until it releases, the glue used on floor tiles is like this, heat it up and you can pry up the tile. Modern white glue that is used for wood and paper is made from chemicals that are very similar to the actual binders in natural wood, and these blend right into the surface structure. If they are waterproof then it may be virtually impossible to unstick things made with these glues, imagine the fix our modern world would be in if the glue used to make plywood was not as strong as it actually is.

Solder is very much like the modern glue that blends into the surface of a material, indeed it could be said that solder was the first example of this type of 'glue'. Fortunately though solder can be affected by the very thing that makes it work in the first place, heat! Obviously then the first step in unsoldering will involve reheating the joint. It might seem to be the only thing required, just heat up the joint and separate the two parts. In some instances this will work, but most of the time there are some real difficulties with this alone. For instance since heat is involved you can't use your fingers to reach in and pull things apart, not without suffering severe burns to your fingertips, a very painful burn, indeed. Some sort of forceps or pliers must be employed, which can be awkward. Because using tools is so tricky, you may be tempted to heat the joint longer than necessary to be sure that the solder is liquefied, that can lead to physical damage to adjacent parts, especially the cicuit traces on the board. As we won't be discussing ways to repair damaged traces until the next article in this series you'll want to avoid this problem for now.

Solder is interesting because it has properties similiar to the three different glues mentioned above. If heating a joint and pulling things apart is not always practical then the next step would be to remove the solder. Since solder invades the surface of the metals it is not possible to remove it all but if only the surface coating remains then the resulting joint will be extremely weak and should fracture like the hide glue with gentle manipulation.

Clearly we can't simply swab up molten solder with a Q-tip or even the 'quicker picker upper', a paper towel, hot solder would burn these materials and being organic they would not readily absorb a hot metal. We could use some sort of wick to absorb the solder as long as the wick is compatible with solder and later we will talk about such products. Another approach would

be to vacuum it up with a wet/dry vacuum cleaner, which would indeed work but is actually a bit too elaborate. Instead there is a whole array of tools ranging from the super simple to utterly elaborate that use vacuum to suck up the liquefied solder.

**Tools and Techniques**

The first tool we'll discuss is the soldering iron used to reheat the joint. No mystery here, simply select the same size iron and style of tip as you would have used to make the connection (see the first article in this series for details). To use the iron, first heat it up and properly tin it, then apply it to the joint so that it touches the lead and sits on top of the solder fillet. This causes the heat to flow through the lead into the hole and on to any solder on the other side of the board. This is important when working on double sided boards, and works just as well on single sided ones. The solder fillet will tell you when it's molten because the tip will slip down into it. Keep the iron in place for one second more and the joint is all set for the next step. Occasionally, it won't be possible to position the iron as described, in that case apply the iron as best you can for no more than about one second longer than would be needed to make the connection. Be very careful not to overheat the connection, if it doesn't seem to be liquefying, back off and evaluate the situation before trying again.

Do you recall those vague allusions to 'strange hand grasps' at the start of this article? The time has arrived to reveal all as we delve into the arcana world of the Solder Suckers. No, these are not a lonely band of misguided consumers who bought the wrong type of solder, rather they are tools. There are three distinct classes, with numerous subspecies, but they all share the common characteristic of providing a small, rapid vacuum that pulls the molten solder out of the joint.

In true zoological fashion lets begin by examining the ancestral Solder Sucker which is by no means extinct. You can find this primitve at your local Radio Shack (Cat. No. 64-2086). All this tool consists of is a rubber bulb attached to a plastic tube that ends in a pointed hollow tip about the size of a circuit pad. Here's how to use it; if you're right-handed hold the bulb in your left hand and squeeze it closed (lefties use your right hand), you'll hear air being expelled out the tip, keep it closed. With the soldering iron in your other hand, apply it as described above. Once the solder is liquid, in one quick, smooth motion remove the iron and with your left hand place the tip of the solder sucker over the joint and immediately release the bulb. Let it spring open quickly, this will create a vacuum at the tip and suck up the molten solder.

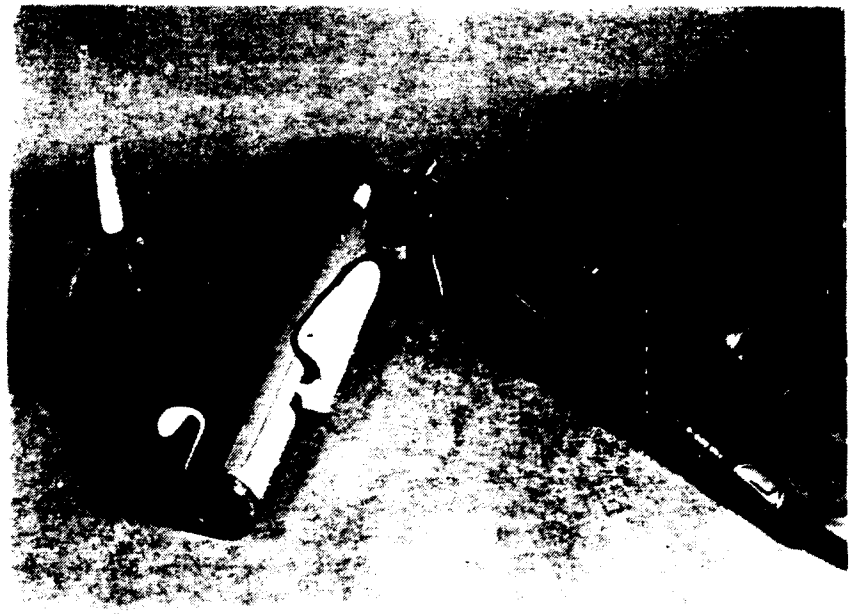


Figure 1: Solder Suckers, from left to right, a simple type, a spring type, acombination of iron and built in bulb.

Sounds simple, but the process is actually easier to describe than to accomplish on the first attempt. Two ways to overcome this; first find someone who knows how to do this and have them teach you, just watching is very helpful, or find a scrap printed circuit board and practice the technique several times until you master it.

The bulb type solder sucker is a very simple tool and has few problems. Occasionally it will fill up with bits of solidified solder and must be cleaned. To do this just hold it over a trash can and pump the bulb to blow the pieces of solder out, if they are too large to fall out then twist off the tip and shake them out of the bulb. The tip itself is made of a coated plastic and solder particles will not stick to it but can wedge themselves into the tube, use a thin probe (e.g. a toothpick) to poke them loose.

Proceeding up the evolutionary ladder of solder suckers we encounter the next great advance, the 'spring-loaded sucker'. In the Radio Shack catalog these are more politely termed Desoldering tools (Catalog Nos. 64-2098 and 64-2085.) Early on it was found that the bulb type is tiring if you unsolder many joints and also requires a well developed technique to work consistently. The spring loaded tool overcomes these problems. The tool consists of a tube similiar in shape to a syringe, inside there is a piston that is forced to the top of the tube by a spring. There is a fingerhold by which you compress the piston down against the spring until it catches on a trigger, in some designs the piston has a shaft extending from the top and this shaft is pushed down to prime the tool.

To use the spring tool first prime it by forcing the spring down, the designs vary so you'll need to figure out how to do this, most of the tools have fingerholds that allow this step to be done with one hand, allowing the other hand to hold and manuever the soldering iron. Congratulations, you now know the strange hand grasps!!

Next you proceed as with the bulb type, heat the joint and when the solder is molten, remove the iron, place the desoldering tool over the joint and release the trigger. The spring will snap up creating a short, sharp suction at the tip. The spring creates a better suction than the bulb type is capable of, which makes this tool ideal for working on modern double sided boards.

Maintaining a spring sucker is much the same as the bulb type in that every so often you'll need to clean out the accumulation of solder bits. After doing

several connections there may be so much solder inside the tip that it will jam up the internal piston, to clear this I use a small piece of stiff wire with a tiny L-shaped bend on the end, poked up into the tip the edge of the wire snares the solder and pulls it out. At some point you'll need to disassemble the tool for a complete cleaning. Be sure to follow the directions, it can be a tricky procedure and due to the spring you could end up with bits of solder flying about if you make a misstep. Brush off the internal parts, lubricate only as directed and then reassemble the unit. One problem with this tool is that the tip will often become distorted due to heat from the solder joints, sometimes you can reshape it by careful scraping and cutting, but eventually it will need replacing, and replacements are available especially for the more expensive of the breed.

A final word about the spring type sucker, try to purchase a good quality model, spend a little more than you really want to. It's been my experience that the inexpensive variety tend to self-destruct after a short period and you're faced with buying another one, better to spend a little more to start with and save in the long term.

Now let's move to the king of the hill in solder suckers. These are really industrial models that most part time electronics hackers would not have, but it is useful to know they exist. Basically these are little self powered vacuum cleaners attached to hollow tipped soldering irons. They range in price from a couple hundred dollars up to a thousand or more. They are very effective and often come with a bewildering array of accessories that make them attractive to electronic assemblers faced with many solder joints to be unsoldered.

Before we leave the topic there does exist a sort of 'missing link' in the species. It can be found in the Radio Shack menagerie (Cat. No. 64-2061.) This is a soldering iron with a right angled hollow tip that leads to a metal tube at the end of which is the old familiar solder sucker bulb. The bulb is positioned such that it can be operated by the same hand that holds the iron. The advantage of this tool is that the hollow tipped iron heats the joint and the bulb then sucks up the molten solder without the hand swapping exercise inherent in the bulb or spring type. It resembles the industrial model in its basic principle.

At first appearance this would seem to be the ideal tool but in my experience this tool has some drawbacks. First, the iron has a hefty wattage rating (45 Watts for the Radio Shack model) and can cause damage to circuit traces if you're not careful. Secondly the tip is usually wider than the standard pad size of a circuit board, this forces you to slide the tip until one part makes contact with the protruding lead, unfortunately this often inteferes with complete cleaning of the circuit board hole. And lastly, it can be rather awkward to operate the bulb, you just can't seem to develop as much vacuum as with the simple standalone bulb and nowhere as much as the spring type tool. There is one instance where this tool is helpful, it occurs when attempting to desolder a DIP and will be discussed later under the mechanical aspects of unsoldering.

And for those who may not believe solder suckers evolve, a new member of the family has been sighted in the last year or two. This type combines the


MTBASIC

Multitasking BASIC Compiler

| | Windowing | Multitasking | ROMable code | Recursive | Price | 8087 support | Multi-line functions |
|---|---|---|---|---|---|---|---|
| **MTBASIC** | **Y** | **Y** | **Y** | **Y** | **$49.95** | **optional** | **Y** |
| MBASIC | n | n | n | n | $350.00 | n | n |
| TURBO PASCAL | IBM only | n | n | Y | $69.95 | optional | Y |
| CBASIC2 | n | n | n | n | $600.00 | n | n |
| BASCOM | n | n | n | n | $495.00 | n | n |

**MTBASIC, the multitasking Basic compiler, has everything you need!**

## Interactive Compiler

MTBASIC is an interactive compiler and a unique Basic language. MTBASIC is easy to use since you can write programs in an interactive environment and them compile them using only one command. MTBASIC is easy to learn because it is similar to many other Basics. The biggest advantage of using a compiled Basic is FAST PROGRAMS. With MTBASIC you get speed and advanced features.

## Features

Multitasking
Windowing
Interactive
Compiles in seconds
Multi-line functions
No runtime fee
Handles interrupts
Fast native code
ROMable code
Formatted I/O
Assembly language calls
In-line machine code

## Complete Package

The MTBASIC package includes all the necessary software to run in interpreter or compiler mode, an installation program (so any system can use windows), demonstration programs, and a comprehensive manual.

### Ordering

MTBASIC is available for CP/M, MS-DOS, and PC-DOS systems for $49.95. MTBASIC with 8087 support is available for MS-DOS for $79.95. Shipping is $3.50 ($10.00 overseas). MD residents add 5% sales tax. MC, Visa, checks and COD accepted.

SOFTAID, Inc.

P.O. Box 2412 Columbia, MD 21045-1412 301/792-8096

MTBASIC and BASCOM are trademarks of ...

spring loaded sucker with an iron. It's so new that Radio Shack hasn't captured one yet. OK Industries, Inc. lists one in their catalog (Model SA-6). This looks as though it might work well, it would seem to handle easily, has a small tip opening, and the snappy action of a spring sucker. Only drawback is that these tools cost between $25 and $35 depending on where you spot it, this would make it a tool for the 'serious' unsolderer.

Having covered the major types of solder suckers we move on to the other method of removing solder, wicking. Because we make the solder into a liquid by heating it, we can also wick it up just like a candle wick or a kerosene heater wick. For solder we need something that solder will stick to in order to achieve the wicking effect, this is accomplished by using braided copper that is flattened into a band.

Using solder wick, or desoldering braid as it is sometimes called, (Radio Shack Cat. No. 64-2090) is easy but there are a few tricks that we'll mention. Take the solder wick and unroll a few inches of it, hold it in your left hand, with the iron in your right hand. Place the wick over the joint, and then place the iron onto the back of the wick right above the joint. Now let the iron heat the wick which then heats the joint. When the solder in the joint becomes molten it will be wicked up into the heated braid. When this happens the wick will turn silvery, you can see this happen. Let it stay for a fraction of a second to wick up as much solder as possible and then remove both wick and iron, and prestol you're done—most of the time that is.

The technique of wick desoldering is very easy to learn but there are several technical facts that affect the outcome. First of all, after desoldering a joint, you must then cut off that part of the wick that is filled with solder, use your nippers to do this, solder wick just won't wick after it is filled with solder. Always work with a freshly cut edge. You can buy solder wick in several different widths from some electronics supply companies because the width is important. You want to use a wick that is as wide as the solder fillet you're trying to unsolder. Too narrow a wick won't absorb all the solder, too wide a wick risks damage to traces from the extra heat it holds. The width sold by Radio Shack is a good all around size but if it's too wide for a small joint then trim the end a little.

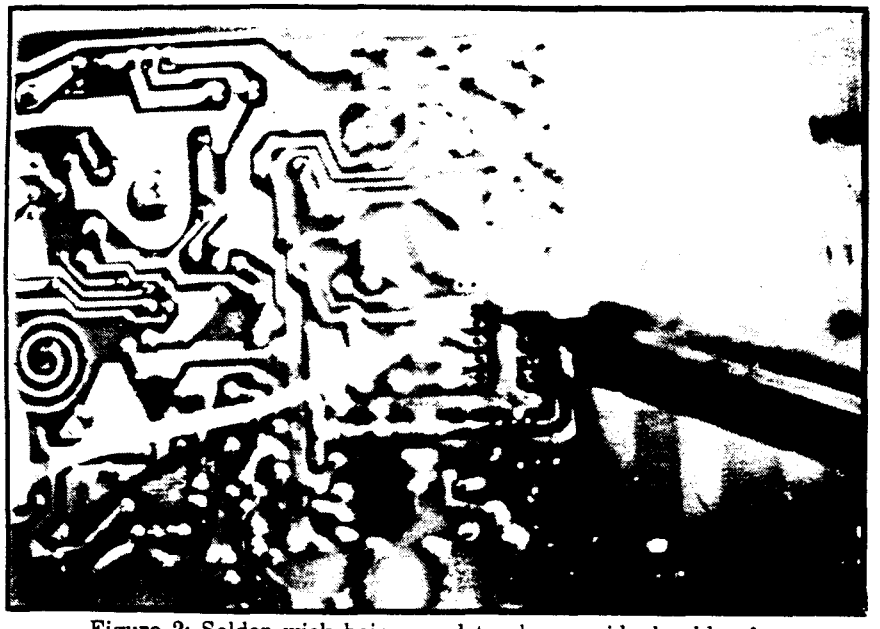


Figure 2: Solder wick being used to clear residual solder from a DIP IC hole.

The wicking action is not as vigorous with solder as with other liquids thus sometimes you may find that solder will remain in the circuit hole or on the opposite side of the board, especially double sided boards. This can also happen when using a solder sucker, although a good spring-loaded type will tend to preclude this. No problem though, just apply the wick on the top side, being careful not to let the side of the iron char components. Many times a stubborn joint will respond to the use of a desoldering tool followed up by solder wick.

Occasionally, you will create a solder bridge between two traces, these rascals are very insidious, causing many problems not the least of which is how to correct them once you identify the culprit. The old method involved using a cleaned iron, which is then placed over the solder bridge causing the solder to creep up the iron by capillary action thus removing the bridge. That sounds much better than it actually works, it requires good technique. Instead use solder wick, just place it over the bridge, heat the wick with the iron and bingo the bridge disappears into the wick. This works so

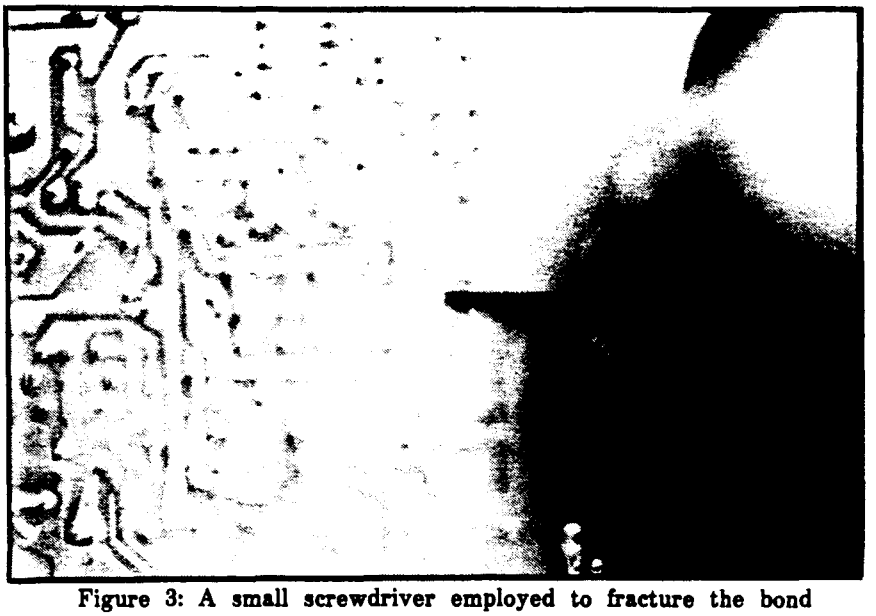


Figure 3: A small screwdriver employed to fracture the bond between DIP lead and plated thru hole by firm inward pressure.

well that sometimes you will have go back and resolder the joint because too much solder went into the wick but it still beats the old soldering iron method. I would be sure to include some solder wick in my tool kit if for no other use than removing solder bridges.

One final note, as in all soldering and unsoldering, never leave a heated solder wick on a connection too long, if the process doesn't work in a few seconds then there is something else wrong and leaving the iron in place longer will only damage the circuit board or the parts you're working on.

**Helping Hands**

One basic rule; if you're trying to unsolder parts from a circuit board that board must be firmly fixed in place, if you're unsoldering a wire from a lug then the lug must be firmly fixed. It is virtually impossible to hold something down while trying to unsolder from it and equally impossible to unsolder moving parts, so you have to employ some type of holder to fix things in place. One tool is Radio Shack's "Helping Hands" (Cat. No. 64-2093), you can also use small vises such as those sold by Radio Shack.

All these bits and pieces can start to add up, especially if you only want to do occasional unsoldering. To solve this, I made my own simple type of "Helping Hands", all I did was cut two pieces of white pine (any other wood will do), one piece is the base and measures 4" wide by 9" long and is ¾" thick, the other is 2 ½" wide, 9" long and ½" thick, all the dimensions are variable, use whatever stock you happen to have. Glue and screw the narrower piece to the wider one along the 9" length to form a right angle. Now take two wooden spring type clothespins and drill one or two holes in one leg of each, use small wood screws and a little glue and attach these to the narrow piece spaced about 4" to 5" apart. This device will hold a circuit board up for inspection and work and you can also leave the board in it and tip it over so that the weight of the holder keeps the board from sliding about. Let your imagination run wild and make similiar holders for whatever size and shape of boards you work with.

**The Art of it All**

Having talked about the unsoldering tools of the trade we now arrive at the larger issue of the art of it all. Knowing when to unsolder, how to unsolder, how to avoid damage, when to deliberately cause damage.

Figure 4: Using a utility knife point to lift up a staked IC pin while heating it with the soldering iron.

It is now time to talk about the special theory of unsolderability. What, you never heard of this theory before? Well that's because I just made it up. No, seriously we can express this as a formula, where U equals the degree of difficulty, P equals the number of pins to be unsoldered, thus we have:

$$U = P^2$$

or to express it in literal terms the more pins or leads a given part has the more difficult it will be to successfully unsolder it, without damaging the part or the board.

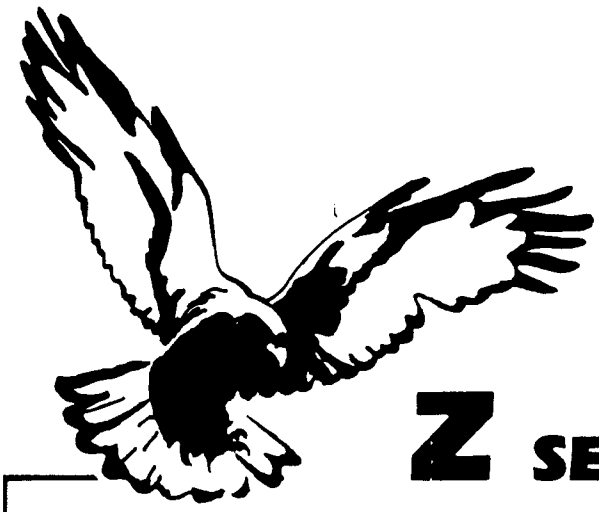
Actually this theory is not my own invention and you may have seen it expressed before but in its inverse form. Some kits or articles will tell you that the best way to unsolder something such as a 16 pin IC DIP is to use a small nippers and cut all the leads, freeing the IC from the board, then unsolder each individual pin. Naturally this works quite well from the boards perspective but leaves a useless I.C.

This is where the art comes into play. If an I.C. is defective then there is no reason to protect it, and you should instead try to protect the circuit board.

NEW FREE CATALOG
CP/M MSDOS
AFFORDABLE ENGINEERING SOFTWARE
TRSDOS PCDOS

- **LOCIPRO** Root Locus — **$69.95**
- **ACTFIL** Active Filter Design/Analysis — **$69.95**
- **STAP** Static Thermal Analysis — **$69.95**
- **MATRIX MAGIC** Matrix Manipulation — **$69.95**
- **RIGHTWRITER** Proofreader & Writing Style Analyzer — **$74.95**
- **ACNAP2** AC Circuit Analysis — **$69.95**
- **DCNAP** DC Circuit Analysis — **$69.95**
- **SPP** Signal/System Analysis — **$69.95**
- **PLOTPRO** Scientific Graph Printing — **$69.95**
- **PCPLOT2** High Resolution Graphics — **$69.95**

BV Engineering
Professional Software
2200 Business Way, Suite 207 • Riverside, CA 92501 • **(714) 781-0252**
VISA MasterCard



# Z SETS YOU FREE!

**Free to create computer environments right for you . . . free to automate repetitive tasks . . . free to increase your productivity. Z-System, the high-performance 8-bit operating system that flies! Optimized assembly language code — full software development system with linkable libraries of often needed subroutines —relocating (ROM and RAM) macro assembler, linker, librarian, cross-reference table generator, debuggers, translators, disassembler — ready to free you!**

**TERM III** New generation communications package provides levels of flexibility, functionality, performance not available until now. Replaces BYE and XMODEM . . . master/server local area network capability . . . public or private bulletin board and electronic message handling are integral features . . . auto-dial/answer, menu install . . . XMODEM (CRC/Checksum), MODEM7 Batch, Kermit, CIS, and XON/XOFF protocols . . . 100-page manual ..... **$99.00**

**Z-MSG** Rolls Royce of message handling systems . . . mates with TERM III or BYE for most advanced overall electronic mail/file transfer capabilities . . . menu installed . . . extreme configurability . . . many levels of access and security . . . word, phrase editor, field search . . . complete message manipulation and database maintenance ..... **$99.95**

**DISCAT** Elegant, menu and command-line driven file and disk catalog manager. Generates and controls multiple master catalogs, working catalog used for update quickness. Nine flexible modules easily altered by user for custom requirements. Works with Z shells (VMENU, VFILER, MENU), aliases, and multiple commands per line ..... **$39.99**

**ZCPR3: The Manual** Bound, 350 pages, typeset book describes features of ZCPR3 command processor, how it works, how to install, and detailed command usage. Bible to understand Z-System ..... **$19.95**

**ZCPR3 and I/OPS** Loose-leaf book, 50 pages, 8-1/2" by 11", describes ins-and-outs of input/output processing using Z-System. Shows how to modify your BIOS to include I/O redirection . . . complements **The Manual** ..... **$9.95**

**More missing links found — Z Application Progams! Fly with eagles! Our programs promote high performance through flexibility! Productivity results from dynamically changeable work environments, matching operator to tasks and machines.**

*Above programs require 48K-byte memory, ZCPR3, Z-Com, or Z-System, and Z80/NSC800/HD64180-based computer. Shipping from stock. State desired disk format, plus two acceptable alternatives. As payment, we accept Visa, Mastercard, personal checks, money orders, and purchase orders from established companies. We also ship UPS COD.*

*Call or write to place order or to obtain literature.*

**Echelon, Inc.** 101 First Street • Suite 427 • Los Altos, CA 94022 • 415/948-3820

On the other hand perhaps it's the IC that you wish to recover and the circuit board that is useless (obsolete or damaged, for instance). You'll have to make a judgement about this issue each time you unsolder anything.

Backing away from multi-lead IC's, it becomes obvious that the easiest thing to unsolder is a single connection part, typically only wires fall into this category. You can use the destructive technique of clipping the wire and then unsoldering the stub. Let's spend just a moment on this method. The advantage is that you can unsolder a stub by simply holding the circuit board in one hand, apply the iron, and when the joint turns molten, rap the edge of the board against something solid such as your workbench and the stub will just fall out, most of the time, stubborn stubs are not unknown. No need to bother with solder suckers, etc.. You may need to use a little solder wick to clean up the pad on the board, but rarely so. This is the great appeal of the destructive method, you lose the part but preserve the board and it's convenient.

Returning to the example of unsoldering a single wire, many times we won't wish to cut the wire because it will become too short, also stranded wire leaves a stub of many little wires that are actually difficult to tap out of the hole. In this case, it is better to use a solder sucker and remove the wire intact. After using the solder sucker you will find two possible conditions either the wire will pull right out with gentle tugging, or it won't. The reason it may not is that enough solder is left in the connection to hold the wire from releasing. More art now is required, you may be able to free the wire by using some solder wick and wicking out the remaining solder, typically this should only be necessary if you really didn't do a good initial job of sucking out the solder, perhaps you didn't use an iron sized to the joint, maybe you used a bulb type sucker and not enough vacuum was developed, maybe some solder remains on the opposite side of the board. Another possiblity is that wire is a very tight fit to the hole and even a small trace of solder will keep it held fast. In this case, what you may need to do is to fix the board in a holder, grab the wire on the opposite side with one hand (far enough away so you won't be burned) and apply the iron while you pull gently, as the iron heats the bit of solder it should lose its grip on the wire.

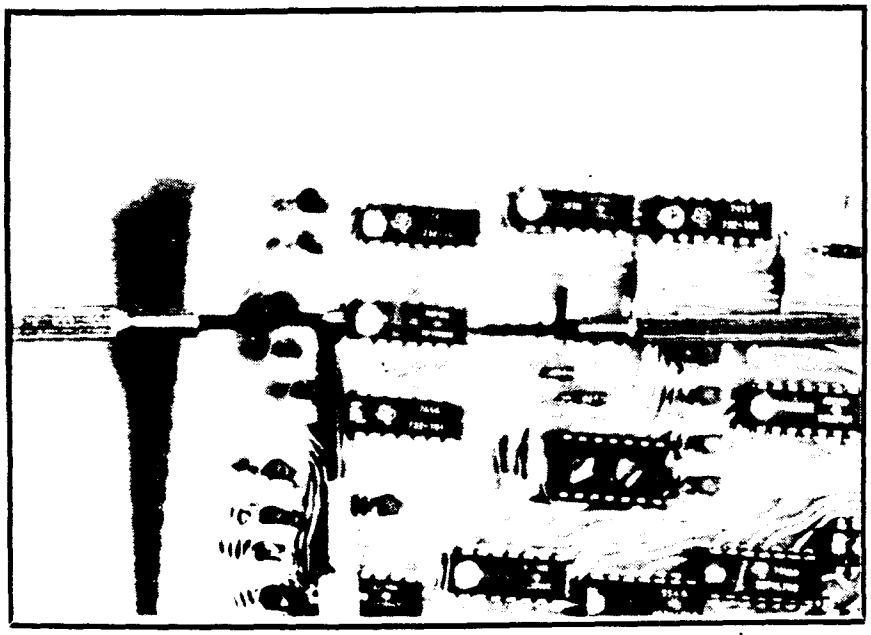


Figure 5: Jeweler's screwdrivers used to pry up an unsoldered DIP. On the left side a resistor acts as fulcrum for the 'lever', on the right another screwdriver at an angle is used with the levered blade threaded between some transistors.

Parts with two or three leads are very common. To unsolder these just remove solder from each lead as described above. About half the time the part will fall right out, and half the time it won't. Wiggle it, this should break any weak remaining solder bond and then the part will come out. If it wiggles freely but still won't come out check for solder blobs at the end of each lead, to remove any just wipe the iron across the lead, most of the solder will adhere to the iron and leave a smooth lead, for really stubborn blobs use solder wick. Check for any residual solder on the top side of the board. Rarely, the part still won't pull out, this is usually because the part was tightly fitted originally and friction is holding in it, just keep prying at it firmly but cautiously.

**Unsoldering A Dip**

The vast majority of digital IC's come in DIP packages, the word DIP is an acronym for dual in-line pins, the smallest has eight pins, four to a side, the largest 64 pins, 32 on each side. Parts with two or three leads are easy to unsolder and even if damaged in the process most are readily available and inexpensive. IC's fall into a middle ground since some are as common as resistors, others harder to find and some almost impossible to replace, and the number of pins is irrelevant in this regard. There are companies that make a living by recovering out-of-production IC's from circuit boards, testing them and re-selling to those who still need them for repairs, etc. Those firms don't care what happens to the boards, they want the IC's, generally we'll want to preserve the board and perhaps the IC. By the way, resistors, capacitors, etc. have leads, IC's have pins or sometimes legs but all three terms are used interchangeably.

If you don't want to save the IC then the best approach is to cut it off the board. Clip each leg as close to the IC body as possible, until the body falls away. Next get a small spring loaded heatsink, forceps, or tweezers with a rubber band to clamp it shut, invert the board and clip this tool onto a pin so that gravity aided by the tool's weight is acting to pull out the pin. Heat it with an iron, being sure to press the iron evenly against both trace and pin but don't wedge the pin, use a gentle touch. You want to be sure to contact both so that on double sided boards the heat will be transfered to the trace on top. Once the solder liquefies the pin should fall out. Repeat for each pin.

There is a variation that requires a little more dexterity and doesn't always work smoothly. In this method you omit the heat sink or forceps and just heat the pin until the solder liquefies then tap the board against the edge of your workbench and it should pop out. Alternately some people blow the pin out when the solder liquefies,

this requires dexterity and practice, and as a hot soldering iron is involved can be hazaradous. Occasionally the pin won't come out because it has wedged, don't try to tap or blow it out any more as you'll risk damaging the board by repeated attempts, instead use the first method described above.

Suppose that you want to get the IC out intact. The first step is to remove the solder from each pin exactly as you would for any other component. Use a solder sucker, bulb or spring type, and as in the technique above be sure that the heat transfers to the opposite side by butting the iron against both pin and trace.

Irregardless of whether you're trying to save the IC or not, be careful not to overheat each joint. Overheating can damage the printed circuit traces, especially the small ones that cluster near IC's. Hold the iron no more than four, possibly five seconds tops. To measure time, use a counting technique, such as saying One-Mississippi, Two-Mississippi, etc. It takes almost exactly a second to say each iteration, and any multi-syllabic state name will do such as One-Massachusetts, etc. Just don't alternate back and forth as that will confuse both time and geography, a sure-fire route to the Twilight Zone.

Allow a little time between the pins so that the IC can cool internally. Once all the pins have been cleaned, carefully inspect them (a magnifying glass is helpful). Look for a dark half-moon in each hole, this indicates that most of the solder has been sucked out. If the hole still looks shiny then solder probably remains in it. To fix this don't try to suck it out, instead re-solder the joint, that's right re-solder it then try to suck it out the solder normally. For some reason it works much better this way. If it still comes out shiny try wicking the solder out. As you work on a recalcitrant joint there's a terrific natural tendency to try too hard, to apply the iron too long, to press too hard—remember this and strive to avoid it.

Let's step back for a moment and take a look at an IC before it's inserted into a board. The pins will typically splay out, and to insert it they will have to be compressed or bent in. Once in position they will splay out again, pressing against the side of the hole which on double sided boards will be plated (the trace extends thru the hole to the other side). No matter how well the solder is removed a small residue will be trapped between the pins and the side of the hole. Individually this is a weak connection and in the case of small two lead parts this can be broken by just wiggling the part, but with DIP's each leg (either 14, 16, 20, 28 on up) contributes to the whole and will keep that IC firmly anchored in place.

To free up these ghost connections will require a simple tool and a moderately difficult procedure. Go out and buy a set of cheap jeweler's screwdrivers, yes cheap because they won't be used for driving screws, instead they will be employed as probes. Feel the ends of each screwdriver and if sharp then round them a little with a file, just enough so that they won't scratch if slid along on a circuit board. Now using a screwdriver with a blade just about the width of an IC's pin, place it against the board next to an IC pin. Holding it down so it won't jump over the pin push gently inwards with just enough force to break the weak solder bond inside the hole. You'll know this has happened because the pin will suddenly give way and then you'll see that it can be wiggled back and forth inside the hole. This technique requires great care and a lot of practice to learn just the appropriate amount of force, but it is necessary. Without this step you'll probably be unable to pry the IC off the board without breaking the IC and lifting some traces.

Sometimes as you attempt the first few pins on an IC you find that too much force is needed and instead of freeing the pin the screwdriver slips over it perhaps even bending it down, when this happens I use an extra step that usually helps. Take some solder wick as thin a width as you have and place it next to the pin just like the screwdriver blade then with the hot soldering iron press in gently. Let the heat of the iron do most of the work. The pin should move inwards in a few seconds and the wick will pull out the residual solder. After this the screwdriver should easily free up each pin. So little solder is involved that you can slide the iron along the wick as you do each pin and do several before trimming the wick is necessary.

Once each pin has been freed, turn the board over and using one of the


THE BEST Z80 ASSEMBLER ON THE MARKET JUST GOT BETTER!

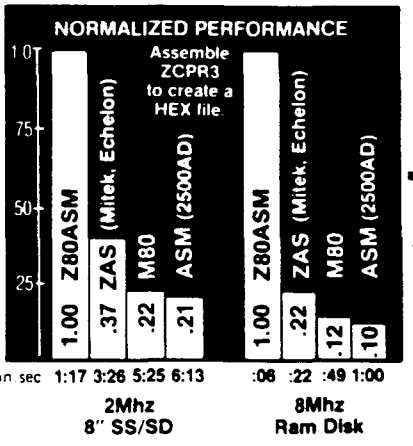
Z80ASM

NOW ONLY $49.95

DON'T ASK HOW OURS CAN BE SO FAST . . . ASK WHY THEIRS ARE SO SLOW!

". . a breath of fresh air . ."
*Computer Language, Feb. 85*

". . in two words, I'd say speed & flexibility",
*Edward Joyce, User's Guide #15*




Now fully compatible with M80 in .Z80 mode with many extensions. Time & date in listing, 16 char. externals, plus many other features.

To order, or to find out more about our complete family of development tools, call or write:

SLR Systems

1622 N. Main St., Butler, PA 16001
(800) 833-3061, (412) 282-0864
Telex 559215 SLR SYS

C.O.D., Check or Money Order Accepted

SHIPPING: USA/CANADA - $3 • OTHER AREAS - $10
Z80 CP/M compatibility required

Figure 6: Center of photo shows three pads attached to heavy heat absorbing traces while in between are three isolated pads which cannot tolerate heat well, other pads are normal types.

larger screwdrivers as a lever pry alternately under each end of the IC. If everything goes right the IC will resist a little at first and then suddenly lift right off, and you'll find little or no trace damage. Sometimes there won't be any way to use a screwdriver to get under the IC because of crowded conditions, you'll have to use your imagination to come up with some tool that will do the job, some commercial IC pullers made for removing them from sockets can be used in this instance, although it will be a little awkward.

The above technique takes only a little more time and effort than the cut and destroy method. It can save a valuable IC or at least allow you to test an IC after removal to verify that it was or was not malfunctioning. It's worth the effort to practice it but there are no sure things in IC unsoldering, sometimes it's a snap and other times it's the IC that 'snaps'.

To summarize: Step 1) Remove as much solder as possible using a desoldering tool. 2) Free each pin using a small screwdriver as a probe. 3) If step 2 won't work use some solder wick and a hot iron to clear any remaining solder, repeat step 2. Single sided boards will usually respond to just the first two steps, double sided boards due to the plated-thru holes will often require the full treatment.

**Snags and Snarls**

There are many things that can conspire to defeat your best efforts to unsolder a DIP, fortunately only one or two problems will rear up at one time. If you can identify the problem then you may take steps to overcome it. What follows is a rogues gallery of the most common ones.

By the far the most frequent trouble is caused by residual solder that bonds the IC firmly into position. You know this has happened when you try to free the lead or pry the IC off the board. There is a fine line between the amount of force needed to remove an IC that is going come off cleanly and one that is stuck in place, exceed that line and the IC will break or some traces will come off with it. Practice is very helpful to allow you to estimate the force needed but experience is indispensable because boards are different, what works on one will fail on others.

Here are some hints on defeating the residual solder dilemma. Always inspect the top of the board, look for solder fillets around the IC pins, double sided circuit boards with plated through holes are the prime culprits. Proper unsoldering techniques should remove these bonds but occasionally one or more will remain. Refer back to the procedures outlined above and try to resolder and then unsolder these joints. If you find quite a lot of them, then an alternative method may help. This involves using an unsoldering tool that contains a built in sucker (remember the 'missing link'), place the tool over the joint let it liquefy the solder then slide it a little to move the IC pin around while releasing the bulb or trigger to suck up the solder.

The above method also works on circuit boards that have small hole diameters, where the standard procedure is not adequate. Once in a great while you may encounter boards with such tiny hole diameters that nothing may work. At this stage we should point out that whenever something is soldered together there is the implied assumption that it will never need to be unsoldered, some designs can make it very hard indeed to safely unsolder an IC. If you absolutely must accomplish such a task there is a tool that may do it for you. It's an accessory for certain soldering irons that is shaped so that it contacts all the pins of an IC at one time. It's not inexpensive and you need one for each IC size and above the 16 pin size it becomes rather impractical, therefore we'll only note its existence without a detailed examination of this tool type.

There exists yet another reason why certain IC pins may contain residual solder. Any pin that is connected to a large trace such as a ground or power pin will prove difficult to unsolder. Using a larger tip on the iron will often help as well as using solder wick. Sometimes these pins may resist all effort to remove the solder, but there is usually only one or two per IC. These are easy to deal with if you have three hands.

Seriously, the third hand is really any sort of holder for the board that lets you reach around to the component side with one hand and apply the soldering iron with the other hand. The scrap wood holder described earlier or a commercial product work equally well. Since you'll be grasping the IC by its plastic body you should be able to use your fingers, and if the body gets so hot as to burn your fingers then you've overheated the IC. You could also use a tool to grasp the IC, the Radio Shack IC Extractor is fine or you can make one from scrap metal by studying the basic idea. Grasp the IC so that you can exert firm but even pull on it, apply the soldering iron to the recalcitrant joint and as it liquefies start pulling. Even when two stubborn joints are involved if they are on opposite sides of the IC then you'll be able to free one side and then do the other. Afterwards you'll need to bend the leads back into shape

but that's easy if you work slowly. The same method can also be used with stubborn resistors, capacitors, etc.

**Crimping and Staking**

Some circuit boards needing repair may have been assembled using these industrial techniques. When the components are mounted on the boards by hand prior to a one step soldering process such as dip or wave soldering, crimping and staking serve to keep the parts from falling out while in process. Unsoldering those parts later can prove extra difficult.

Crimping is done to small two lead parts, a special forming tool bends the leads so that when the part is inserted a high degree of friction with the board holes keeps it in place. To remove these parts is not too hard, you may to need wiggle them about a little more vigorously and occasionally the three hand method helps.

Staking is done to IC's, two or more leads usually at diagonal ends are bent over flat against the pad to lock the IC into place. Once soldered, the staked pins are really tough nuts to crack. To deal with them first proceed to unsolder all the pins as you normally would, in the case of the staked leads remove all the solder you can from on top of them. Because of the staking they will still be firmly anchored. To free them mount the board in your holder and take a small X-acto® or utility knife in one hand and your soldering iron in the other. Apply the iron over the pin, holding the knife so that you can slip it under the staked-over pin as the solder liquefies. Merely slip the knife under and lift a little, remove the iron, wiggle the knife slightly so the solder bond doesn't reattach. Use the greatest of care when attempting this procedure, you'll be using a sharp knife and a hot iron, plan your moves so that if you slip neither knife or iron is likely to injure your hands. Absolutely wear eye protection, even better wear a face shield if you have one. Next take a small screwdriver and very slowly bend the lead into an upright position, bend too fast and the lead will fatigue and break. Use a good desoldering tool to clean out the hole (you may need to use the techniques described for stubborn joints).

For various reasons staking is becoming rare so you may never encounter these problems. Since it makes unsoldering so much more difficult you'll appreciate why we don't encourage its use when assembling a kit. Even the cut and destroy method of IC removal is partially ineffective against staked leads.

**The Isolated Pad**

This is hard to categorize, is it an unsoldering problem or a board repair problem? Certain IC's do not have all their pins connected to the circuit, the pad for those pins will have no trace leading away from it, it's isolated. If

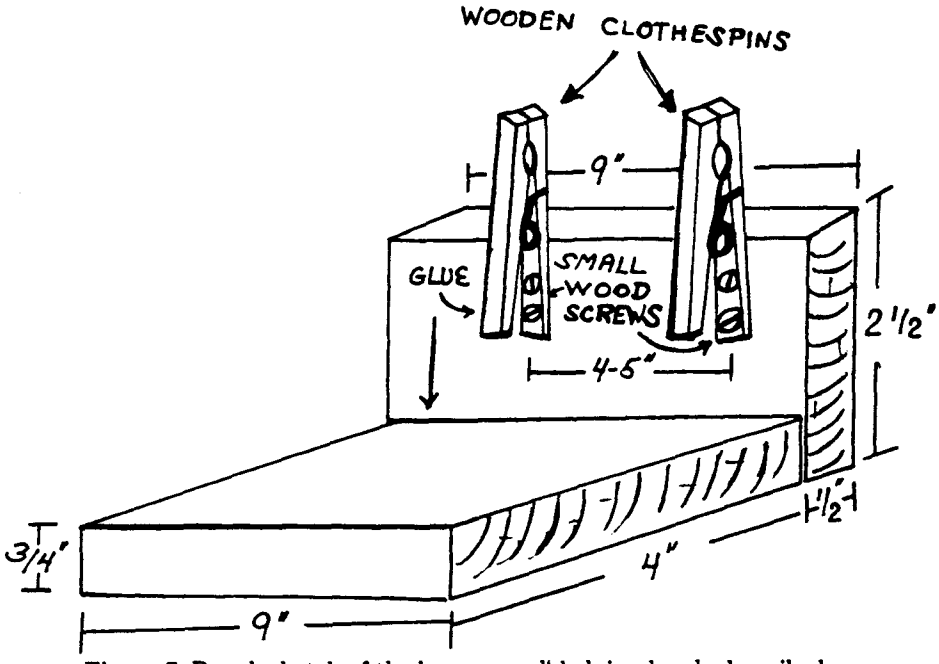
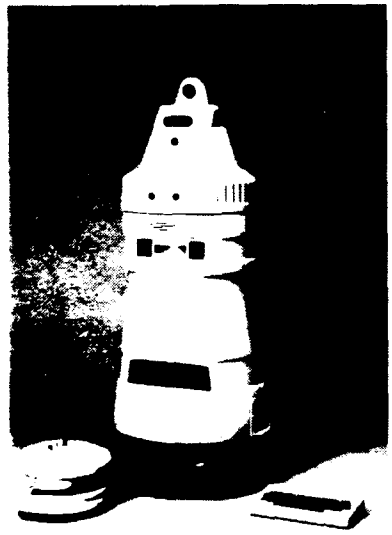


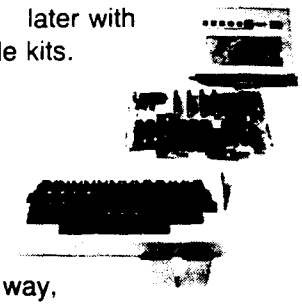

**Figure 7: Rough sketch of the 'scrap wood' helping hands described in the article, all dimensions and materials can be modified as needed.**

GEMINI
THE AUTONOMOUS ROBOT
IS NOW PRICED FOR EVERYONE!



Buy each subassembly as a kit or factory assembled and create your own GEMINI Robot.

Or, for convenience, start with **GEMINEX** our starter kit, and expand to GEMINI later with upgrade kits.



Either way,
**Buy** a piece of tomorrow
**TODAY**!

CALL or WRITE For Our FREE Brochure.

arctec systems
9104 Red Branch Road
Columbia, Maryland 21045
(301)730-1237
Telex 87-781

you try to unsolder one of these leads, you may find that the pad will delaminate, it may even be sucked right up into the desoldering tool. This is especially likely on single sided boards. The best news we have for you is, it's not your fault. Isolated pads simply can't dissipate heat very well, without a trace heat is confined in the pad and quickly delaminates it. On double sided boards the plated thru holes help anchor the pads in, and sometimes there is a trace but on the top side under the IC where it can't be seen.

Of course the pad isn't actually necessary, it's there because the pattern used to make the printed circuit included pads for all the leads. You only need to repair an isolated pad for appearance sake or if you need to attach a jumper to the lead, a pad facilitates that. And in the next article we discuss glue-on pads that make such a repair rather easy to accomplish.

**To Socket Or Not To Socket That Is The Question.**

With apologies to the Bard of Avon, this brings us to the matter of using sockets or eschewing them. You may ask why discuss this in an article about unsoldering things, the reason is very simple – if you control how something is put together then you also affect how difficult it may be to take it apart.

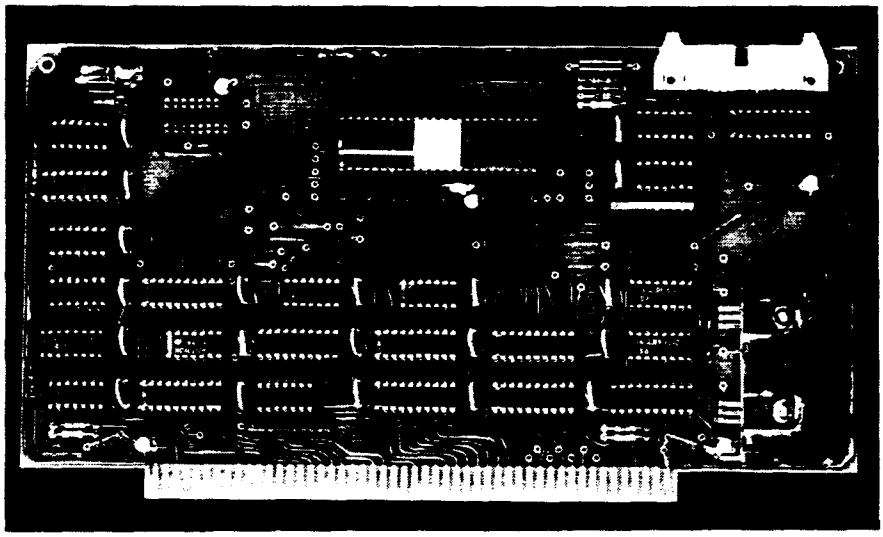
Looking at commercially produced circuit boards, you'll notice the distinct lack of sockets. There is one very obvious reason for this, sockets cost money and for companies producing large numbers of boards omitting them adds up to big savings. For the casual builder the cost of sockets is almost insignificant especially when weighed against the convenience they provide. From this it might seem that you should always use sockets, some people including myself would take this position, others hold opposing views and it may be worthwhile to delve a little deeper into the reasons for the controversy.

There are some additional and subtle reasons why commercial builders omit sockets. If you read the article on soldering you may remember the part about how corrosion affects all metals. It happens that soldering a connection almost entirely eliminates that danger, a soldered joint is considered in the trade 'gas-tight' that is unaffected by the oxygen and other contaminants in the air. Wire-wrapping, and some other connection methods also produce gas-tight joints. But the junction between an IC's pins and a socket's metal are not truly gas-tight. When sockets first came into use they were particularly prone to this problem and earned a reputation for unreliability.

Socket manufacturers responded to this by making a whole series of improvements. They designed better contact points, and then they re-designed them again, until today each socket maker trumpets the effectiveness of their design. They began to gold plate the contact points, gold being extremely resistant to corrosion. All these steps taken together have achieved modern sockets that perform equivalently to a soldered connection. Yet the early problems still influence some builders, who may not use sockets at all. Others use only a certain brand, or especially avoid some brands. Buying from a reputable source should result in a good quality socket, always chose the gold plated type as the small extra cost is worth it.

Now we circle back to a very subtle reason for omitting sockets, directly related to unsoldering. Commercial assemblers have available repair

*(Continued on page 33)*

**INTELLICOMP Introduces Inexpensive S-100 68008 CPU Board**

The card pictured above is $65 for the bare board, $210 for the kit, or $265 assembled and tested. It uses only standard parts. A sample BIOS for CP/M 68K is available on disk for $20. The board works fine with Digital Research Computers 64K RAM boards and semi disks. A detailed description of the board appears in Issue 16 of The Computer Journal.

For additional information call Intellicomp, Inc. at (614) 846-0216 (evenings best time) or write to :
Bruce Posey
**Intellicomp, Inc.**
292 Lambourne Ave., Worthington, OH 43085

# Analog Data Acquisition and Control

## Connecting Your Computer to the Real World

by Jerry Houston

### Analog Fundamentals

After my first data processing class in college (Business BASIC, of course), I felt ready to take over the world. Nothing could stop me now—I could make a computer do anything! (Most readers can probably remember the feeling.) The shock of reality came a little later, while I was still in school, and working for a security company.

My supervisor occasionally installed some fabulously expensive solid-state security systems, and it occurred to me that a computer like the Commodore 64 (just then becoming really affordable) could make all the decisions needed for commercial security. I was right, of course, but nobody I talked with knew how to make the computer respond to inputs from security switches. I was too busy with full time work and full time school to take on an additional study of electronics, and begrudgingly let go of the idea.

Actually, I gave up doing anything about it at the time, but I couldn't quit thinking of fascinating projects that would be possible if my computer weren't limited to working just with its peripherals. I had no shortage of ideas—just no way to make any of them work.

I was the kind of person who passed over magazine articles and advertisements dealing with analog/digital electronics. I'd never had the opportunity to learn much of the terminology involved, and dismissed the subject as something that would probably only interest a technician anyway. But when my business and computer training landed me a non-technical job in a company that makes A/D converters, I managed to get myself hooked, but good!

For the totally uninitiated, as I certainly was, I should explain that analog/digital conversion provides a way for any computer to "talk with" the rest of the world—the Real World that exists outside the computer. There are analog/digital data acquisition and control systems... whew! let's call them "ADCs" from now on ... available from a number of companies, that not only let the computer MONITOR its environment but allow it to CONTROL its environment as well.

These ADCs provide digital controlled outputs that can be used to operate relays or indicators, and some even come with a transmitter to operate the familiar BSR X-10 series of AC line carrier remote control modules. These are the modules available at Radio Shack stores and elsewhere, that are turned on or off by radio-frequency signals sent over the AC lines. Once a computer can operate BSR modules, a programmer can easily design a sophisticated home control (and energy management) program using convenient wireless remote control.

Most comuters that we know of today are DIGITAL computers. They respond to electrical impulses that are either there, or they aren't. Bits in our computers are either ON or OFF. By bunching those bits into bytes and words, we can express larger digital values, but information in a computer can all be traced back to those two values, ON or OFF. Most of the important information in our environment comes in ANALOG form—it varies continuously along a scale of some kind, and has meaning only because it relates to some observable physical phenomenon that we can perceive.

If we hear from an astronomer that the normal temperature on the surface of a certain planet is 300° Kelvin, most of us would have learned very little. If the same astronomer then says, "by the way, water freezes at 273° K and boils at 373°K", suddenly that first statement means a lot more. Without doing a lot of tricky math, it's easy to see that 300°K is comfortably above freezing, and a long way from boiling—phenomena that we all understand.

Analog information is everywhere we look. If we glance at the speedometer, we don't want to know whether we're moving or not, we want to know "how fast". It doesn't help to know that there's water in a swimming hole—before we dive in, we need to know "how deep".

A computer can do some remarkable things if it's able to gather analog data from its environment, and if that data is converted into a digital form the computer can deal with. Knowing "how dry" the lawn is, the computer can decide when to turn on the sprinklers. Knowing "how bright" the sun is, the computer can open the drapes in the winter, or close them in the summer. "How fast" a drill is sinking into the ground can be important to an oil well driller.

Actually, there are few fields of endeavor that can't benefit from computer automation, now that chips are cheap and—you guessed it!—easy-to-use ADCs are available. These devices take care of the details of interfacing a computer to the Real World such that a programmer can concentrate on writing code to solve the problem at hand. He doesn't have to become expert at digital electronics.

### Another Peripheral

ADCs come in various packages. Some are printed-circuit cards, designed to plug into a bus slot of a particular computer. Others are external devices that look a lot like a modem. Because the internal variety link directly with the data bus of the computer, they often provide faster operating speeds than external models.

External ADCs usually communicate with the computer via the RS-232 C serial interface. One advantage to such a device is that it can be located close to the sensors being read, often hundreds of feet away from the computer, as the length of certain kinds of sensor leads can be critical. Another advantage is that the external ADC won't require a particular computer to operate it. Communicating with standard ASCII characters, it'll work with the computer at home, the one at the office or lab, or the new one you get next year.

Regardless of its type of mounting, ADCs can be intelligent or operate according to discrete (fixed) logic. Models

# NGS FORTH

*A FAST FORTH, OPTIMIZED FOR THE IBM PERSONAL COMPUTER AND MS-DOS COMPATIBLES.*

**STANDARD FEATURES INCLUDE:**

- 79 STANDARD
- DIRECT I/O ACCESS
- FULL ACCESS TO MS-DOS FILES AND FUNCTIONS
- ENVIRONMENT SAVE & LOAD
- MULTI-SEGMENTED FOR LARGE APPLICATIONS
- EXTENDED ADDRESSING
- MEMORY ALLOCATION CONFIGURABLE ON-LINE
- AUTO LOAD SCREEN BOOT
- LINE & SCREEN EDITORS
- DECOMPILER AND DEBUGGING AIDS
- 8088 ASSEMBLER
- GRAPHICS & SOUND
- NGS ENHANCEMENTS
- DETAILED MANUAL
- INEXPENSIVE UPGRADES
- NGS USER NEWSLETTER

*A COMPLETE FORTH DEVELOPMENT SYSTEM.*

**PRICES START AT $70**

**NEW◆HP-150 & HP-110 VERSIONS AVAILABLE**

**NEXT GENERATION SYSTEMS**
**P.O.BOX 2987**
**SANTA CLARA, CA. 95055**
**(408) 241-5909**


are available that can acquire data at high speed and store it in an internal buffer, releasing it to the computer later for processing. Other kinds communicate with the computer in real time, passing along each reading as it is taken, and all the processing intelligence resides within the computer.

**The Hardware**

With ADC systems, like any other product, the cost of the device depends a lot on the features that it offers. Since most ADCs do pretty much the same thing, the range of prices depends more on HOW they do it. The main considerations are:

1. Number of each kind of input or output.
2. Resolution (accuracy) obtainable.
3. Speed of analog sampling.
4. Additional features, such as BSR controller, analog outputs, memory buffers, etc.

As in making any other kind of purchasing decision, the prospective user needs to analyze his needs and then to buy accordingly.

The very simplest ADCs offer as few as 4 or 8 analog intputs and little else. In fact, they're not really ADCs, because the "C" is missing – the control function. They're useful for monitoring analog sensors and laboratory apparatus, where the purpose is to log data to a printer or disk file.

The next step usually includes some number of digital outputs as well as analog inputs. With such an interface, a computer can monitor the temperature, for example, and turn on a furnace or air conditioner when it's needed.

If the ADC also includes digital inputs, the computer can monitor ON/OFF conditions such as security switches or event counters. Since digital inputs can be monitored directly by a computer, an ADC isn't really needed for this particular job, but digital inputs on an analog interface are pretty handy anyway. If the computer controls a home environment, it needs to monitor analog parameters (light, temperature, moisture, sound, and so on), and if digital inputs are available the computer can handle the security needs as well. The ADC will pass properly conditioned digital values to the computer in a manner that makes programming easy.

Two other features are sometimes useful, the BSR transmitter already mentioned, and D/A outputs. D/A conversion is the reverse of A/D, naturally, and means that the ADC will output an analog voltage that represents a digital value passed to it by the computer. D/A outputs can be used to control the speed of motors, operate chart recorders, or provide a variable-voltage for other special purposes.

Resolution has to do with the accuracy and sensitivity of the analog inputs, and ultimately with the cost of the equipment and the speed at which it can operate. Since the resulting digital value is sent to the computer as a collection of bits that are either ON or OFF, a convenient and common way to refer to the resolution of an ADC is in terms of the number of bits used. ADCs are commonly available that provide 8-bit or 12-bit resolution, and not-so-commonly with 16-bit or higher.

The size of each increment of resolution has to do with the overall range that can be read by the inputs (represented as a voltage) and the number of bits of resolution. Since each bit can represent only one of 2 conditions (ON or OFF), 8-bit resolution provides 2 raised to the 8th power, or 256 increments. 12-bit resolution offers 2 to the 12th, or 4096 increments. If the ADC is also capable of determining polarity, then it can differentiate that number of increments to either side of zero (doubling range with the same resolution).

To put resolution into perspective, consider that you might want to monitor and control the temperature of a heat-treating furnace. Your sensor might be one that operates over a range of 0 to 2000°F and you probably would have the ADC input well matched to the sensor, so that none of its range was wasted. With 8-bits of resolution, each increment you could measure would be nearly 8° wide. With 12-bit resolution, the increments would each be less than 0.5°, and with 16-bit resolution (1-in-65,535), you could be accurate to within 0.03°.

I promised you that resolution has something to do with both cost and speed, didn't I? Well, as you might expect, 12-bit systems are generally more expensive than 8-bit systems and usually slower. How slow? Even inexpensive 8-bit systems can often read thousands of samples per second. If your application requires that kind of speed AND your computer and language can do something with infor-

mation coming in that fast AND you can get by with 1-in-256 resolution, then consider one of the fine 8-bit systems that are available. Other factors being equal, it will cost the least.

If you need more resolution, 12 bits will provide 1-in-4096, as mentioned earlier. Most 12-bit ADCs on the market will sample at speeds from 5 or so up to 100 samples per second. As with all the other factors, your application is important here. To digitize speech patterns might require many thousands of samples per second, whereas someone monitoring the movement of a glacier might be well served by one measurement a week!

Some ADC systems are made up of a basic circuit board to which optional modules can be connected, thus providing a choice of the features mentioned above. Also, some manufacturers are responsive to custom orders, and make options available when needed. An ADC that normally includes six controlled outputs and a BSR transmitter is also available with twelve controlled outputs and no transmitter. People working in the field (with no way to use BSR modules) sometimes opt for the additional outputs.

Some companies offer the ultimate in options, intelligent microprocessor-controlled data acquisition and control systems designed and built just for a particular task. With no need for a supervising computer, such systems can be very inexpensive in reasonable quantities. If your needs are of an OEM nature, or your application requires a number of systems, ask if one can be built to your specifications.

**Applications**

The following applications are all made possible by a data acquisition and control system. In most cases, the programmers involved are NOT computer professionals, but specialists in their own particular fields. They're able to program their computers to make the necessary decisions, and that's pretty much all that's needed. Some of these uses for Real World computing may suggest ways that an ADC system could make your life easier!

(1) WSB-TV in Atlanta uses a Commodore 64 to collect meteorological data from 13 remote weather stations in the surrounding region to provide detailed reports of current conditions. Each station consists of a modem, an ADC, and sensors to measure wind speed and direction, temperature, and relative humidity. Every fifteen minutes throughout the day, the C-64 calls the remote stations in sequence and acquires up-to-the-minute meteorological data. The Atlanta area is characterized by small agricultural areas that differ dramatically in weather. Because of this system, farmers in each of these regions now have access to vital regional weather information not available until now.

(2) A southern California electronics firm is using an ADC, controlled by an Apple IIe computer, to perform final manufacturing tests and calibration of medical power supplies. All 16 of the ADC's analog inputs are employed to measure voltages at test points under various load conditions. The power supply loads are applied systematically via the ADC's power line carrier remote control system. The computer is programmed to prompt the technician when an adjustment is required, or a unit is defective, and to produce a printout of the final test results. Even the clock frequency is monitored via a frequency-to-voltage converter. It is estimated that the automated power supply checkout is now accomplished in one-seventh the time that a technician previously required.

(3) Amana has reduced the time required to test room air conditioners from 20 minutes to only 2 minutes through the use of an ADC monitored by a Commodore 64 computer. The new test, which monitors the rate of temperature change and the amount of current that is consumed in the process, no longer requires a special testing booth and dedicated equipment. It requires less attention from technicians, can be done on the factory floor, and saves a considerable amount of energy. The first system is working so well that a second is being developed to run concurrently. A third Commodore 64 is used to process the test data in the DP department.

(4) A graduate student at the University of Southern California is using an ADC and a Macintosh computer in conjunction with a spectrofluorometer and spectrophotometer to investigate the optical properties of marine phytoplankton in the Arctic Ocean. Samples of natural populations of marine phytoplankton are filtered and then placed in the spectrofluorometer or spectrophotometer for measurements of fluorescence excitation and spectral absorption. The output is directed to the Macintosh through the ADC and has the advantage of allowing real-time display of the data, rapid processing, and graphical representation.

(5) A former pharmacist, now a quadraplegic confined to a wheel chair, uses an ADC with an IBM-PC and Keytronics 5152 Voice-Actuated keyboard to control and monitor his environment. He uses the BSR X-10 transmitter feature of the ADC to control lights and appliances in response to voice commands that he gives to his computer. This is an early start for him — soon he'll be able to control his heat pump, home security system, and other devices in the same way. His success in controlling his environment has prompted him to develop and market such systems to aid other handicapped persons, using the Corona PC when voice-actuation is required, and the Commodore-64 as a lower-cost alternative for those cases in which the user is able to operate a standard keyboard.

(6) A grower of miniature roses in Seattle uses a Radio Shack Model 100 portable computer and an ADC to monitor and control his commercial greenhouse. Analog sensors monitor soil moisture, soil and air temperatures, relative humidity, and light

BLANKENSHIP BASIC
GIVES YOU MORE FOR YOUR MONEY!
FOR THE APPLE II+, IIe and IIc
DOS 3.3 VERSION OR PRODOS VER. $25
BOTH FOR $39.95
ADD $2.00 POSTAGE AND HANDLING
10 DAY MONEY BACK GUARANTEE
APPLE is a registerd trademark of APPLE Computer Inc.

1. Real interpreter, not a pre-processor
2. WHILE-ENDWHILE and REPEAT-UNTIL loops
3. True IF-THEN-ELSE-ENDIF (using WHEN)
4. PRINT.USING, FILE, MERGE, RANDOMIZE
5. PRINT and TAB commands work in HIRES
6. 80 columns supported on IIe and IIc
7. Full editor with AUTO-NUM and RENUM
8. Fast SORT, SEARCH and INSTR$ commands
9. BOX, BOXFILL, DRAW.USING and SOUND
10. Listings are indented automatically
11. DISK command replaces DOS's CHR$(4)
12. DEFINE and PERFORM named procedures
13. 99% upward compatible with Applesoft
14. All commands entered normally, no &'s
15. 100's of satisfied users world wide
16. FREE newsletter available to owners
17. Makes your Apple into a NEW machine!!

MAIL CHECK TO: JOHN BLANKENSHIP,
P. O. BOX 47934, ATLANTA, GA 30362

**FOR TRS-80 MODELS 1, 3 & 4**
**IBM PC, XT, AND COMPAQ**

**The MMSFORTH System. Compare.**

- The speed, compactness and extensibility of the MMSFORTH total software environment, optimized for the popular IBM PC and TRS-80 Models 1, 3 and 4.
- An integrated system of sophisticated application programs: word processing, database management, communications, general ledger and more, all with powerful capabilities, surprising speed and ease of use.
- With source code, for custom modifications by you or MMS.
- The famous MMS support, including detailed manuals and examples, telephone tips, additional programs and inexpensive program updates, User Groups worldwide, the MMSFORTH Newsletter, Forth-related books, workshops and professional consulting.

**MMSFORTH**

**A World of Difference!**

The total software environment for IBM PC, TRS-80 Model 1, 3, 4 and close friends.

- Personal License (required):
  - MMSFORTH System Disk (IBM PC) . . . . $249.95
  - MMSFORTH System Disk (TRS-80 1, 3 or 4) 129.95
- Personal License (optional modules):
  - FORTHCOM communications module . . . . $ 39.95
  - UTILITIES . . . . . 39.95
  - GAMES . . . . . 39.95
  - EXPERT-2 expert system . . . . . 89.95
  - DATAHANDLER . . . . . 59.95
  - DATAHANDLER-PLUS (PC only, 128K req.) 99.95
  - FORTHWRITE word processor . . . . . 175.00
- Corporate Site License Extensions . . . . . from $1,000
- Some recommended Forth books:
  - UNDERSTANDING FORTH (overview) . . . $ 2.95
  - STARTING FORTH (programming) . . . . . 18.95
  - THINKING FORTH (technique) . . . . . 15.95
  - BEGINNING FORTH (re MMSFORTH) . . . 16.95

Shipping/handling & tax extra. No returns on software.
Ask your dealer to show you the world of MMSFORTH, or request our free brochure.

**MILLER MICROCOMPUTER SERVICES**
**61 Lake Shore Road, Natick, MA 01760**
**(617) 653-6136**

levels. Watering and fertilizer application are automated by this system, as well as temperature and humidity control.

(7) The Environmental Engineering group at Louisiana State University's Civil Engineering Department is developing a remote water quality monitoring and control system for use in soft-shell crab mariculture systems. The system is based on a Kaypro computer and an ADC, and the objectives of the research include control and optimization of the molting process in blue crabs in totally enclosed aquatic systems. Real-time data on water quality and system status are collected by conventional laboratory instrumentation interfaced to the microcomputer through an ADC, monitoring pH, temperature, dissolved oxygen, light and water levels.

(8) An environmental consulting firm is currently studying the change in air quality resulting from air passage over two Seattle area sanitary landfills. Data logging and sampling stations consisting of an ADC, a NEC PC-8201 portable computer, a wind vane, an anemometer, and an air sample pump were installed at locatons upwind and downwind of the landfills. The NEC is programmed to monitor running averages of wind speed and direction, and to activate the sample pump when the wind parameters are within appropriate limits. The environmental data as well as sample duration and time are stored in RAM. Each system is battery powered and uses only 65 ma for monitoring and 350 ma when the pump is activated.

(9) The City of Grand Rapids uses an Atari home computer and an ADC to monitor the cyanide content in the wastewater they treat. Cyanide is a pollutant that threatens the microorganisms used to treat waste, and wastewater that contains cyanide must be shunted into a special holding area from which it can be released later at a rate that will cause no harm.

(10) To study the formation of patterned ground in arctic permafrost regions, an ADC in conjunction with a Radio Shack Model 100 was used by University of Washington Quaternary Research Center scientists to monitor strain gauges and temperature sensors buried in the soil. Four systems have been deployed near Spitzbergen in Norway to record stresses and soil motion resulting from the annual freeze-thaw cycle. The soil surface in this area is marked by a distinct pattern of stone circles thought to result from the slow overturning of the soil in a convection-cell manner. The systems are battery powered, and cold-room tested to -40°C prior to field installation.

(11) A fruit storage company in Washington State uses an ADC and a Kaypro computer to monitor and control special rooms in which apples are stored. Temperature, light, and relative humidity are carefully controlled so that the fruit will become dormant, but not die, and it can be brought to market months later in just-picked condition.

(12) Two professors in the Department of Urban Planning at the University of Washington are using an ADC with an IBM-PC to study the effects of lighting glare and solar radiation on personal comfort and energy management for new building design and construction. Subjects are placed in a room with a desk and lamp and asked to determine what level of light they find to be most comfortable to work with. The subject controls a hand-held device which, through the ADC, is used to brighten and dim the level of solar light emitted by an artificial 'window' in relationship to the level of internal light, while maintaining a total illumination constant.

(13) A sophisticated home control system has been developed by a Seattle resident, using an ADC and a Macintosh. The system monitors and graphs energy use, temperatures, basement moisture, and a variety of security sensors. When a possible security intrusion is detected, lights are turned on progressively closer to the source of the alarm, simulating someone approaching and turning on lights. If efforts to enter continue, a conventional alarm is sounded. Home temperatures are controlled to different setpoints according to time of day and occupancy.

(14) The maker of a popular brand of disposable diapers uses an Epson HX-20 and an ADC equipped with a moisture sensor to test new types of lining materials. This research should lead to more efficient linings for the diapers, and dryer babies.

**Vendor Listing**

A listing of suppliers that manufacture ADC equipment is provided for the convenience of those who want to

find out more about particular equipment. The usual authors' disclaimers must apply, along with my apology to any company that I haven't found out about yet.

The products that I have listed range from inexpensive to not so, and from simple to complex. I've included ADCs that are useable with a variety of computers, and left out some that are an accessory for one system only. It is therefor not a complete listing, but a very good place to start. Because many of the products are better known by name than the companies that make them, they're listed alphabetically by product name in the vendor list.

**Program Narratives**

Two programs accompany this article. Program One is written in Turbo Pascal® and measures the performance of ADC systems in terms of how fast analog channels can be read. Program Two is written in MTBASIC® , and lets the computer operate as a multichannel data logger.

In each case, code is used that reads the analog inputs of the ADC-1 Data Acquisition and Control System from Remote Measurement Systems, Inc. of Seattle. This is a popular ADC system, and the one most familiar to the author, but it is not at all difficult to substitute channel reading instructions for an ADC from another manufacturer. In each program, the code used to acquire analog data is identified as to its function. In Program Two, the first seven tasks look very much the same, but each has its own unique values, making it necessary to repeat a similar routine seven times to read seven channels independently.

**Program One (HISPEED.PAS)**

Two characteristics of data acquisition and control systems that are often important to the user are (a), the rate at which analog data can be acquired and (b), the effective resolution available. Both of these parameters are easy to measure with a computer program designed to read the ADC.

HISPEED starts with FUNCTION ACD, which converts BCD (binary coded decimal) values from a clock card into ASCII decimal representation for display on the computer screen and for manipulation as decimal values by the program. This program was written to

**Vendor List**

| | |
|---|---|
| Model | ADC-1 Data Acquisition & Control System |
| Manufacturer | Remote Measurement Systems, Inc.<br>2633 Eastlake Ave E, Suite 206<br>Seattle, WA 98102<br>(206) DATA-255 |
| Features | 16 Analog inputs of 12-bit resolution plus sign, 4 digital inputs. 6 controlled outputs, transmitter for BSR R/C modules. Interfaces with TTL ports or RS-232. A/D sampling rate of 20, 50 or 100 samples/sec. CMOS for low power use - can be battery-powered. |
| Cost | $449 standard model. Many options available. |
| Model | BUSSter D16R |
| Manufacturer | Connecticut Microcomputer Inc.<br>150 Pocono Rd<br>Brookfield, CT 06804<br>(203) 354-9395 |
| Features | 16 analog inputs of 8-bit resolution, 100-byte buffer for storing readings. A/D sampling rate 100/sec. RS-232 or IEEE-488 available. |
| Cost | $495 for model described. Devices also available for digital input/output, and analog outputs, $495 to $695. |
| Model | MTM1 |
| Manufacturer | Software Science<br>PO Box 44232<br>Cincinnati, OH 45244<br>(513) 561-2060 |
| Features | 8 analog inputs of 8-bit resolution, 12 digital outputs. RS-232. A/D sampling speed dependent on baud rate, up to 480/sec. |
| Cost | $249 circuit board, $89 power supply, $129 enclosure. |
| Model | ONE/05 |
| Manufacturer | Tarus Computer Products, Inc.<br>1755 Woodward Dr<br>Ottawa, ONT K2C 0P9<br>(613) 226-5361 |
| Features | 16 analog inputs of 12-bit resolution, 16 digital I/O channels. Has built-in amplifier for direct reading of thermocouples. Z-80A microprocessor controlled. RS-232. |
| Price | $2595 (US) for standard version. $2975 with two analog outputs and 4 counter/accumulators. |

run on a Kaypro 4-84, but many other clock cards provide BCD data for date and time values.

PROCEDURES GETTIME and GETDATE are specific to the Kaypro clock card as written, but could be changed to work with many others. If you already have a routine written to get the time and date from your system clock, just insert them in place of these. To convert these procedures for another system, refer to TASK 8 in Program Two, where the function of each port is commented in more detail.

FUNCTIONS GETDATA and ANALOGIN are used together to send a command to the ADC-1 and to get a complete byte back from it. I write these as separate functions, because GETDATA is also needed with functions that read the digital inputs and send commands to the BSR transmitter of the ADC-1 system. They could, of course, be combined into a single function. To see how they work, look first at ANALOGIN. Since it calls GETDATA, ANALOGIN must come second in the listing.

ANALOGIN is a function to which a channel number is passed as a parameter, and which returns an integer value (the reading from that channel) to the calling routine. Because the ADC-1 provides 12-bit resolution, the answer can't be obtained in one 8-bit byte, and two are used. HighByte contains the 4 MSB (most significant bits) of the data plus other bits that indicate polarity, whether the A/D conversion is finished, and an over-range indication. LowByte contains the 8 LSB (least significant bits) of the analog value at the channel. Now it's time to take a look at GETDATA...

GETDATA also uses a parameter, the byte to be sent as a command to the ADC-1. The answer byte is the value that is stored in GETDATA, as if it were a variable. This function first sends the ADC-1 its command byte through the serial port (port 4 for the Kaypro), checks the status port (port 6 for the Kaypro), and compares the status value to the Mask (bit 1 for the Kaypro). The status check is repeated until a complete byte has been received, then GETDATA is assigned the value read from the serial port. Needless to say, to use this function with other computers, replace the port and mask values used by the Kaypro with those required for your machine. Now, back to ANALOGIN...

| | |
|---|---|
| Model | PL-1000 Measurement & Control System |
| Manufacturer | Elexor Associates<br>PO Box 246<br>Morris Plains, NJ 07950<br>(201) 299-1615 |
| Features | 16 analog inputs of 12-bit resolution plus sign, 16 digital inputs, 16 digital outputs. CMOS for low power use. Standalone operation possible with built-in BASIC interpretor and 8K RAM. Unit can accomodate up to 2 I/O boards (optional) for additional channels. RS-232. |
| Price | $899 basic system. Expansion boards $329 - $499. Many system options. |
| Model | Q-3024 Remote Data Collection System |
| Manufacturer | Quasitronics<br>211 Vandale Drive<br>Houston, PA 15342<br>(800) 245-4192 |
| Features | 2 single-ended analog inputs, 4 digital outputs. A/D resolution 1 in 5000 or 1 in 20000 using BCD digits, sample rate is 7.5/sec. RS-232. |
| Price | $495 |
| Model | WB-31 "White Box" Interface |
| Manufacturer | Omega Engineering, Inc.<br>One Omega Drive<br>Stamford, CT 06907<br>(203) 324-FLOW |
| Features | 2 analog inputs of 12-bit resolution, 4 digital outputs. A/D sample rate 7.5/sec. RS-232. |
| Price | $395. Other models to $2000+. Omega has a fine catalog of sensors, that is well worth requesting. |
| Model | 12232 Data Acquisition & Control System |
| Manufacturer | Starbuck Data Company<br>PO Box 24<br>Newton Lower Falls, MA 02162<br>(617) 237-7695 |
| Features | Microprocessor controlled, 8 analog inputs of 12-bit resolution, 8 digital inputs, 8 digital outputs. RAM to store 2000 data points of burst data. |
| Cost | $690. Analog 8-bit 8032 version is now $390. |

ANALOGIN sends a command byte that represents the channel number -1, which tells the ADC-1 to start digitizing that channel. (The terminal strips are labeled 1-16, and people tend to think that way too, but the ADC-1 is expecting a value from 0-15.) The next step is to assign HighByte the value returned from the ADC-1 when the command 161 is sent to it. The bit that represents the binary number 128 is checked until it is a 0, indicating that digitizing is complete. Then it's safe to read the LowByte, with a command of 145.

Only the 4 LSB of HighByte contain data (the rest are status), so HighByte is ANDed with 15 to result in the variable MaskedHighByte. From here, it's a simple matter of multiplying the masked high byte by 256 and adding the low byte. One final check is made to determine whether the sign flag was cleared – if HighByte AND 16 0, then the value is a negative one and the appropriate sign is added.

The procedure to calculate the square of the sample mean is used with the following procedure that calculates the sample standard deviation. All the analog readings that have been taken were stored in the array Buffer[1..10000]. For the number of readings that were requested, these elements are summed, then divided by the number of readings to provide the arithmetic mean. Then this mean is squared.

The standard deviation procedure is straightforward. It squares each individual reading, sums the squares and uses a very standard statistical formula to determine the standard deviation of the entire sample.

The program main logic should be easy to follow, as the variables, procedure names, and function names are self-describing. Basically, the program asks the user how many samples to read, checks the time, reads the samples and stores them to the Buffer[ ] array, then checks the time once more. The number of seconds required to do the reading is divided by the number of readings made to determine how many samples per second could be read.

The user can decide to display all of the samples to the screen for a visual check, or only a few of them. Pressing RETURN will display them all. The number of them that were displayed to the screen will be used to calculate the standard deviation. Another RETURN

## Program 1

```
10 '   LOGGER.BAS IS A PROGRAM TO READ UP TO 7 CHANNELS FROM
11 '   AN ADC-1 DATA ACQUISITION AND CONTROL SYSTEM, AND LOG
12 '   DATA TO A DISK FILE.  THIS SOURCE CODE MUST BE COMPILED
13 '   WITH THE MTBASIC COMPILER, AVAILABLE FROM SOFTAID, INC.
14 '
16 '                    JERRY HOUSTON
17 '            REMOTE MEASUREMENT SYSTEMS, INC.
20 '                 (206) 328-2255
21 '
22 '
100 INTEGER I,J,K,R,C                                  ' UTILITY COUNTERS
110 INTEGER YEAR,MONTH,DAY,HOUR,MINUTE,SECOND          ' DATE AND TIME VALUES
120 INTEGER HBYTE,HMASK,LBYTE,CHARIN,RESULT            ' ANALOG READ VALUES
130 INTEGER TIKS1,TIKS2,TIKS3,TIKS4,TKS(7)             ' TIMING VALUES
131 INTEGER TIKS5,TIKS6,TIKS7                          ' MORE TIMING VALUES
132 INTEGER CHANNEL(7)                                 ' FLAG - CHANNEL USED?
135 INTEGER CONVERT,X                                  ' FUNCTION NAMES DECLARED
136 INTEGER SENSOR                                     ' LAST CHANNEL READ
140 STRING  FNAME$(14),LABEL$(20,7)                    ' FILENAME, CHANNEL LABELS
145 STRING  A$                                         ' Y/N INPUT FROM CONSOLE
146 STRING CON$(3),COFF$(3)                            ' CURSOR ON, OFF (KAYPRO)
150 '
160 '   THIS FUNCTION CONVERTS BCD NUMBERS FROM CLOCK CARD TO
170 '   ASCII DECIMAL REPRESENTATION FOR USE BY SCREEN AND FILE
180 '
190 DEF CONVERT(X)
200 CONVERT = (( X / 16 ) * 10) + (X -(16 * (X/16)))
210 FNEND
215 '
216 FOR I = 1 TO 7
217 CHANNEL(I) = 0                                     ' INITIALIZE CHANNEL FLAGS
218 NEXT I
220 '
221 '   THESE ASSIGNMENTS CREATE TWO STRINGS THAT WILL TURN THE
222 '   KAYPRO CURSOR ON OR OFF, PROVIDING CLEARER DISPLAY ON THE CRT
223 '
225 CON$=CONCAT$(CHR$(27),"B"): CON$=CONCAT$(CON$,"4")
230 COFF$=CONCAT$(CHR$(27),"C"): COFF$=CONCAT$(COFF$,"4")
231 '
234 '
235 '   MODULE TO SELECT CHANNELS TO BE READ, LABELS FOR THEM, AND
240 '   FILENAME TO USE FOR STORING DATA TO DISK DRIVE
250 '
251 OUT 0,14                                           ' SET BAUD TO 9600
252 ERASE                                              ' CLEAR SCREEN
253 WSELECT 0                                          ' SELECT A WINDOW TO USE
255 WINDOW 0,0,23,78:WFRAME "_","|":WINDOW 1,1,22,77
260 WCLEAR                                             ' CLEAR SCREEN
265 PRINT
270 PRINT "     ENTER FILE NAME TO BE USED TO STORE DATA: ";
280 INPUT FNAME$
290 PRINT: PRINT "          ARE YOU SATISFIED WITH FILENAME (Y,N): ";
300 INPUT A$: IF A$ <> "Y" THEN 260
310 WCLEAR                                             ' CLEAR SCREEN
315 PRINT
320 FOR I = 1 TO 7                                     ' GET CHANNEL DATA
325 TKS(I) = 0                                         ' INITIALIZE TICS TO 0
330 PRINT "     SCHEDULE ANALOG INPUT 0";I;"? (Y,N) ";
340 INPUT A$
350 IF A$ = "N"  THEN 355
351 PRINT "          LABEL FOR INPUT 0";I;": ";:INPUT LABEL$(I)
352 PRINT "          TIMING TICS: ";:INPUT TKS(I)
353 CHANNEL(I) = 1                                     ' TURN ON CHANNEL FLAG
354 PRINT
355 IF TKS(I) = 0 THEN TKS(I) = 31000                  ' IF NOT USED, CHECK SELDOM
360 NEXT I
364 TIKS1 = TKS(1) : TIKS2 = TKS(2) : TIKS3 = TKS(3) : TIKS4 = TKS(4)
365 TIKS5 = TKS(5) : TIKS6 = TKS(6) : TIKS7 = TKS(7)
366 PRINT
370 PRINT: PRINT "       ARE THE SELECTIONS AS YOU WANT THEM? (Y,N) ";:INPUT A$
375 IF A$ <> "Y" THEN 310
380 OPEN 1,1,FNAME$                                    ' OPEN FILE FOR OUTPUT
381 PRINT COFF$:PRINT"          SAMPLING WILL BEGIN WHEN YOU PRESS <RETURN> ";
    :INPUT A$
400 '
410 '
420 '    AT THIS POINT, UP TO 7 CHANNELS HAVE BEEN SELECTED FOR
430 '    MONITORING, AND SAMPLING WILL BEGIN WITH ANY INPUT FOR A$
440 '
450 WSELECT 0
452 WCLEAR: CURSOR 1,0: PRINT"                CURRENT DATE AND TIME:"
453 CURSOR 3,0: PRINT"        (PRESS <ESC> TO END PROGRAM AND CLOSE DATA FILE)"
455 CURSOR 4,0
456 PRINT
460 PRINT "    SAMPLING      CHANNEL          INPUT                        VALUE"
470 PRINT "      FLAG        NUMBER           LABEL          TICS          READ"
480 PRINT "     ----------------------------------------------------------------"
    :PRINT
490 FOR I = 1 TO 7
500 IF CHANNEL(I) = 0 THEN PRINT "                              *** UNSELECTED *** "
    : PRINT: GOTO 520
510 FPRINT "X23,I1,X8,B20,X2,I5,X7,I4,Z",I,LABEL$(I),TKS(I): PRINT
520 NEXT I
525 '
530 '
531 RUN 1,TIKS1
532 RUN 2,TIKS2
533 RUN 3,TIKS3
534 RUN 4,TIKS4
535 RUN 5,TIKS5
536 RUN 6,TIKS6
537 RUN 7,TIKS7
```

will prepare the program for another test, and telling it to read 0 samples will quit.

**Program Two (LOGGER.BAS)**

The second program is written in MTBASIC® source code, and to run it must be compiled. Because of its size, it won't compile to memory, and you will need to use the DISK COMPILE option of MTBASIC to produce an executable .COM file from it. The multitasking feature of MTBASIC makes it easy to write a general-purpose analog data logging program like this. One of the nine available tasks is used to read the system clock and another is used to log data to the disk file, so that leaves seven analog channels that can be read, each according to its own schedule.

Many people are not yet familiar with using MTBASIC (though it has been the subject of a review in The Computer Journal), so I'll mention as I go some of the things a new user will need to look out for.

Praises to its inventor for providing some Pascal-like features in MTBASIC! Variables are defined before use, as in lines 100-146. Multi-line functions are

```
538 RUN 8,60
539 RUN 9,10
540 J = KEY                                    ' CHECK FOR KEYPRESSED
541 IF J = 27 THEN 543                         ' IF <ESC> THEN QUIT
542 GOTO 540                                   ' OTHERWISE, CONTINUE
543 CLOSE 1                                    ' CLOSE OUTPUT FILE
544 PRINT CON0: ERASE                          ' CURSOR BACK ON,CLEAR SCREEN
545 STOP                                       ' END PROGRAM
546 '
550 '   THIS PROGRAM USES TASKS 1 TO 7 TO READ ADC-1 CHANNELS
560 '   1 TO 7.  TASK 8 IS USED TO READ THE TIME AND DATE FROM
570 '   THE ONBOARD CLOCK CARD (MODIFY ACCORDING TO YOUR CARD
580 '   IF IT IS NOT THE STANDARD KAYPRO CARD), AND TASK 9 IS
590 '   USED TO WRITE DATA TO THE DISK BUFFER WHEN CHANNELS ARE
600 '   READ.  THE DISK FILE IS SEQUENTIAL, AND IT WILL BE UP
610 '   TO AN OUTPUT PROGRAM (CRT OR REPORT) TO FORMAT THE DATA
620 '   ACCORDING TO CHANNEL NUMBERS, TIMES, OR OTHER PARAMETERS.
630 '
640 TASK 1                                     ' DEFINE BEGINNING OF TASK 1
650 IF CHANNEL(1) = 0 THEN 870                 ' SKIP IF CHANNEL NOT SELECTED
655 INTOFF                                     ' INTERRUPTS OFF
660 WSELECT 0                                  ' SELECT ORIGINAL WINDOW
670 CURSOR 9,3                                 ' SET CURSOR TO PRINT MSG
680 PRINT ">> READING >>"                      ' PRINT READING MESSAGE
690 OUT 4,0                                    ' START A/D ON CHANNEL 1
700 IF BAND(INP(6),1) = 0 THEN 700             ' WAIT FOR REPLY FROM ADC-1
710 CHARIN = INP(4)                            ' GET (UNUSED) REPLY
720 OUT 4,161                                  ' ASK FOR HIGH BYTE OF DATA
730 IF BAND(INP(6),1) = 0 THEN 730             ' WAIT FOR REPLY
740 HBYTE = INP(4)                             ' GET HIGH BYTE OF DATA
750 IF BAND(HBYTE,128) THEN 720                ' NOT FINISHED A/D YET
760 OUT 4,145                                  ' ASK FOR LOW BYTE OF DATA
770 IF BAND(INP(6),1) = 0 THEN 770             ' WAIT UNTIL RECEIVED
780 LBYTE = INP(4)                             ' GET LOW BYTE OF DATA
790 HMASK = BAND(HBYTE,15)                     ' MASK FOR 4 DATA BITS USED
800 RESULT = LBYTE + 256 * HMASK               ' COMBINE INTO TOTAL VALUE
810 IF BAND(HBYTE,16) = 0 THEN RESULT = - RESULT ' CHECK NEGATIVE FLAG
820 CURSOR 9,3                                 ' SET CURSOR TO ERASE MSG
830 PRINT "                    "               ' ERASE READING MESSAGE
840 CURSOR 9,66                                ' SET CURSOR TO PRINT VALUE
850 FPRINT "I5,I",RESULT                       ' PRINT VALUE READ
860 K = 1: SENSOR = 1                          ' DATA-TO-DISK FLAG
865 INTON                                      ' INTERRUPTS BACK ON
870 EXIT                                       ' END OF TASK 1
880 '
890 '
900 TASK 2                                     ' DEFINE BEGINNING OF TASK 2
910 IF CHANNEL(2) = 0 THEN 1130                ' SKIP IF CHANNEL NOT SELECTED
915 INTOFF                                     ' INTERRUPTS OFF
```


*Sidekick* for CP/M!

**Write-Hand-Man**

Desk Accessories for CP/M

**NEW! Now with automatic screen refresh!**

Suspend CP/M applications such as WordStar, dBase, and SuperCalc, with a single keystroke and look up phone numbers, edit a notepad, make appointments, view files and directories, communicate with other computers, and do simple arithmetic. Return to undisturbed application! All made possible by **Write-Hand-Man.** Ready to run after a simple terminal configuration! No installation required.

Don't be put down by 16 bit computer owners. Now any CP/M 2.2 machine can have the power of *Sidekick.*

Bonus! User extendable! Add your own applications.

**$49.95** plus tax (California residents), shipping included! Volume and dealer discounts.

Available on IBM 8 inch and Northstar 5 inch disks. Other 5 inch formats available with a $5.00 handling charge. CP/M 2.2 required; CP/M 3 not supported.

COD or checks ok, no credit cards

**Poor Person Software**

3721 Starr King Circle

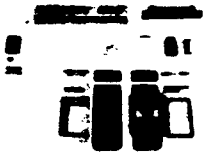
Palo Alto, CA 94306

tel 415-493-3735

Write-Hand-Man trademark of Poor Person Software, CP/M trademark of Digital Research, *Sidekick* trademark of Borland International, dBase trademark of Ashton-Tate, WordStar trademark of Micropro, SuperCalc a trademark of Sorcim.



**APROTEK 1000™ EPROM PROGRAMMER**



TECHNICAL BREAKTHROUGH NOW ALLOWS A PRICE BREAKTHROUGH

**only $250.00**

***A SIMPLE, INEXPENSIVE SOLUTION TO PROGRAMMING EPROMS***

The ***APROTEK 1000*** can program 5 volt, 25XX series through 2564, 27XX series through 27256 and 68XX devices plus any CMOS versions of the above types. Included with each programmer is a personality module of your choice (others are only $10.00 ea. when purchased with ***APROTEK 1000***). Later, you may require future modules at only $15.00 ea., postage paid. Available personality modules: PM2716, PM2732, PM2732A, PM2764, PM2764A, PM27128, PM27256, PM2532, PM2564, PM68764 (includes 68766). (Please specify modules by these numbers)

***APROTEK 1000*** comes complete with a menu driven BASIC driver programmer listing which allows READ, WRITE, COPY, and VERIFY with Checksum. Easily adapted for use with IBM, Apple, Kaypro, and other microcomputers with a RS-232 port. Also included is a menu driven CPM assembly language driver listing with Z-80 (DART) and 8080 (8251) I/O port examples. Interface is a simple 3-wire RS-232C with a female DB-25 connector. A handshake character is sent by the programmer after programming each byte. The interface is switch selectable at the following 6 baud rates: 300, 1.2k, 2.4k, 4.8k, 9.6k and 19.2k baud. Data format for programming is "absolute code". (i.e., it will program exactly what it is sent starting at EPROM address 0). Other standard downloading formats are easily converted to absolute (object) code.

The ***APROTEK 1000*** is truly universal. It comes standard at 117 VAC 50 60 HZ and may be internally jumpered for 220-240 VAC 50 60 AZ. FCC verification (CLASS B) has been obtained for the ***APROTEK 1000***.

***APROTEK 1000 is covered by a 1 year parts and labor warranty.***

***FINALLY — A Simple, Inexpensive Solution To Erasing EPROMS***

***APROTEK-200™ EPROM ERASER***
Simply insert one or two EPROMS and switch ON. In about 10 minutes, you switch OFF and are ready to reprogram.
***APROTEK-200™*** only ***$45.00.***

***APROTEK-300™*** only ***$60.00.***
This eraser is identical to ***APROTEK-200™*** but has a built-in timer so that the ultraviolet lamp automatically turns off in 10 minutes, eliminating any risk of overexposure damage to your EPROMS.
***APROTEK-300™*** only ***$60.00.***

**APROPOS TECHNOLOGY**

**1071-A Avenida Acaso, Camarillo, CA 93010**

**CALL OUR TOLL FREE ORDER LINES TODAY:**

**1-(800) 962-5800 USA or 1-(800) 962-3800 CALIFORNIA**

**TECHNICAL INFORMATION: 1-(805) 482-3604**

Add Shipping Per Item: $3.00 Cont. U.S. $6.00 CAN. Mexico, HI, AK, UPS Blue

```
920 WSELECT 0                                   ' SELECT ORIGINAL WINDOW
930 CURSOR 11,3                                 ' SET CURSOR TO PRINT MSG
940 PRINT ">> READING >>";                      ' PRINT READING MESSAGE
950 OUT 4,1                                     ' START A/D ON CHANNEL 2
960 IF BAND(INP(6),1) = 0 THEN 960              ' WAIT FOR REPLY FROM ADC-1
970 CHARIN = INP(4)                             ' GET (UNUSED) REPLY
980 OUT 4,161                                   ' ASK FOR HIGH BYTE OF DATA
990 IF BAND(INP(6),1) = 0 THEN 990              ' WAIT FOR REPLY
1000 HBYTE = INP(4)                             ' GET HIGH BYTE OF DATA
1010 IF BAND(HBYTE,128) THEN 980                ' NOT FINISHED A/D YET
1020 OUT 4,145                                  ' ASK FOR LOW BYTE OF DATA
1030 IF BAND(INP(6),1) = 0 THEN 1030            ' WAIT UNTIL RECEIVED
1040 LBYTE = INP(4)                             ' GET LOW BYTE OF DATA
1050 HMASK = BAND(HBYTE,15)                     ' MASK FOR 4 DATA BITS USED
1060 RESULT = LBYTE + 256 * HMASK               ' COMBINE INTO TOTAL VALUE
1070 IF BAND(HBYTE,16) = 0 THEN RESULT = - RESULT' CHECK NEGATIVE FLAG
1080 CURSOR 11,3                                ' SET CURSOR TO ERASE MSG
1090 PRINT "                "                   ' ERASE READING MESSAGE
1100 CURSOR 11,66                               ' SET CURSOR TO PRINT VALUE
1110 FPRINT "I5,Z",RESULT                       ' PRINT VALUE READ
1120 K = 1 : SENSOR = 2                         ' DATA-TO-DISK FLAG
1125 INTON                                      ' INTERRUPTS BACK ON
1130 EXIT                                       ' END OF TASK 2
1140 '
1150 '
1160 TASK 3                                     ' DEFINE BEGINNING OF TASK 3
1170 IF CHANNEL(3) = 0 THEN 1390                ' SKIP IF CHANNEL NOT SELECTED
1175 INTOFF                                     ' INTERRUPTS OFF
1180 WSELECT 0                                  ' SELECT ORIGINAL WINDOW
1190 CURSOR 13,3                                ' SET CURSOR TO PRINT MSG
1200 PRINT ">> READING >>";                     ' PRINT READING MESSAGE
1210 OUT 4,2                                    ' START A/D ON CHANNEL 3
1220 IF BAND(INP(6),1) = 0 THEN 1220            ' WAIT FOR REPLY FROM ADC-1
1230 CHARIN = INP(4)                            ' GET (UNUSED) REPLY
1240 OUT 4,161                                  ' ASK FOR HIGH BYTE OF DATA
1250 IF BAND(INP(6),1) = 0 THEN 1250            ' WAIT FOR REPLY
1260 HBYTE = INP(4)                             ' GET HIGH BYTE OF DATA
1270 IF BAND(HBYTE,128) THEN 1240               ' NOT FINISHED A/D YET
1280 OUT 4,145                                  ' ASK FOR LOW BYTE OF DATA
1290 IF BAND(INP(6),1) = 0 THEN 1290            ' WAIT UNTIL RECEIVED
1300 LBYTE = INP(4)                             ' GET LOW BYTE OF DATA
1310 HMASK = BAND(HBYTE,15)                     ' MASK FOR 4 DATA BITS USED
1320 RESULT = LBYTE + 256 * HMASK               ' COMBINE INTO TOTAL VALUE
1330 IF BAND(HBYTE,16) = 0 THEN RESULT = - RESULT' CHECK NEGATIVE FLAG
1340 CURSOR 13,3                                ' SET CURSOR TO ERASE MSG
1350 PRINT "                "                   ' ERASE READING MESSAGE
1360 CURSOR 13,66                               ' SET CURSOR TO PRINT VALUE
1370 FPRINT "I5,Z",RESULT                       ' PRINT VALUE READ
1380 K = 1: SENSOR = 3                          ' DATA-TO-DISK FLAG
1385 INTON                                      ' INTERRUPTS BACK ON
1390 EXIT                                       ' END OF TASK 3
1400 '
1410 '
1420 TASK 4                                     ' DEFINE BEGINNING OF TASK 4
1430 IF CHANNEL(4) = 0 THEN 1650                ' SKIP IF CHANNEL NOT SELECTED
1435 INTOFF                                     ' INTERRUPTS OFF
1440 WSELECT 0                                  ' SELECT ORIGINAL WINDOW
1450 CURSOR 15,3                                ' SET CURSOR TO PRINT MSG
1460 PRINT ">> READING >>";                     ' PRINT READING MESSAGE
1470 OUT 4,3                                    ' START A/D ON CHANNEL 4
1480 IF BAND(INP(6),1) = 0 THEN 1480            ' WAIT FOR REPLY FROM ADC-1
1490 CHARIN = INP(4)                            ' GET (UNUSED) REPLY
1500 OUT 4,161                                  ' ASK FOR HIGH BYTE OF DATA
1510 IF BAND(INP(6),1) = 0 THEN 1510            ' WAIT FOR REPLY
1520 HBYTE = INP(4)                             ' GET HIGH BYTE OF DATA
1530 IF BAND(HBYTE,128) THEN 1500               ' NOT FINISHED A/D YET
1540 OUT 4,145                                  ' ASK FOR LOW BYTE OF DATA
1550 IF BAND(INP(6),1) = 0 THEN 1550            ' WAIT UNTIL RECEIVED
1560 LBYTE = INP(4)                             ' GET LOW BYTE OF DATA
1570 HMASK = BAND(HBYTE,15)                     ' MASK FOR 4 DATA BITS USED
1580 RESULT = LBYTE + 256 * HMASK               ' COMBINE INTO TOTAL VALUE
1590 IF BAND(HBYTE,16) = 0 THEN RESULT = - RESULT' CHECK NEGATIVE FLAG
1600 CURSOR 15,3                                ' SET CURSOR TO ERASE MSG
1610 PRINT "                "                   ' ERASE READING MESSAGE
1620 CURSOR 15,66                               ' SET CURSOR TO PRINT VALUE
1630 FPRINT "I5,Z",RESULT                       ' PRINT VALUE READ
1640 K = 1: SENSOR = 4                          ' DATA-TO-DISK TASK FLAG
1645 INTON                                      ' INTERRUPTS BACK ON
1650 EXIT                                       ' END OF TASK 4
1660 '
1670 '
1680 TASK 5                                     ' DEFINE BEGINNING OF TASK 5
1690 IF CHANNEL(5) = 0 THEN 1930                ' SKIP IF CHANNEL NOT SELECTED
1700 INTOFF                                     ' INTERRUPTS OFF
1710 WSELECT 0                                  ' SELECT ORIGINAL WINDOW
1720 CURSOR 17,3                                ' SET CURSOR TO PRINT MSG
1730 PRINT ">> READING >>";                     ' PRINT READING MESSAGE
1740 OUT 4,4                                    ' START A/D ON CHANNEL 5
1750 IF BAND(INP(6),1) = 0 THEN 1750            ' WAIT FOR REPLY FROM ADC-1
1760 CHARIN = INP(4)                            ' GET (UNUSED) REPLY
1770 OUT 4,161                                  ' ASK FOR HIGH BYTE OF DATA
1780 IF BAND(INP(6),1) = 0 THEN 1780            ' WAIT FOR REPLY
1790 HBYTE = INP(4)                             ' GET HIGH BYTE OF DATA
1800 IF BAND(HBYTE,128) THEN 1770               ' NOT FINISHED A/D YET
1810 OUT 4,145                                  ' ASK FOR LOW BYTE OF DATA
1820 IF BAND(INP(6),1) = 0 THEN 1820            ' WAIT UNTIL RECEIVED
1830 LBYTE = INP(4)                             ' GET LOW BYTE OF DATA
1840 HMASK = BAND(HBYTE,15)                     ' MASK FOR 4 DATA BITS USED
1850 RESULT = LBYTE + 256 * HMASK               ' COMBINE INTO TOTAL VALUE
1860 IF BAND(HBYTE,16) = 0 THEN RESULT = - RESULT' CHECK NEGATIVE FLAG
1870 CURSOR 17,3                                ' SET CURSOR TO ERASE MSG
1880 PRINT "                "                   ' ERASE READING MESSAGE
1890 CURSOR 17,66                               ' SET CURSOR TO PRINT VALUE
1900 FPRINT "I5,Z",RESULT                       ' PRINT VALUE READ
```

supported in the latest version of MT-BASIC, such as the one at 190-210 which converts BCD numbers into ASCII numbers (as did FUNCTION ACD in Program One). Lines 216-218 initialize an array of channel flags to zero, which flags will later be used to indicate which channel has just been read.

MTBASIC supports multiple instructions on a physical line, using semicolon separators, but using short lines like these leaves plenty of room for good documentation, and I'll recommend easier-to-follow code every time. Significantly, the compiler does NOT translate the comments, so there is no efficiency advantage to be gained by leaving them out, and no real excuse to pack as many instructions on as few lines as possible.

The statements at 225-230 create two strings that are the escape sequences to turn off the Kaypro cursor. These might need to be translated for another computer, or left out entirely if that feature is not supported. (Notice that it isn't possible to concatenate strings using the operator, as in most versions of BASIC. The CONCAT$() function must be used each time one string is added to another.) Turning off the cursor provides a cleaner screen display, and is recommended.

CAUTION—Remember to turn the cursor ON at the end of the program, or you may have to re-boot to get it back!

The actual purpose of the rest of the code is pretty obvious from the comments, so I'll concentrate on explaining the workings of MTBASIC, where it differs substantially from other versions. The first seven tasks are designed to read the analog channels of the ADC-1 system, and would certainly have to be modified to work with a different interface. They are so similar (except for variables that indicate the channel that was read and cursor positioning) that I'll explain only the first task, then go on to tasks 8 and 9, which read the clock card and write to disk.

Task 0 (zero) is the main logic of the program, and it actually includes all the code up to line 545, the STOP command. Line 251 uses the familiar OUT statement to send a value of 14 to port 0, setting the Kaypro serial port to a baud rate of 9600. Other computers accomplish this with different port numbers and values, or with an OPEN statement.

The ERASE statement in line 252 clears the screen. Since WINDOW 0 (used throughout the program so that the CURSOR positioning statement will work) includes the entire screen, the WCLEAR statements (as in 260) also clear the whole screen. In other cases, they could be used to clear just a single window while leaving the rest of the screen alone.

Line 253 selects WINDOW 0 for use until another WSELECT statement is given. Line 255 establishes that WINDOW 0 will exist from screen lines 0 to 23 and from screen columns 0 to 78. WFRAME defines the characters to be used to outline the screen, the first character used for the horizontal lines (top and bottom) and the second used for the vertical lines (sides). After the window is properly framed, it is redefined to be one character smaller in each direction, so the framing characters are protected from overwriting.

The next lines just print prompts to the screen, and ask the user whether to schedule each of the first seven analog input channels. If the answer for a channel is "yes", then the user is asked

```
1910 K = 1: SENSOR = 5                          ' DATA-TO-DISK TASK FLAG
1920 INTON                                      ' INTERRUPTS BACK ON
1930 EXIT                                       ' END OF TASK 5
1940 '
1950 '
1960 TASK 6                                     ' DEFINE BEGINNING OF TASK 6
1970 IF CHANNEL(6) = 0 THEN 2210                ' SKIP IF CHANNEL NOT SELECTED
1980 INTOFF                                     ' INTERRUPTS OFF
1990 WSELECT 0                                  ' SELECT ORIGINAL WINDOW
2000 CURSOR 19,3                                ' SET CURSOR TO PRINT MSG
2010 PRINT ">> READING >>";                     ' PRINT READING MESSAGE
2020 OUT 4,5                                    ' START A/D ON CHANNEL 6
2030 IF BAND(INP(6),1) = 0 THEN 2030            ' WAIT FOR REPLY FROM ADC-1
2040 CHARIN = INP(4)                            ' GET (UNUSED) REPLY
2050 OUT 4,161                                  ' ASK FOR HIGH BYTE OF DATA
2060 IF BAND(INP(6),1) = 0 THEN 2060            ' WAIT FOR REPLY
2070 HBYTE = INP(4)                             ' GET HIGH BYTE OF DATA
2080 IF BAND(HBYTE,128) THEN 2050               ' NOT FINISHED A/D YET
2090 OUT 4,145                                  ' ASK FOR LOW BYTE OF DATA
2100 IF BAND(INP(6),1) = 0 THEN 2100            ' WAIT UNTIL RECEIVED
2110 LBYTE = INP(4)                             ' GET LOW BYTE OF DATA
2120 HMASK = BAND(HBYTE,15)                     ' MASK FOR 4 DATA BITS USED
2130 RESULT = LBYTE + 256 * HMASK               ' COMBINE INTO TOTAL VALUE
2140 IF BAND(HBYTE,16) = 0 THEN RESULT = - RESULT' CHECK NEGATIVE FLAG
2150 CURSOR 19,3                                ' SET CURSOR TO ERASE MSG
2160 PRINT "                "                   ' ERASE READING MESSAGE
2170 CURSOR 19,66                               ' SET CURSOR TO PRINT VALUE
2180 FPRINT "I5,I",RESULT                       ' PRINT VALUE READ
2190 K = 1: SENSOR = 6                          ' DATA-TO-DISK TASK FLAG
2200 INTON                                      ' INTERRUPTS BACK ON
2210 EXIT                                       ' END OF TASK 6
2220 '
2230 '
2240 TASK 7                                     ' DEFINE BEGINNING OF TASK 7
2250 IF CHANNEL(7) = 0 THEN 2490                ' SKIP IF CHANNEL NOT SELECTED
2260 INTOFF                                     ' INTERRUPTS OFF
2270 WSELECT 0                                  ' SELECT ORIGINAL WINDOW
2280 CURSOR 21,3                                ' SET CURSOR TO PRINT MSG
2290 PRINT ">> READING >>";                     ' PRINT READING MESSAGE
2300 OUT 4,6                                    ' START A/D ON CHANNEL 7
2310 IF BAND(INP(6),1) = 0 THEN 2310            ' WAIT FOR REPLY FROM ADC-1
2320 CHARIN = INP(4)                            ' GET (UNUSED) REPLY
2330 OUT 4,161                                  ' ASK FOR HIGH BYTE OF DATA
2340 IF BAND(INP(6),1) = 0 THEN 2340            ' WAIT FOR REPLY
2350 HBYTE = INP(4)                             ' GET HIGH BYTE OF DATA
2360 IF BAND(HBYTE,128) THEN 2330               ' NOT FINISHED A/D YET
2370 OUT 4,145                                  ' ASK FOR LOW BYTE OF DATA
2380 IF BAND(INP(6),1) = 0 THEN 2380            ' WAIT UNTIL RECEIVED
2390 LBYTE = INP(4)                             ' GET LOW BYTE OF DATA
2400 HMASK = BAND(HBYTE,15)                     ' MASK FOR 4 DATA BITS USED
2410 RESULT = LBYTE + 256 * HMASK               ' COMBINE INTO TOTAL VALUE
2420 IF BAND(HBYTE,16) = 0 THEN RESULT = - RESULT' CHECK NEGATIVE FLAG
2430 CURSOR 21,3                                ' SET CURSOR TO ERASE MSG
2440 PRINT "                "                   ' ERASE READING MESSAGE
2450 CURSOR 21,66                               ' SET CURSOR TO PRINT VALUE
2460 FPRINT "I5,I",RESULT                       ' PRINT VALUE READ
2470 K = 1: SENSOR = 7                          ' DATA-TO-DISK TASK FLAG
2480 INTON                                      ' INTERRUPTS BACK ON
2490 EXIT                                       ' END OF TASK 7
2500 '
2510 '
2520 TASK 8                                     ' DEFINE BEGINNING OF TASK 8
2525 INTOFF                                     ' INTERRUPTS OFF DURING TASK
2530 WSELECT 0                                  ' SELECT FULL SCREEN WINDOW
2540 OUT 34,15                                  ' WAKE UP CLOCK CARD
2550 OUT 32,9                                   ' ASK FOR YEAR
2560 YEAR = CONVERT(INP(36))                    ' GET YEAR
2570 OUT 32,7                                   ' ASK FOR MONTH
2580 MONTH = CONVERT(INP(36))                   ' GET MONTH
2590 OUT 32,6                                   ' ASK FOR DAY
2600 DAY = CONVERT(INP(36))                     ' GET DAY
2610 OUT 32,4                                   ' ASK FOR HOUR
2620 HOUR = CONVERT(INP(36))                    ' GET HOUR
2630 OUT 32,3                                   ' ASK FOR MINUTE
2640 MINUTE = CONVERT(INP(36))                  ' GET MINUTE
2650 OUT 32,2                                   ' ASK FOR SECONDS
2660 SECOND = CONVERT(INP(36))                  ' GET SECONDS
2670 CURSOR 1,40
2680 FPRINT "I2,S1,I2,S1,I2,X2,I2,S1,I2,S1,I2,I2",MONTH,"/",DAY,"/",YEAR,
     HOUR,":",MINUTE,":",SECOND
2685 INTON                                      ' INTERRUPTS BACK ON
2690 EXIT                                       ' END OF TASK 8
2700 '
2710 '
2715 TASK 9                                     ' DEFINE BEGINNING OF TASK 9
2720 INTOFF                                     ' WHILE WRITING FILE NO INT
2740 IF K = 0 THEN 2790                         ' WRITE ONLY IF A CHANNEL READ
2750 FILE 1                                     ' DIRECT OUTPUT TO FILE
2760 FPRINT "I2,X3,I2,X1,I2,X1,I2,X2,I2,X1,I2,X1,I2,X2,I4",SENSOR,YEAR,
     MONTH,DAY,HOUR,MINUTE,SECOND,RESULT
2770 FILE 0                                     ' DESELECT FILE
2780 K = 0                                      ' INDICATE RECORD WRITTEN
2790 INTON                                      ' INTERRUPTS BACK ON
2800 EXIT                                       ' END OF TASK 9
END
```


## "BMON"

### Software In-Circuit Emulator

Links your CP/M computer with any Z80 based computer or controller that you may develop. All that is needed is BMON, 8K of ROM space, and a handshakeable bi-directable I/O port (either RS232 or Parallel).

**Features:**

—Full program development debugger with Breakpoints, Snaps, Stops, & Waits.

—Single Step program execution.

—Download file from CP/M system to development RAM.

—Upload Memory from development RAM to CP/M disk.

—Two versions: Master BMON runs in your CP/M system. Slave BMON runs in your target system.

Note: Requires Microsoft's M80 & L80 assembler & linker to setup Slave BMON.

*8" SSSD Disk containing Master BMON, Slave BMON, CONSOL, BMONIO, CONSOLIO, and Users Manual .....$49.95*

Shipped Via prepaid UPS
—No COD or P.O. Box—
Check or Money Order to:

**Barnes Research & Development**
750 W. Ventura St.
Altadena, CA 91101
**(818) 794-1244**

*CP/M is a trademark of Digital Research Inc*
*M80 & L80 are trademarks of Microsoft Inc*

## Program 2

```
PROGRAM HISPEED;

(*     This program demonstrates high speed analog data acquisition
       with an ADC-1 system.  Rather than display data at once, it
       simply stores data to memory as it is read.

       If wanted, the readings are displayed as integer numbers, and
       a calculation is made of the sample standard deviation.  In this
       way, not only the speed but also the resolution of a data
       acquisition system can be tested.                              *)

LABEL   Start, Stop;

TYPE    Str = String[8];

VAR     I,J,Value,Code,Channel : Integer;
        Time,X : String[8];
        Buffer : Array[1..10000] of Integer;
        StartSeconds, StopSeconds, Elapsed, Samples : Real;
        Element,ElementSquared :  Real;
        SampleMean, SampleSum, SumOfSquares,
        SampleMeanSquared, StandardDeviation :  Real;
        Count, Times  :  Integer;

Function ACD(X : Byte) : Byte;
(* Converts BCD byte into standard ASCII coded decimal representation *)
Begin
   ACD := ((X Div 16) * 10) + (X - ( 16 * (X Div 16)));
End;

Procedure GetTime(X : Str);
Var     Minute,Second,Hundr : Integer;
        Sec  : String[4];
        Hun  : String[2];
Begin
   Port[34] := 15;
   Port[32] := 3;
   Minute   := ACD(Port[36]);
   Port[32] := 2;
   Second   := (Minute * 60) + ACD(Port[36]);
   Str(Second,Sec);
   Port[32] := 1;
   Hundr    := ACD(Port[36]);
   Str(Hundr,Hun);
   Time     := Sec + '.' + Hun;
End;

Function GetData(Charout : Byte) : Byte;
Const  StatusPort = 6;    (* these values computer-specific *)
       SerialPort = 4;
       Mask       = 1;
Var    Status : Byte;
Begin
   Port[SerialPort] := Charout;

   Repeat
      Status := Port[StatusPort];      (* read value from status port *)
      Until ((Status and Mask) <> 0); (* check for full byte received *)
   Getdata := Port[SerialPort];        (* read value from serial port *)
End;

Function Analogin(channel : Byte) : Integer;
VAR    HighByte,LowByte,MaskedHighByte : Byte;
       Value : Integer;
Begin
   Value := GetData(Channel -1);
      Repeat
      HighByte := GetData(161);
      Until ((HighByte and 128) = 0);(* checks for digitizing done *)
   LowByte := GetData(145);
   MaskedHighByte := (HighByte and 15) ;        (* mask off needed bits *)
   Value := LowByte + (256 * MaskedHighByte) ; (* combine for 12-bit value *)
   If ((HighByte and 16) = 0) Then Value :=  - Value;
   Analogin := Value;
End;

Procedure CalculateSampleMeanSquared;
Begin
   SampleSum := 0.0;
   For I := 2 to J Do SampleSum := SampleSum + Buffer[I];
   SampleMean := SampleSum / (J - 1);
   SampleMeanSquared := Sqr(SampleMean);
End;

Procedure CalculateStandardDeviation;
Begin
   CalculateSampleMeanSquared;
   SumOfSquares := 0.0;
   For I := 2 to J Do
      Begin
         Element := Buffer[I];
         SumOfSquares := SumOfSquares + Sqr(Element);
      End;
   StandardDeviation := Sqrt(((SumOfSquares - ((J-1) * SampleMeanSquared))
                                          / (J-1)));
End;

Begin            (* Main Program Logic *)

   Port[0] := 14;                        (* sets Kaypro baud rate to 9600 *)
```

for a label to describe the input and the number of timing tics that must pass before that channel is read. You'll need to experiment with these interrupt tics to determine how to schedule the tasks. I haven't made any careful measurements, but 200 tics seems to be about a second with the Kaypro and a 4 MHz CPU. Other computers will vary greatly.

For each channel that is NOT selected, the value of TKS() is set to 31000, a big enough number so that the program won't spend time very often checking to see whether to execute that task. Line 353 turns on the channel flag for each channel that's selected—otherwise it stays 0 as it was initialized in 217.

The OPEN statement at line 380 sets aside an output buffer for the selected sequential file.

The next statement unique to MT-BASIC (well, actually it looks an awful lot like a FORTRAN format statement!) is at line 510 where it starts out FPRINT. This is a formatted print statement, and takes the place of PRINT USING. In this line the X values refer to blanks, the I values to integers, the S values to strings, and the Z at the end passes a carriage-return. With the Z at the end, the statement is like a Pascal WRITELN procedure, without the Z it's like a Pascal WRITE procedure, and doesn't move to the next screen line automatically.

The format string is followed by the variable list to be printed, in this case the channel number, a label to identify the input, and the number of tics assigned to it for timing purposes. When all seven channels have been thus identified (or labeled as "unselected" by line 500) the program is ready to begin executing.

Task 0, the main logic, contains the RUN statements that determine when each of the other tasks will be executed. These RUN statements include the task number and the number of tics to use. The tics can be supplied as an integer constant or as a simple variable, but NOT as a subscripted variable (array). For convenience, the loop at 310-360 inputs the tic assignments as the array TKS(I), but these have to be converted—one at a time—into the simple variables TIKS1 through TIKS7 at 364-365. (Note that MTBASIC is not at all tolerant of variable names that start out the same as reserved words (like TICS), so variables like TICS1

couldn't be used. It IS ok to use keywords entirely within a variable name, so something like XTICS1 would have been alright.)

That's about it for the main logic. Line 540 uses the KEY function to check for an ESCAPE key pressed (ASCII value 27), and gracefully exits the program (in that case remember to turn ON the cursor if you've turned it OFF). All that's necessary once the tasks have been listed in RUN statements is to keep Task 0 executing until you're ready to quit. I did it here by checking for a key pressed, but the same thing can be accomplished by using a line that does a GOTO to itself. I think it makes more sense to provide an exit from the program, so I just keep checking for an input that signifies it's time to stop.

As mentioned before, the first seven tasks here are very similar. They start with a TASK statement like the one in line 640 and end with an EXIT statement like the one at 870.

Because these seven tasks all acquire data from the same peripheral, they must disable interrupts while reading that data. Otherwise it would be possible for several tasks to be sending conflicting commands to the ADC-1. This is accomplished with the INTOFF (interrupts off) in 655 and the INTON (interrupts on) in 865. The rest of the analog-read logic is identical to that used in Program One, except that these tasks also contain some code to print results to the CRT screen.

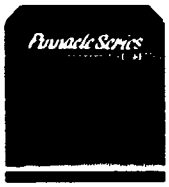
Where line 670 says CURSOR 9,3 it is positioning the cursor within the WINDOW being used, not the entire screen. In this program it amounts to the same thing, but the distinction is important in programs that use several windows. Again, the processing logic is well documented in the source code, so I'll comment here on the MTBASIC syntax.

Line 730 uses the BAND (bit-logical-and) function to check the byte at the status port (INP(6) for the Kaypro). If bit 1 is 0, a complete byte hasn't been received at the serial port. FPRINT is used again in line 850, printing the RESULT in a 5-character integer format with a carriage-return at the end (the Z). Each of these seven tasks prints a message such as "READING" beside the screen line for that channel, letting the user know when a particular channel is being read. Then it gets the value from the ADC and prints that in its


wabash®

Pinnacle Series DISKETTES

77¢ ea.



Certified 100% Error Free
Meets all Industry Standards
Manufacturer of Magnetic Media for Over 20 years
Reinforced Hub Ring
Lifetime Warranty

BOXED with ENVELOPES and LABELS

| | 10 | 100 |
|---|---|---|
| 5.25" SSDD, SOFT SECTOR, w/hub ring | $ .95 | $ .77 |
| 5.25" DSDD, SOFT SECTOR, w/hub ring | 1.09 | .88 |

wabash DATATECH DISKETTES

Lifetime Warranty 100% Error Free
UNIQUE EASEL-BACK CASE functions as Library Box for convenient, permanent storage and easy diskette access.

| | 10 | 100 |
|---|---|---|
| 5.25" SSSD, SOFT SECTOR, w/hub ring | $ .98 | $ .91 |
| 5.25" SSDD, SOFT SECTOR, w/hub ring | 1.18 | .99 |
| 5.25" DSDD, SOFT SECTOR, w/hub ring | 1.28 | 1.11 |
| 5.25" DSQD, SOFT SECTOR, w/hub ring | 1.88 | 1.59 |
| 5.25" SSSD, 10 SECTOR, HARD w/hub ring | 1.18 | .98 |
| 8" SSSD, SOFT SECTOR, Unformatted | 1.79 | 1.59 |
| 8" SSDD, SOFT SECTOR, Unformatted | 1.88 | 1.69 |
| 8" DSDD, SOFT SECTOR, Unformatted | 1.98 | 1.89 |
| 3.5" SSDD, 135 TPI | 2.79 | 2.59 |
| 3.5" DSDD, 135 TPI | 3.89 | 3.69 |

SUPER SPECIAL!!
5.25" HIGH DENSITY DISKETTES for use on IBM PC-AT
10 / $2.35 each, 100 / $2.20 each

DEALERS! SCHOOLS! USER GROUPS!
CALL FOR VOLUME DISCOUNTS

Terms: Add $3 shipping & handling for U.S. orders. Outside USA add $10 to cover postage. In Illinois add 7% sales tax or provide resale certificate. Prices & terms subject to change without notice.

CALL TOLL FREE (orders only)
VISA 1-800-222-1248 MasterCard
in Illinois or for information
312-882-8315
AUTHORIZED WABASH DISTRIBUTOR

DIGITAL IMAGES
1185 TOWER RD. SCHAUMBURG, IL 60195

proper place on the same line.

Line 860 turns on the flag K, indicating that a channel has been read, and places the channel number in the variable SENSOR. These values are used by the task that writes to the disk file.

Task 8 reads the Kaypro clock card, and again, the comments show what each line is meant to do. This task would need to be modified (maybe a little, maybe a lot) to read from a different clock.

Task 9 writes to the disk file. Since RESULT is used by all the analog-read tasks, and since the proper channel number, time, and date stamps must accompany the RESULT into the disk record, interrupts are turned off at the beginning of the task and back on at the end. Line 2740 checks to see whether a channel read has been done. In other words, if K is a 0, nothing is done. If K has been turned on by one of the other tasks, then the record is written.

The FILE 1 statement at 2750 determines which open file is used (here, we're only using one). After the FILE 1 statement, the FPRINT statement (or any regular PRINT statement) will output to the file, not the CRT screen. The FILE 0 statement that follows will cause subsequent PRINTs to go to the CRT until a file is specified again. Finally, the flag K is cleared and we're done.

```
Start:
   Write(^Z);                                  (* clear screen *)
   Write('Channel to read? (0 to stop) ');
   Read(Channel);
   If Channel = 0 then goto Stop;
   Write('    How many samples? (1-10,000) ');
   Readln(J);
   If J = 0 then goto Stop;
   Writeln;
   GetTime(Time);

   Val(Time,StartSeconds,Code);
   StartSeconds := StartSeconds + 0.04;
   Writeln('Started at:  ',(StartSeconds):7:2);
   For I := 1 to J Do Buffer[I] := Analogin(Channel);
   GetTime(Time);
   Val(Time,StopSeconds,Code);
   Writeln('Stopped at:  ',StopSeconds:7:2);
   Elapsed := StopSeconds - StartSeconds;
   Writeln('Elapsed time:',Elapsed:7:2,' Seconds');
   Samples := J;
   Writeln('Sample Rate: ',(Samples / Elapsed):7:2,' Samples/Second'^J^J);
   Write('How many samples to display? (<RET> for all) ');
   Readln(J);
   Writeln;
   For I := 2 to J Do Write(Buffer[I],'    ');
   Writeln;
   CalculateStandardDeviation;
   Write('Sample mean: ',SampleMean:7:2);
   Writeln('            Standard Deviation: ',StandardDeviation:7:2);
   Write(^J,'Press <RETURN> to continue... ');
   Read(J);
   Goto Start;
Stop:

End.
```

**Final Notes On MTBASIC**

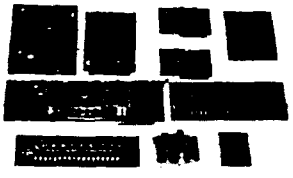
MTBASIC provides many useful features that endear it to the heart of anyone who likes to write good, readable code. Its one significant drawback, I feel, is the way redundant spaces are stripped from a program with the SAVE command. The apostrophe (single quote) can be used to define a REM statement, as you see here, but beware of the SAVE command. The compiler won't object to the neat spacing that you provide to make your source readable, but if you SAVE a program from MTBASIC, those spaces will be gone!

MTBASIC (up to the version I have) does not contain any kind of editor, not even a line editor like MBASIC® . You CAN replace a line by re-typing it, and small errors can be corrected this way and the program re-compiled for testing.

To make the changes permanent in your source code, however, use WordStar® , or VDO® , or some other editor to alter the source. This gets a little tiresome—load the editor, change the program, load MTBASIC, compile, crash, load the editor, change the program... . But the result will be a source that hasn't been corrupted by having all the spacing removed from it. Months later, when you need to make a modification, you'll be glad you did it the hard way.

In my experience, MTBASIC data acquisition programs are capable of logging analog data nearly as fast as an equivalent Turbo Pascal program, provided that use isn't made of the windowing feature, which takes time to execute. Testing an ADC-1 with nearly identical versions of HISPEED, written in MBASIC, MTBASIC and Turbo Pascal, I got results of 10, 88, and 103 samples per second, respectively. I recommend MTBASIC for applications that can benefit from interrupt-driven multitasking capabilities, as can many Real World computing applications. ■

MICROCOMPUTERS
AND
INTERFACES



We have six singl[illegible]rd computers, two video boards an[illegible]aces for the IBM-PC and AP[illegible]ers. You can use our pr[illegible] systems, heat contr[illegible], light cont[illegible] automated slide sh[illegible], irrigation systems, [illegible]tems, automated p[illegible]ot control just to n[illegible]available. For catalog call [illegible]ite to:

JOHN BELL ENGINEERING, INC.
400 OXFORD WAY
BELMONT, CA 94002
(415) 592-8411

---

*Jerry Houston has degrees in Business Administration and Data processing, and has published numerous articles in the area of Real World computing. He is Marketing Director for Remote Measurement Systems, Inc. of Seattle and teaches computer programming at Griffin College and Seattle Central Community College. Mail will reach him at:*

*Jerry Houston*
*2633 Eastlake Ave. E, Suite 206*
*Seattle, WA 98102*

# Books of Interest

**Design and Application of Small Standarized Components**
Data Book 757
Published by Stock Drive Products
55 South Denton Ave.
New Hyde Park, NY 11040
Distributed by Educational Products
PO Box 606
Mineola, NY 11501
784 pages, 5½ × 7½ softbound, $7.95

One of the reasons I became so interested in computers is that they enable me to control mechanical devices such as lathes, robots, and other motion producing devices. I have a long list of mechanical projects to work on and I have most of the computer programming and interfacing information I need, but I have had difficulty locating application information on the small gears, shafts, bearings, motors, timing belts, etc. needed for assembly. We live in a rural area, and the saying is "If it doesn't fit on a steer or a sawmill you can't get it." I can get information on two inch diameter shafts and bearings, but if I ask about 1/16" shafts they look at me like I'm nuts (I probably am or I wouldn't be messing with computers, but that's besides the point). I was very happy to locate this Data Book, and they also have Master Catalog 757 "Handbook of Small Standardized Components" in which they list parts for sale.

It is difficult to review this Handbook because it is a collection of text, charts, and specifications on a wide variety of subjects and I can't list everything it covers. I'll briefly describe the various sections, and comment on the parts I find esecially interesting.

•**Section D1 Designers Data.** Part A-Mathematical formulas and tables including energy, inertia, torque, power and work. Part B-Conversion of units. Part C-Properties of materials, finishes and surface texture, including properties of steels, heat treating terms, hardness conversions, and surface texture and roughness measurement. Part D-Useful shop data, including drill sizes, standard gages, primary fits, press fits, bushing and shaft bore sizes. Part E-Fasteners including standards for inch size fasteners, screw head dimensions, strength of fasteners, screws and pins, pin standards, and non-metallic fasteners.

•**Section D2 Practical Design Hints.** Part A-Machining; minimizing machining, effective clamping, simplifying machining with composit assemblies, turning, boring, drilling, milling, broaching, grinding, and tapping. Part B-Assembly; avoiding overspecification of dimensions of mating parts, allowing for thermal expansion and wear, designing for accesibility and ease of disassembly.

•**Section 1 Gear Design.** This section can help you understand the terms you will run into when specifying and ordering gears. It covers involute curves, pitch circles, diametral pitch, gear nomenclature, pitch diameter and center distance, pressure angle, helical gears, spur gears, racks, internal gears, worm gears, bevel gearing, strength, materials, lubrication, etc.

•**Section 2 Speed Reducers and Gear Trains.** While the previous section covered the subject of gears, this sections covers two or more gears working together (called a gear train). Some of the topics are: simple spur gear trains, compound spur gear trains, planetary gears, bevel gear differentials, force transmission, tangential forces, power flow, efficiency, minimization of gear train inertia, backlash, transmission error, configurations and designs, limit stops, etc.

•**Section 3 Basic Electricity.** A brief section on AC circuits, fundamental laws, and three-phase power.

•**Section 4 Motors.** Part A: Design considerations, motor characteristics, selecting gear motors, motor speed controls. Part B: Escap® ironless rotor DC micromotors and step motors, physical laws governing Escap motor applications, basic motor physics, gearboxes, control of DC micromotors, tutorial on micromotor application problems, Escap step motors. Part C: Step motors; principal types primary step motor performance characteristics, damping, step motor driving and control, considerations for step motor applications.

•**Section 5 Negator® constant force springs and spring motors.** Design, stock parts, saftey considerations.

•**Section 6 Couplings and Universal Joints.** Types of flexible couplings, kinematics, joint selection, torque rating, secondary couples, joints in series.

•**Section 7 Drives.** Part A: belt and chain drives; V belt design data, roller chain design data, center distances, lubrication, tensioning mechanisims. Part B: synchronous belt drives; belt tooth configurations, pully pitch, design and installation, standards, drive ratio tables, center distance factor tables, design based on horsepower or allowable torque.

•**Section 8 Shafts and bearings.** Determination of stresses for solid shafts, sizing porous-metal bearings, nonmetalic bearings, ball bearings, bearing loads, tolerances and clearances.

•**Section 9 Vibration and shock isolation.** Fundamentals of vibration, vibration analysis, vibration mounts.

•**Section 10 Introduction to robots.** Manipulator geometry; joints, degrees of freedom, yaw pitch and roll, dimensions of a link, geometric configurations, load capacity. Manipulator motion, manipulator components, applications.

This data book does not cover the subjects as extensively as an Engineering manual but it would take a shelf-full of books to cover this wide range of information. It is very useful for the computerist who needs a quick source of information on sizes, fits, gears, center distances, strength, and other information. ■

**Unsoldering**

*(continued from page 18)*

technicians who are intimately familiar with the product. Confronted with a bad board they often zero in on the offending chip with just a few simple tests. Rarely do they need to unsolder more than one or two IC's before they cure the malfunction. Anyone who has repaired many boards can confirm this fact because a few, very enlightened vendors will include the troubleshooting steps in the documentation and by following that data you can quickly isolate the bad components. Otherwise if unfamiliar with a circuit you may wind up unsoldering five, six, seven or more IC's before achieving a fix. Clearly if the IC's are in sockets the repair process will proceed far quicker and easier.

**Printed Circuit Boards, The Inside Story!**

It may seem out of place to talk about circuit boards in this article when the next will be all about repairing them, but since unsoldering failures are the chief cause for needing to repair them, it's useful to briefly discuss them so as to limit the potential problems. Actually we're going to talk about geography and social issues that may seem to have nothing do with our prime subject but in fact they do.

In North America most circuit boards are manufactured not by the kit makers or product assemblers, but by specialized firms that act as subcontractors. Many of the boards these firms produce are destined for products that will be used by the Military. When circuit boards first appeared the Military tested them under the harsh conditions their equipment must endure, and the early boards failed miserably, however the potential benefits of boards led makers to develop types that could stand up to Military testing. Eventually these types were approved and a set of specifications called 'Mil Specs' defines how these boards are to be made. Typically these boards use glass filled epoxy construction techniques which require special manufacturing procedures but the resulting boards are very durable, indeed. Most circuit board makers must be able to produce Mil Spec boards and find it economical to only produce that type, which means that even if a company doesn't need such rugged boards they will get them anyways and it won't cost extra.

If a circuit board was made in North America then chances are it is of epoxy type construction and can withstand unsoldering heat very well, also the traces will adhere a little better thus reducing the chance of delamination. These boards often seem able to withstand an M-1 Tank.

In the Far East which is the other great source of products with circuit boards there is almost no need to meet Mil Specs and most product makers produce their own boards. They make boards the old fashioned way which is a paper laminating technique. These boards stand up well to consumer use but not so well to unsoldering heat. It's quite easy to delaminate both the traces and even the board itself by using too heavy a hand with the iron.

What we said above is only a generalization, both types of boards are produced in both places so knowing the origin of a board is just the first step in identifying its construction. Epoxy boards are usually a light beige color in their raw state but more likely are green or blue due to an applied solder mask. Paper laminate boards are often


EVERYTHING YOU NEED . . . $279.00

Now it's easy to program the Heath-Zenith HERO-1® Robot with an Apple® II. HERO® Macros for the S-C Software 6800 Cross Assembler program in Heath's Robot Interpreter Language with easily remembered mnemonics. The HERO® Macros come with 30 pages of documentation.

[For example, the line: 1130 > MVWRIM GRIP, OPEN, 60, FAST instructs the HERO® to open his gripper 60 units at fast speed. Motor position is expressed in base-10.]

Transfer to HERO® with ROBI . . . an affordable interface for the robotics experimenter . . . is simple.

- ROBI is a complete package. No additional hardware required for Apple® or HERO®.
- ROBI installs quickly in an Apple® II, II+, or IIe. Once installed, no hardware changes are needed. Within minutes, you will be programming HERO®.
- With ROBI and the Cross Assembler, the programmer uses Apple®'s memory to write the program, and HERO®'s memory to run the program.
- Not "copy protected," archival copies may be made as needed.
- ROBI offers expansion potential.

VISA and MasterCard accepted.

BERSEARCH Information Services

26160 Edelweiss Circle
Evergreen, Colorado 80439

The Cross Assembler with HERO® Macros sells for $100.00; the ROBI Interface sells for $199.00. Both as a package — $279.00.

To order, or for more information, call (303) 670-6137.

APPLE® is a trademark of Apple Computer. HERO® is a trademark of Heath Electronics.

## With these tools you can really program!

**S-C Macro Assembler** — Version 2.0 is the latest in a long line of enhancements to the oldest commercially available assembler for Apple computers. Supports the full instruction set and all addressing modes of the 6502, 65C02, 65802, and 65816 processors, as well as Steve Wozniak's SWEET-16 pseudo-processor. Compatible with any Apple II, II Plus, //e, or //c having at least 64K RAM and one disk drive.

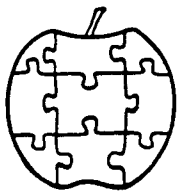
S-C Macro Assembler is well-known for ease-of-use and powerful features, rated number one by "The Book of Apple Software". You get an integrated editor/assembler with over 29 commands and over 20 directives including macros, conditional assembly, global search/replace, edit, and more. Thousands of users in over 30 countries and in every type of industry attest to its speed, dependability, and user-friendliness. S-C Macro Assembler blends power, simplicity, and performance to provide the optimum capabilities for both beginning and professional programmers.

Both DOS 3.3 and ProDOS versions are available now, $100 each; you can save by getting both together for only $120. Upgrades are available for owners of previous versions. Disks are not copy-protected, so you can easily make back-up copies.

**Cross Assembler Modules** — Owners of the S-C Macro Assembler may add the ability to develop programs for other systems. We have modules for most of the popular microprocessors, priced from $32.50 to $50 each: Motorola 6800/1/2, 6805, 6809, 68000; Hitachi 6301; Zilog Z-80, Z-8; Intel 8048, 8051, 8085; RCA 1802/4/5; DEC LSI-11; GI 1650, 1670; and others.

All of the cross assemblers retain the full power of the S-C Macro Assembler. You can develop programs for burning into EPROMs, transfer through a data-link, or direct execution by some of the plug-in co-processor cards now on the market.

**Apple Assembly Line** — Monthly newsletter for assembly language programmers, beginner or advanced. Increase your skill with the most powerful and important language available for your Apple. Packed with techniques; handy utility programs; commented listings of code from the ROMs, DOS, and ProDOS; reviews of the latest relevant books and hardware; and more! All back issues since October 1980 available. Subscription $18 per year (add $3 for first class postage in USA, Canada, Mexico; add $14 postage for other countries).



**S-C Software Corporation**
2331 Gus Thomasson, Suite 125
Dallas, Texas 75228
(214) 324-2050

**Professional Apple Software Since 1978**
Visa, MasterCard, American Express, COD accepted
Apple is a trademark of Apple Computer Inc.


a deep brown shade from the lacquer material used to seal them. If you can't be sure then proceed as though it is a paper laminate type, in other words treat it carefully.

**Wrapup**

In the Olympic games there is an event called the High Jump. It involves one competitor starting from a standing position and jumping as high as possible over a bar, no pole or other mechanical device can be used. The premier American in this event is Dwight Stone and if you saw him during the '84 Summer Games, you would have seen him standing in front of the bar for some time deep in thought. He was not 'pysching' himself up, rather he was mentally rehearsing the jump, thinking through every move in his mind before actually doing it. In many ways unsoldering demands the same mental preparation if it is to be successful. Also like the games, success is not assured, Dwight Stone did not win the gold medal but he did compete.

Here is a simple checklist for unsoldering:

1. Decide what is the most important goal. Don't try to save a 50 cent IC only to damage an expensive board.
2. Have a plan, that is what tool or method will be used.
3. Be prepared to change the plan if it isn't working.
4. If you're trying a method for the first time, then practice on a scrap item.
5. Remember nothing works all the time and everything can be fixed, look for the next article in this series.

**Sources**

To say that we've only scratched the surface of the art of unsoldering is to put it mildly. Clearly the key to reliably practicing this art is to practice it. But not on the motherboard of a two thousand dollar computer, use a scrap board. Where can you get scrap boards? Check with the local TV or Stereo repair place, they may be throwing them away. A local electronics manufacturer may also be a source, be sure to ask if you can take any apparently discarded boards. Some surplus electronics outlets sell scrap boards, people buy them to experiment with or to salvage parts off them. Remember that just like functioning boards, scrap boards come in different types and present a variety of unsoldering challenges.

Look in the back of electronics magazines for the names of surplus firms. Below is a firm that currently has a limited supply of boards with relays and other parts that are sold as salvageable.

John J. Meshna, Jr. Inc., PO Box 62, E. Lynn, MA 01904 (617)595-2275. Cat. No. SP-88A has one relay and lots of parts for $2.00, SP-88/2 two relays and parts, $2.00, SP-88/3 three relays for $3.00 etc. They do have a $15 minimum order so you may wish to get their catalog No. SP-36 and see all the weird stuff they sell.

For a good catalog of unsoldering tools: Edlie Electronics, 2700 Hempstead Turnpike, Levittown, NY 11756, (800) 645-4722.

A short-form catalog of soldering and desoldering tools is available by writing to: Customer Service, OK Industries, 3455 Conner St., Bronx, NY 10475. ■

# Build the Circuit Designer 1 MPB

## Part 2: Programming the Single Board Computer

by Neil Bungard

### Introduction

In the last issue of *The Computer Journal* we looked at Part 1 of an article describing the Circuit Designer-1, MPB (CD-1, MPB). In that article, the design of the CD-1, MPB (an 8035 based single board computer) was presented and its operation was explained. This month I will mention a few applications for which I have personally used the CD-1, MPB, and we will present a simple read/write memory circuit which plugs into the CD-1, MPB program memory socket and allows the user to enter programs in machine language.

The CD-1, MPB was originally created because of the need for a simple, general purpose, single board computer to be used as an intelligent control unit in several projects which were under construction. Among these projects were an automatic telephone number recording system, a CO gas monitoring instrument, and a classroom microprocessor trainer. The CD-1, MPB successfully served as the control unit for all of these projects with changes only in the software from project to project.

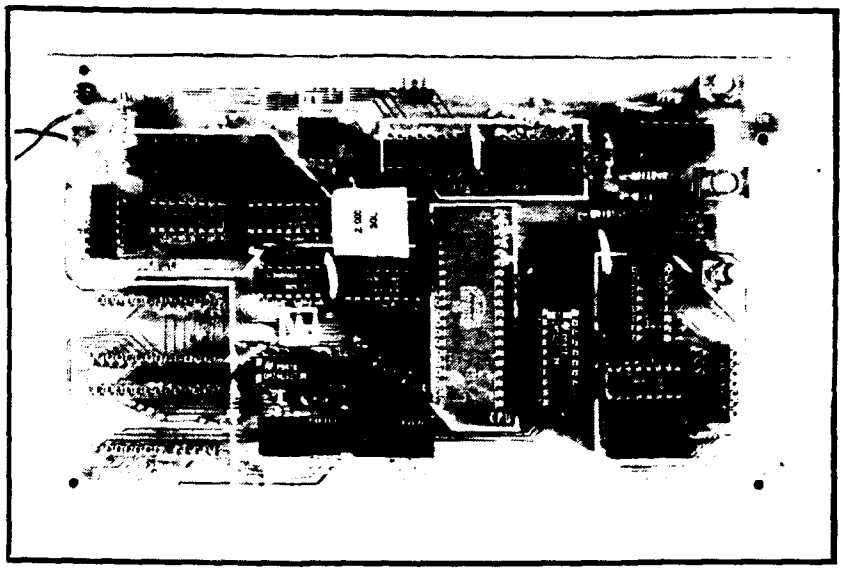


**Figure 10:** Photo of the completed CD-1.

### Programming the CD-1, MPB

**The Instruction Set.** As in any microprocessor control system, operations are controlled via program instructions. The 8035 has an instruction set which consists of 90 commands. The 8035 literature is out of print, but specifications on the 80C35 which has the same instruction set are available from Intel by calling 1-800-538-1876 (or 1-800-672-1833 in California) and requesting the 80C48/80C35 specifications #210943-001. The instruction set for the 80C48, 80C35, 80C49, and 80C39 is shown in Table 4.

**The Interface Sockets.** The CD-1, MPB communicates with the outside world via its on board interface sockets. The placement, pin assignment, and explanation of the interface sockets are shown in Figure 12.

**Using R/W Memory In The Program Memory Space.** Since the 8035 can only execute instructions from its program memory, which is read only memory, there is no capability for writing dynamic program instructions into the program memory area. Furthermore, since the size of the program memory is only 4K bytes, the manufacturer decided not to give the CPU direct memory access (DMA) capabilities. What this means is that the program memory of the 8035 was meant to be PROM or EPROM.

With the CD-1 memory board it is possible to write dynamic programs into the 8035 program memory space. This is done by having the CD-1 memory board perform a DMA. The DMA is accomplished by the following sequence of events:

1. (Refer to Figure 15 concerning this explanation). Placing the run/examine switch on the memory board to the examine position (closed) pulls the reset input on the CPU LO through diode D1. (This puts the CPU in the reset mode).

2. The data bus is disconnected (Tristated) from the CPU when the CPU is in the reset mode. Also with the run/examine switch in the examine position a ground is applied to the base of inverting transistor T1. The output of the inverting transistor is applied through the memory board interface socket and disconnects the address bus multiplexer outputs of the CPU board from the memory ICs (IC6 and IC7). The address and data buses are now free for use by the memory board.

3. The address and data lines to the memory ICs are connected to a set of DIP switches via IC1, IC2, and IC3 (74LS373's). When the run/examine switch (S1) is in the examine position (closed) a ground is applied to pin 1 of IC1 and IC2 thus connecting one side of the address DIP switches to the address inputs of the memory ICs. The memory ICs are enabled via a ground applied by switch S1 through diode D2. The data DIP switches remain disconnected from the memory ICs until the deposit switch (S2) is depressed, and logic states are applied to the address and data bits via the DIP switches. When the switches are open the lines are floating and the memory ICs see a logic 1. To apply a logic 0

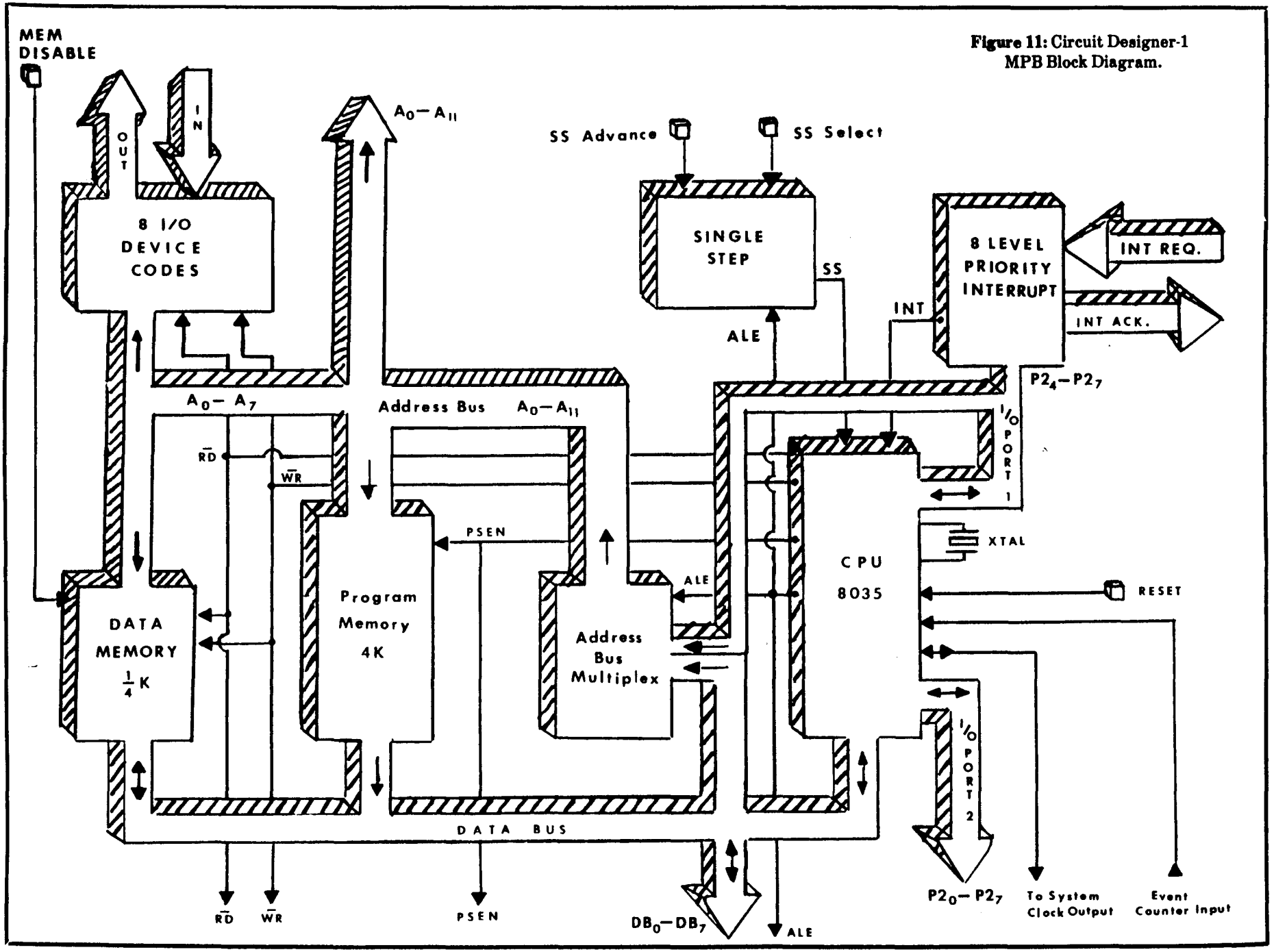



**Figure 11:** Circuit Designer-1 MPB Block Diagram.

to an address, or data bit, the bit must be grounded by closing its corresponding DIP switch.

4. Examining the contents of the memory locations is accomplished with S1 in the examine position (closed) and S2 not depressed (open). In this configuration the memory ICs will be in the read mode and the contents of the memory location set on the address DIP switches will be reflected on the memory board displays. Programming is accomplished by setting the address and data DIP switches to the desired values and depressing the deposit switch (S2). This writes the data into the selected memory location, which is accomplished by simultaneously enabling pin 1 of IC3 (74LS373), and pulling pin 10 LO on the memory ICs. When the deposit switch is released the change will be reflected by the memory board displays.

5. After all the instructions have been entered into memory the program is executed by placing the run/examine switch in the run position (open). This disconnects the memory board data and address lines from the program memory, connects the CPU address and data lines to the program memory, and removes the ground from the reset pin on the CPU. This is accomplished in exactly the reverse order as going from the run mode to the examine mode in the explanation above.

$\overline{\text{PSEN}}$ is the program memory read pulse which inputs instructions to the CPU. $\overline{\text{PSEN}}$ enters the memory board via pin 24 of the interface socket and goes to the $\overline{\text{CS}}$ input of the memory ICs. This pulse is active LO. Also, a capacitor and resistor circuit is connected to the reset pin on the CPU to cause a small time delay when going from the examine mode to the run mode. This is necessary to ensure that the address bus multiplexer outputs from the CPU are completely enabled before the CPU begins executing instructions.

In the CD-1 Program Memory Board design (see Figure 15), 2114 memory ICs were used. These ICs are 1K by 4-bit memories, and by utilizing two 2114's, 1K of program memory was obtained. If you need more than 1K of program memory, you can use different memory ICs or add additional 2114's and decode the higher order address bits to enable the appropriate 1K "block". The 8035 has the capability of accessing up to 4K bytes of program memory.

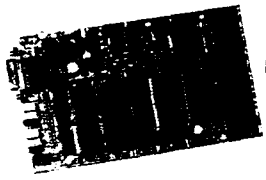
Once a program has been written and debugged using the CD-1 program memory board, the same program can be written into a 2716, 5 volt EPROM which will directly replace the memory board. The 2716 EPROM is non-volatile memory which means that when power is removed it will retain its program. This enables the CD-1 microprocessor board to be utilized as a dedicated monitor and/or control device. The CD-1 microprocessor board has sockets to accommodate two 2716's so that frequently used subroutines could be written into a 2716 and placed in the higher 2K byte memory socket. With the CD-1 memory board plugged into the lower 2K byte memory socket, dynamic programs can be written with the ability to call subroutines from the 2716 EPROM on the microprocessor


MSC-LAT1
$649

KAYPRO™ users can share the advantage to LAT1. Just take off your main KAYPRO board and put LAT1-K into your cabinet.
All advantage of LAT1 is yours now./
$bd+

ZENET NETWORK through twist pair
• 6Mhz HD64B180 (Z80 upward compatible ) 512K byte on board (256K installed, 384K RAM DISK)
• LAN:ZENET port 800K baud CSMA CD twist pair bus type upto 500 meters HDLC
• Floppy: 3.5, 5 and 8 inch, d/s [illegible] side and d/s track automatic desity/[illegible]g
• Hard disk: SCSI interface [illegible]
• Video: 80 [illegible]h [illegible] (color) and 640 X 200 pixel color graphic [illegible] byte video RAM character set download from disk
• Timer: battery back up calendar
• Serial: RS232C X 2 and TTL X 1
• Parallel: centronics type, 16 bit TTL, 7/8 bit keyboard port (32 characters FIFO)
• O.S.: Turbo Dos, MP/M (multiuser) banked CP/M plus (single user)
• Size: 10 X 6 inch 4 layered
• Assembled and tested
• BIOS source code available
• Completely faster than other Z80SBC

MSC-PCX
8088 expansion card for LAT1 soon available


MSC-MTC
WORLD SMALLEST COMPUTER
• Full personal CP/M system in palm 4mhz Z80 256K RAM (128K RAM DISK)
• Serial: RS232C X 2 automatic baud rate checking
• Pararell: centronics type printer port
• Floppy: 3.5 inch micro floppy disk drive 800K byte (option 5, 3.5 inch drive d/s sided d/s track automatic density checking)
• O.S.: CP/M plus bank version
• BIOS source code available
• Completely faster than other Z80SBC

MSC-MTC/P
Full assembled pcb of MTC under $[illegible] in OEM quantity
$299

MSC-ICO
$499
Full featured CP/M plus system
• Z80 4mhz 128K Byte RAM [illegible] Floppy 3.5, 5 and 8 inch d/s density d/s sided and d/s track upto 4 disk drives Automatic density format check
• Serial: RS232C X 2
• Pararell: Centronics type, 16 bits I/O, 7/8 bit keyboard port
• Timer: battery back up calendar
• Video: 80 X 24 high speed CRT controller
• O.S.: CP/M plus bank version included
• Size: 10 X 6 inch 4 layered
• BIOS source code available
• DRI CP/M plus manual $50
• New word word processor program for MSC-ICO ADD $50
• Completely faster than other Z80SBC

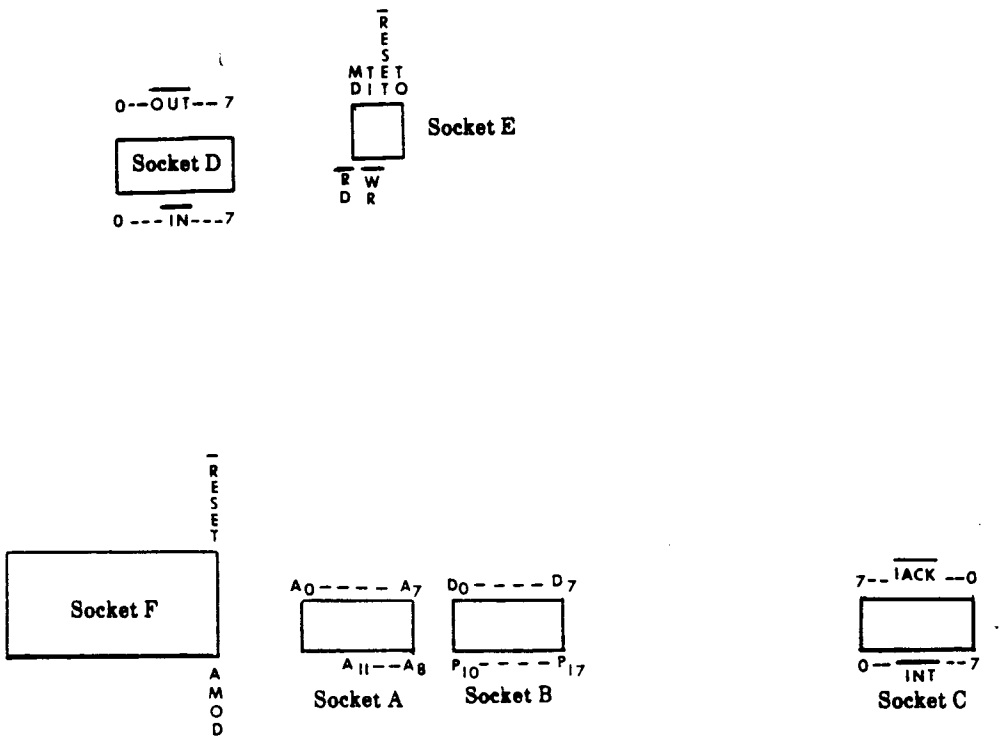
MSC-HCS
$199
Expansion card for ICO
RAM disk (upto 2M byte) and SCSI hard disk interface card for ICO with installation program

CP/M plus is a registered trademark of Digital Research Inc.
Z80 is a registered trademark of Zilog Inc.
Turbo Dos is a registered trademark of Software 2000 Inc.
Mountain Side Computer and ZENET are trademark of Southern Pacific Limited

Distributors
England-Quanta systems 01-253-8423
Denmark-Danbit 03-662020
Finland-BB Soft 90-692-6297
India-Betamatix PVT Ltd. 0812-71989

Manufacturer and international distributer
SOUTHERN PACIFIC LIMITED
Sanwa Bldg., 2-16-20 Minamisaiwai, Nishi, Yokohama, JAPAN 220
Phone: 045-314-9514 Telex: 3822320 SPACIF J
Advanced single board computer technology company

USA distributer
SOUTHERN PACIFIC
COMPUTER PRODCTS U.S.A., INC.
P.O.BOX 4427, Berkeley, CA 94704-0427 U.S.A.
Phone: 415-253-1270
Dealer and distributer inquiries welcome




**INTERFACE SOCKETS**

This is a diagram of the pin assignment and placement for the interface sockets.
A0 through A11-multiplexed address bus outputs.
D0 through D7-data bus interface (bidirectional).
P10 through P17-I/O port 1 interface (quazibidirectional).
$\overline{\text{INT}}$ (0 through 7)-priority interrupt request inputs (active lo).
$\overline{\text{IACK}}$ (0 through 7)-interrupt acknowledge outputs (active lo).
AMOD-address multiplexer output disable (active high).
RESET-CPU reset (active lo).
$\overline{\text{RD}}$-external data memory read pulse.
$\overline{\text{WR}}$-external data memory write pulse.
TO-a test input which may be sampled by a conditional jump instruction. The internal CPU clock signal can be output via TO.
T1-A test input which can be sampled by a jump-on-condition instruction. T1 can also be used to input a signal to counter/timer logic when it is serving as an event counter.
M.D.-External memory disable.
$\overline{\text{OUT}}$ (0 through 7)-output device code pulses.
$\overline{\text{IN}}$ (0 through 7)-input device code pulses.

**Figure 12:** Interface sockets.

board.

The memory circuit explained above is adequate for entering small programs for system checkout or for gaining an understanding of how the CD-1, MPB operates on its most basic level. But, if longer programs are to be executed on the CD-1, MPB, some other method of entering code will be needed. In the next issue of *The Computer Journal* a more effective way of entering program instructions into the 8035 program memory space will be discussed.

## Construction of CD-1, MPB

Double sided circuit boards have presented a problem to hobbyists in the past because of the need for plated through holes. The CD-1 utilizes a double sided board but to eliminate the need for plated through holes I have inserted and soldered wires on both sides of the feed through holes, soldered passive components (resistors, capacitors, etc.) on both sides of the board, and used Molex® connectors (soldered on both sides) for the ICs. Care must be taken to insure that cold solder connections are avoided. One faulty connection on a single board computer could cause many hours of troubleshooting misery before the problem is located. A parts placement layout is provided in Figure 13 to aid in the construction and assembly of the CD-1, MPB, and a complete parts list is provided in Figure 14.

Figure 10 is a photograph of the completed CD-1, MPB. Full size positive prints of a silk screen outline and the front and back foil patterns are available for $1.00 postage. Film

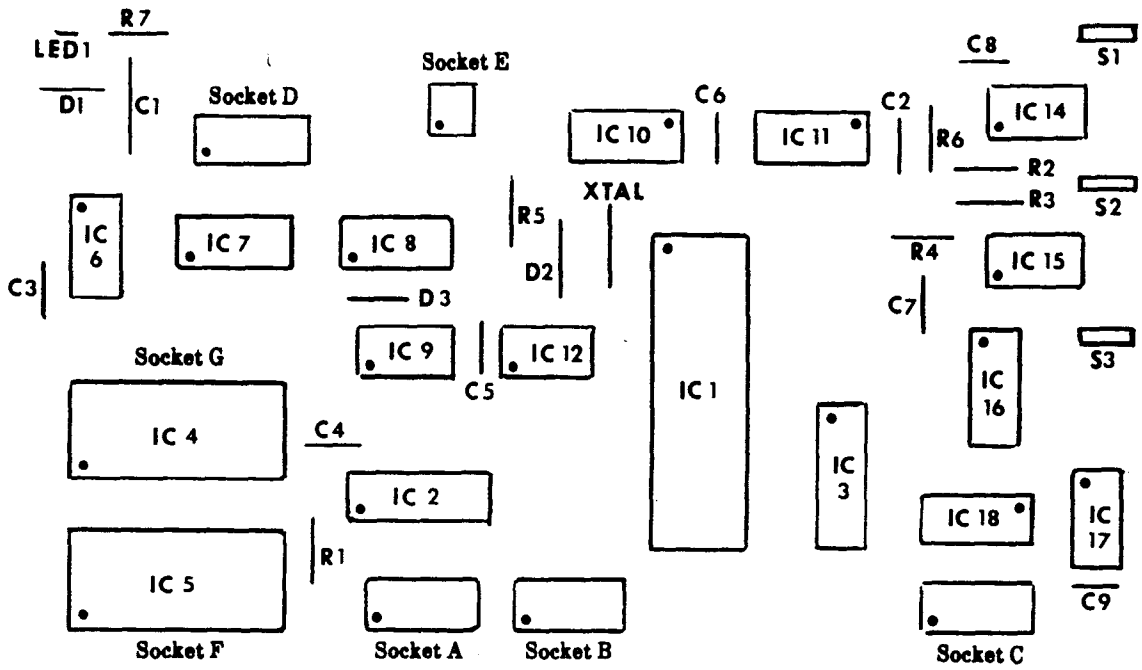



**Figure 13:** Parts placement.

CTM

*"...received my moneys worth with just one issue..."*

*—J. Trenbick*

*"...always stop to read CTM, even though most other magazines I receive (and write for) only get cursory examination..."*

*—Fred Blechman, K6UGT*

U.S.A. ..... $15.00 for 1 year
Mexico, Canada ..... $25.00
Foreign ..... $35.00(land) - $55.00(air)
(U.S. funds only)
Permanent (U.S. Subscription) ..... $100.00
Sample Copy ..... $3.50

**CHET LAMBERT, W4WDR**
1704 Sam Drive • Birmingham, AL 35235
(205) 854-0271

# FREE SOFTWARE

## RENT FROM THE PUBLIC DOMAIN!

User Group Software isn't copyrighted, so there are no fees to pay! 1000's of CP/M and IBM software programs in .COM and source code to copy yourself! Games, business, utilities! All FREE!

**CP/M USERS GROUP LIBRARY**
Volumes 1-92, 46 disks rental—$45

**SIG/M USERS GROUP LIBRARY**
Volumes 1-90, 46 disks rental—$45
Volumes 91-176, 44 disks rental—$50
*SPECIAL! Rent all SIG/M volumes for $90*

**K.U.G. (Charlottesville)** 25 Volumes—$25

**IBM PC-SIG (PC-DOS) LIBRARY**
Volumes 1-200, 5¼" disks $200

**174 FORMATS AVAILABLE! SPECIFY.**

Public Domain User Group Catalog Disk $5 pp. (CP/M only) (payment in advance, please). Rental is for 7 days after receipt, 3 days grace to return. Use credit card, no disk deposit. Shipping, handling & insurance—$7.50 per library.

**(619) 914-0925 information,(9-5)**
**(619) 727-1015 anytime order machine**
Have your credit card ready! VISA, MasterCard, Am. Exp.

**Public Domain Software Center**
1533 Avohill Dr.
Vista, CA 92083

**Table 4:** 80C35/80C48 Instruction Set. Courtesy of Intel.

**Accumulator**

| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| ADD A, R | Add register to A | 1 | 1 |
| ADD A, @R | Add data memory to A | 1 | 1 |
| ADD A, # data | Add immediate to A | 2 | 2 |
| ADDC A, R | Add register with carry | 1 | 1 |
| ADDC A, @R | Add data memory with carry | 1 | 1 |
| ADDC A, # data | Add immediate with carry | 2 | 2 |
| ANL A, R | And register to A | 1 | 1 |
| ANL A, @R | And data memory to A | 1 | 1 |
| ANL A, # data | And immediate to A | 2 | 2 |
| ORL A, R | Or register to A | 1 | 1 |
| ORL A @R | Or data memory to A | 1 | 1 |
| ORL A, # data | Or immediate to A | 2 | 2 |
| XRL A, R | Exclusive or register to A | 1 | 1 |
| XRL A, @R | Exclusive or data memory to A | 1 | 1 |
| XRL, A, # data | Exclusive or immediate to A | 2 | 2 |
| INC A | Increment A | 1 | 1 |
| DEC A | Decrement A | 1 | 1 |
| CLR A | Clear A | 1 | 1 |
| CPL A | Complement A | 1 | 1 |
| DA A | Decimal adjust A | 1 | 1 |
| SWAP A | Swap nibbles of A | 1 | 1 |
| RL A | Rotate A left | 1 | 1 |
| RLC A | Rotate A left through carry | 1 | 1 |
| RR A | Rotate A right | 1 | 1 |
| RRC A | Rotate A right through carry | 1 | 1 |

**Input/Output**

| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| IN A, P | Input port to A | 1 | 2 |
| OUTL P, A | Output A to port | 1 | 2 |
| ANL P, # data | And immediate to port | 2 | 2 |
| ORL P, # data | Or immediate to port | 2 | 2 |
| INS A, BUS | Input BUS to A | 1 | 2 |
| OUTL BUS, A | Output A to BUS | 1 | 2 |
| ANL BUS, # data | And immediate to BUS | 2 | 2 |
| ORL BUS, # data | Or immediate to BUS | 2 | 2 |
| MOVD A, P | Input expander port to A | 1 | 2 |
| MOVD P, A | Output A to expander port | 1 | 2 |
| ANLD P, A | And A to expander port | 1 | 2 |
| ORLD P, A | Or A to expander port | 1 | 2 |

**Branch**

| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| JMP addr | Jump unconditional | 2 | 2 |
| JMPP @A | Jump indirect | 1 | 2 |
| DJNZ R, addr | Decrement register and skip | 2 | 2 |
| JC addr | Jump on carry = 1 | 2 | 2 |
| JNC addr | Jump on carry = 0 | 2 | 2 |
| JZ addr | Jump on A zero | 2 | 2 |
| JNZ addr | Jump on A not zero | 2 | 2 |
| JT0 addr | Jump on T0 = 1 | 2 | 2 |
| JNT0 addr | Jump on T0 = 0 | 2 | 2 |
| JT1 addr | Jump on T1 = 1 | 2 | 2 |
| JNT1 addr | Jump on T1 = 0 | 2 | 2 |
| JF0 addr | Jump on F0 = 1 | 2 | 2 |
| JF1 addr | Jump on F1 = 1 | 2 | 2 |
| JTF addr | Jump on timer flag | 2 | 2 |
| JNI addr | Jump on INT = 0 | 2 | 2 |
| JBb addr | Jump on accumulator bit | 2 | 2 |

**Subroutine**

| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| CALL addr | Jump to subroutine | 2 | 2 |
| RET | Return | 1 | 2 |
| RETR | Return and restore status | 1 | 2 |

**Flags**

| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| CLR C | Clear carry | 1 | 1 |
| CPL C | Complement carry | 1 | 1 |
| CLR F0 | Clear flag 0 | 1 | 1 |
| CPL F0 | Complement flag 0 | 1 | 1 |
| CLR F1 | Clear flag 1 | 1 | 1 |
| CPL F1 | Complement flag 1 | 1 | 1 |

C1 – 10 mF electrolytic capacitor.
C2 – 22 mF tantalum capacitor.
C3 through C9 – 0.1 mF ceramic disc capacitor.
D1 – 1 amp silicon diode.
D2, D3 – 1N914 switching diode.
IC1 – 8035 CPU.
IC2, IC3 – 74LS373 octal tristate latch.
IC4, IC5 – 2716 2K x 8 read only memory.
IC6 – 74LS260 dual 5-input positive NOR gates.
IC7, IC8, IC18 – 74LS138 3-to-8 line decoders.
IC9 – 74LS32 quad 2-input positive OR gates.
IC10, IC11 – 2112 random access memories.
IC12 – 74LS86 quad 2-input exclusive or gates.
IC14 – 74LS00 quad 2-input positive NAND gates.
IC15 – 74LS74 dual D-type positive-edge-triggered flip flops.
IC16 – 74148 8-line-to3-line octal priority encoders.
IC17 – 74LS30 8-input positive NAND gate.
LED1 – 0.200 red light emitting diode.
R1 through R5 – 1000 ohms.
R6 – 4,700 ohms.
R7 – 330 ohms.
S1, S3 – SPST switch.
S2 – SPDT momentary switch.
XTAL – 1 MHz to 6 MHz crystal.
MISC – (6) 14 pin low profile IC sockets, (10) 16 pin low profile IC Sockets, (1) 4 pin low profile IC socket, (2) 28 pin low profile IC sockets, printed circuit board, wire, alligator clips, rubber feet, hardware, 5 VDC supply @ 1 amp, solder, tec.

**Figure 14:** Parts list for CD-1 MPB.

**Data Moves**

| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| MOV A, R | Move register to A | 1 | 1 |
| MOV A, @R | Move data memory to A | 1 | 1 |
| MOV A, # data | Move immediate to A | 2 | 2 |
| MOV R, A | Move A to register | 1 | 1 |
| MOV @R, A | Move A to data memory | 1 | 1 |
| MOV R, # data | Move immediate to register | 2 | 2 |
| MOV @R, # data | Move immediate to data memory | 2 | 2 |
| MOV A, PSW | Move PSW to A | 1 | 1 |
| MOV PSW, A | Move A to PSW | 1 | 1 |
| XCH A, R | Exchange A and register | 1 | 1 |
| XCH A, @R | Exchange A and data memory | 1 | 1 |
| XCHD A, @R | Exchange nibble of A and register | 1 | 1 |
| MOVX A, @R | Move external data memory to A | 1 | 2 |
| MOVX @R, A | Move A to external data memory | 1 | 2 |
| MOVP A, @A | Move to A from current page | 1 | 2 |
| MOVP3 A, @ | Move to A from page 3 | 1 | 2 |

**Control**

| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| EN I | Enable external interrupt | 1 | 1 |
| DIS I | Disable external interrupt | 1 | 1 |
| SEL RB0 | Select register bank 0 | 1 | 1 |
| SEL RB1 | Select register bank 1 | 1 | 1 |
| SEL MB0 | Select memory bank 0 | 1 | 1 |
| SEL MB1 | Select memory bank 1 | 1 | 1 |
| ENT0 CLK | Enable clock output on T0 | 1 | 1 |

| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| NOP | No operation | 1 | 1 |
| IDL | Select Idle Operation | 1 | 1 |

**Registers**

| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| INC R | Increment register | 1 | 1 |
| INC @R | Increment data memory | 1 | 1 |
| DEC R | Decrement register | 1 | 1 |

**Timer/Counter**

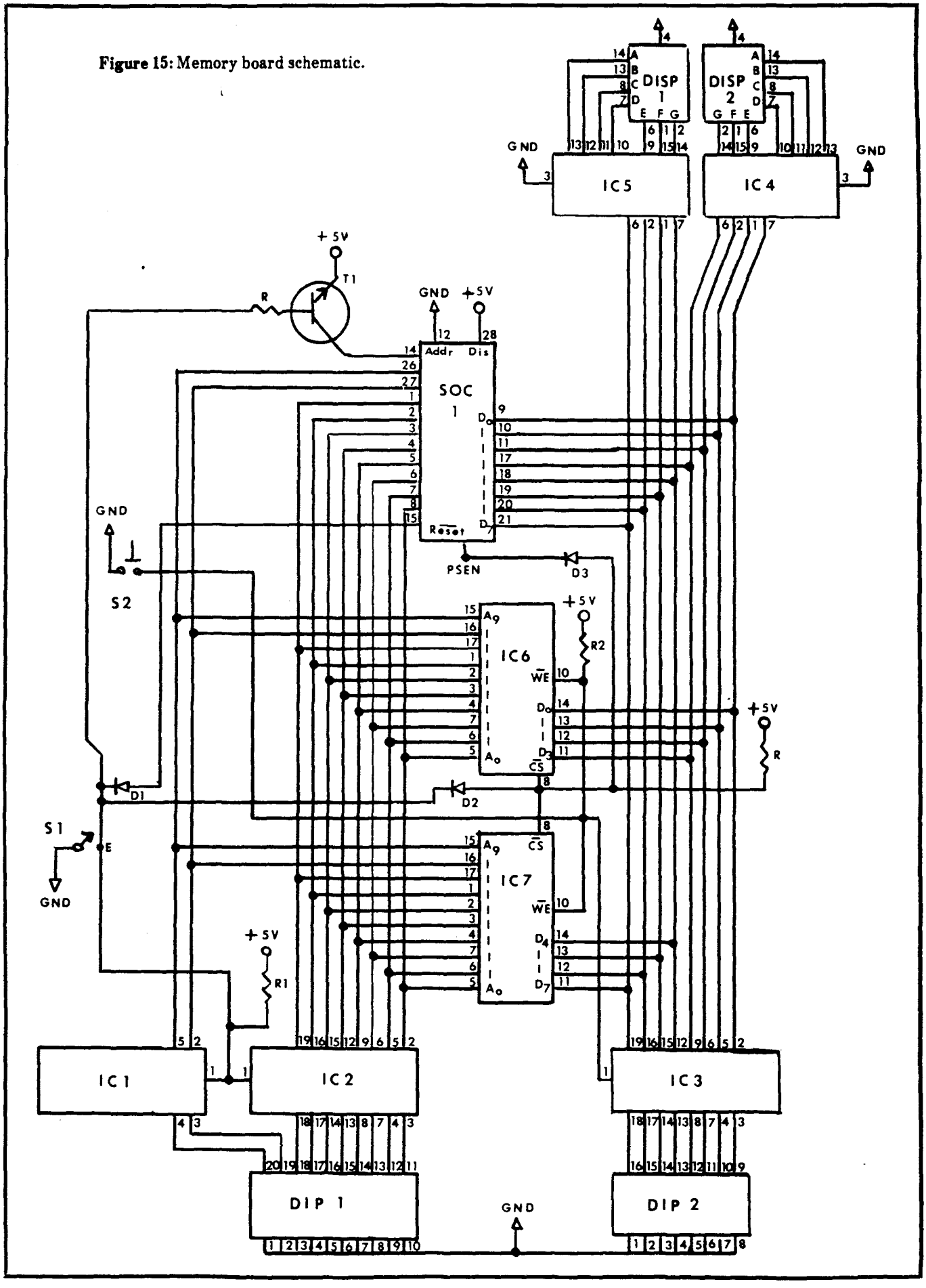
| Mnemonic | Description | Bytes | Cycles |
|---|---|---|---|
| MOV A, T | Read timer/counter | 1 | 1 |
| MOV T, A | Load timer/counter | 1 | 1 |
| STRT T | Start timer | 1 | 1 |
| STRT CNT | Start timer | 1 | 1 |
| STOP TCNT | Stop timer/counter | 1 | 1 |
| EN TCNTI | Enable timer/counter interrupt | 1 | 1 |
| DIS TCNTI | Disable timer/counter interrupt | 1 | 1 |

positives can also be supplied for a reasonable charge on request. The silk screen, which outlines each functional section of the CD-1, MPB, is especially useful for instructional purposes, and adds to the overall appearance of the board.

**Conclusion** As mentioned earlier, there may be times when you will want to write larger programs to be executed by the CD-1, MPB. Furthermore, you may want to purchase an 8035 assembler so that you can write the code in assembly instead of machine language. In the next issue of *The Computer Journal* I will present a device which will make these things

*(Continued on page 47)*

**Figure 15:** Memory board schematic.



# New Products

**ROMable MTBASIC Now Available**

Softaid, Inc. has released a new version of MTBASIC for OEMs and manufactures of board level computer products. The new version, MTBASIC*ROM can be burned into ROM and will execute from ROM. It is only available for Z-80 systems.

MTBASIC*ROM has all of the features of normal MTBASIC except for disk commands, since ROM based systems generally don't have a disk connected. Without the disk commands, MTBASIC*ROM will fit into 24K of ROM. A special version is available with disk commands, but it requires 30K of ROM.

Since every vendor's system has a different configuration, and CP/M® is usually not resident, MTBASIC*ROM is supplied on disk as .REL and .LIB files. A single assembly language module must be provided by the vendor to configure the I/O for the target system

There are no royalties for distributing MTBASIC*ROM in a board level product. MTBASIC*ROM is available for $1,100 from Softaid, Inc., PO Box 2412, Columbia, MD 21045, phone (301) 792-8096.

---

**Linear Optimizer**

The Acme Computer Company has announced release 2 of their Linear Optimizer program. Among the enhancements in the latest release are:

- Subscripted variables with up to 127 dimensions. Means for declaring array constants together with array operations and conditional constructs make defining large problems quick and easy.
- Input and output statements. Several file formats are supported, including the ubiquitous DIF format already used by dozens of commercial packages. Automatic dimensioning ("input until end of data") together with the conditional forms mentioned above make possible size independent matrix generators—perfect for imbedded applications.
- Report statements, which allow the user to include values in the final report that can be computed from the solution. This not only eliminates the need for additional constraints, it allows the display of nonlinear functions of the solution values.

Linear Optimizer with its high level matrix generator language makes it possible to create special applications that require no knowledge of linear programming by the user. Like spread sheet templates and data base applications, Linear Optimizer models add the essential elements to a useful, powerful package: programmability and integration.

Linear Optimizer is available for PC-DOS 2.0® , PC-DOS 3.0® , or CP/M 2.2® systems for $400. To order, or to request more information, contact Stephen Vestal at The Acme Computer Company, Box 51193, Seattle, WA 98115 phone (206) 522-6655.

---

**Programmable Microconrtoller**

Basicon now offers a 3 × 4" fully programmable microcontroller with 2K RAM and 40 I/O lines for less than $200.

With INTEGER BASIC as its resident programming language, the MC-1Z is a versatile controller for a wide range of applications, including OEM instrumentation, process control, research and development and even personal computing. It is based on the Zilog 8671 processor and can be upgraded to 16K RAM on the same board or with a 4K or 8K EPROM.

The MC-1Z is fully self-contained, easily programmed and comes with a variety of peripherals designed to maximize its utility. Included is a clock/calendar, two fast timer/counters, 6 interrupts and EPROM receptacle. Add cabling and an RS-232-C compatible terminal and you have a complete applications programmer with BAUD rates from 110 to 19,200. Complete hardware and software manuals are included, and the MC-1Z is available for shipment.

Contact Fred King at Basicon, Inc., 11895 NW Cornell Road, Portland, OR 97229 phone (503) 626-1012.

**Higher Sampling Speed for ADC-1**

Remote Measurement has announced an improved version of their ADC-1 Data Acquisition and Control System. The analog channel sampling rates have been increased for a choice of 7 to 15 samples per second to a choice of 20, 50 or 100 samples per second.

Application notes are available that provide users with Turbo Pascal® routines that can be used to acquire data at the speeds the ADC-1 is now capable of. They point out that BASIC programs can be still used, and even the BASIC programs will show some increase in speed with the new version.

The ADC-1 is compatible with any computer with an RS-232-C serial interface. The system provides sixteen analog input channels with a resolution of 12 bits plus sign (11 bits at high speed), four digital inputs, six hardwire controlled outputs, and the ability to operate BSR-type AC line-carrier remote control modules. The ADC-1 allows the user to concentrate on the design of the project and program, free from the details of electronic interfacing.

The host computer or modem sends commands to the ADC-1 in a single 8 bit byte, through a serial port. For each byte received, the ADC-1 returns an 8 bit byte that contains status and measurement data. The communications interface supports standard RS-232-C as well as the TTL levels used by Commodore and Atari serial ports. Baud rates are selectable from 300 to 9600 bps. Interconnection instructions and programming examples for most popular microcomputers are included in the owner's manual.

Sensitivity of the analog inputs is 0.1 mV ofer a range of ± 0.4 V (modifiable to ± 4.0 V). The A/D inputs are user-configurable for operation at 20, 50, of 100 samples per second. An optional instrumentation amplifier provides sensitivity at microvolt levels for use with thermocouples. The four digital inputs may be used to monitor or count on/off signals at up to 400 Hz, while the six controlled outputs can operate external electronic devices. Remote control of lamps and appliances is an additional capability of the ADC-1, via signals

transmitted over AC wiring to line-carrier remote control modules of the BSR X-10 system.

Single quantity price is $449, from Remote Measurement Systems, Inc., 2633 Eastlake Ave. East, Suite 206, Seattle, WA 98102 phone (206) 328-2255.

**Universal MAC INKER**

Computer Friends has announced the new Universal Cartridge Ribbon Re-Inker. Now, one Mac Inker re-inks just about any fabric ribbon cartridge available currently or in the future.

They claim that operation is extremely simple and wholly automatic, and that their new formula, lubricated, carbonless ink yields a printout darker than many original ribbons. They further state that a good quality ribbon can be re-inked up to 80-100 times, depending on its original length.

The Mac Inker can also be used to ink (and then re-ink) blank cartridges in any of 6 brilliant colors (or combinations thereof):Red, Blue, Green, Brown, Yellow, or Purple using the new Computer Friends ink.

The cost is $60 for the universal base and $8.50 for each cartridge driver kit, from Computer Friends, 6415 SW Canyon Ct., Suite 10, Portland, OR 97225, phone (503) 297-2321 or (800) 547-3303.

*Editor's Note: We have been using a MacInker for over a year to ink six to eight Epson MX-80 ribbons a month, and are would not want to do without it.*

**Basic-52 Computer/Controller**

The Micromint BASIC-52 Computer/Controller is a standalone single board microcomputer which needs only a power supply and terminal to become a complete system programmable in BASIC or machine language. The BASIC-52 uses the Intel 8052AH-BASIC microprocessor which contains a ROM resident 8K byte floating point BASIC interpreter. It contains sockets for up to 48K bytes of RAM/EPROM, an "intelligent" 2764/128 EPROM programmer, 3 parallel ports, a serial terminal port with auto baud rate selection, a serial printer port, and is bus compatible with the Micromint BCC11/BCC21 Z8 Systems/Controllers and BCC series expansion boards.

The BASIC-52 treats the EPROM as "write once" mass storage. When a BASIC application program is saved to EPROM, it is tagged with an identifying number and stored only in the amount of EPROM required to fit the program. Additional application programs can be stored to the same EPROM and recalled for execution by requesting a particular ROM number or designated as an autostart program.

Since the BASIC-52 is bus oriented, it supports Micromint's Term-mite ST smart terminal board, BCC14 memory expansion board, BCC33 I/O expansion board, BCC40 AC/DC I/O board, BCC13 A/D converter board, and BCC08 serial I/O expansion board.

The BASIC-52 Computer/Controller with 8K bytes of RAM is available for $239. For orders call 1-800-635-3355. For technical information or to request a data sheet call 1-203-871-6170.

*(Continued on page 50)*


INTRODUCING

CLOCKWORKS™

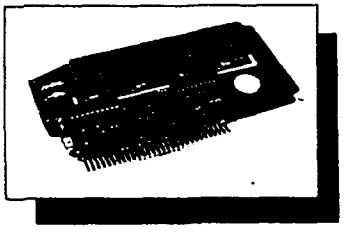
The real-time clock card for the Apple™ features:

- PRODOS/APPLEWORKS™ and DOS 3.3 compatible
- 24 hour and 12 hour AM/PM formats
- Time increments of 1 millisecond to 99 years
- Automatically time and date stamps your files
- Powerful on-board firmware in 4K EPROM
- High capacity LITHIUM® coincell battery
- Displays the date and time on Appleworks™ screen
- Eight BSR serial ports for future expansion
- Full documentation included in a users manual
- Includes more software on disk

$99

ALL PRODUCTS MADE IN USA
5 YEAR EXCLUSIVE WARRANTY
FREE SHIPPING CONTINENTAL USA/Limited Time Offer
VISA, MASTERCHARGE ACCEPTED

RAM 80e™

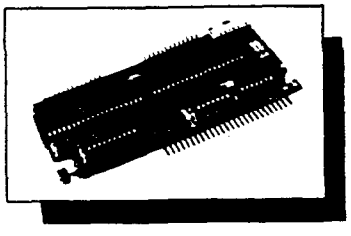


- Extends the IIe to 128K RAM
- Adds 80 column video display
- Enhances spreadsheet and word-processor viewing
- Allows double high resolution graphics

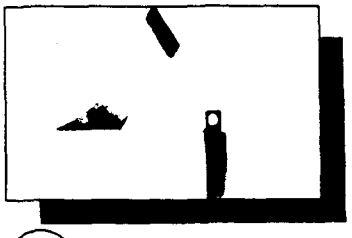
Easily installs in slot 3 of the Apple IIe™. Comes with full documentation. $79

MICROPORT 32™

An advanced digital I/O interface card for the Apple™. Provides four 8-bit ports and two additional lines for handshake per port. Has interrupt arbitration circuit, interrupt routing switches, and more. Excellent choice for monitoring and control applications. New low price $65.



EXTEND-50™



Reliably extends all 50 signals at internal slots to a 50 pin DIP. One side plugs into an Apple™ slot, the other plugs into any breadboard/protoboard. Now you can easily wire an interface card circuit without tedious wirewrapping or soldering techniques. A must for every designer. Only $34.

MSR Micro Systems Research 404/928-9394
4099 Maxanne Drive Kennesaw, Georgia 30144

## Editor

*(Continued from page 1)*

changes until it works. They also design languages which require many steps and considerable time between the writing of the code and execution of the program. While this approach may be acceptable for business related applications where the desired result is know in advance, it is not suitable for experimental work such as robotics or control where the program is a tool and not the final result.

What I would like is a programming tool (should it be called a language?) that is easy to program, generates small ROMable fast acting code, is interactive, and can be customized or extended by the user. I'm looking at this from the point of a user who has to program in order to accomplish their goal in a fast changing experimental situation instead of a programmer who is writing code for a canned program to be used by someone else. I haven't had the time to fully define what I want, but I'll list some initial thoughts hoping that you'll add your input. Perhaps we can generate enough interest to start a special user's group to work on this project.

I would like to be able to write sections in assembly language, add them to a library, and call only the required routines from a higher level language which I'll refer to as the "Handler". This sounds like a Macro Assembler, but I was thinking of the Handler as a high level language with built-in routines for the non-critical code sections. A possible candidate is C, and I have the BD Software C compiler to evaluate for this use. BDS C has the advantages of a small run time overhead, the ability to include only the library files (which can be either C or 8080 assembler HEX code) actually used, and it can produce ROMable code. The awkward part is that you have to use a word processor to generate (or change) the source code, then compile and link the program before you can run it. Then if you want to change a portion written in assembler, you have to rewrite the source using a word processor, assemble to a HEX file, put the new file in the library, and then compile and link the C program. I'm putting heavy emphasis on the ease of revising the program, and many people won't agree with this because they don't need it, but when working on a robotics application you'll want to make frequent changes in the sequence of motions, the speed, the position, the dwell time, etc. In these cases, an interactive BASIC-like language would be convenient if it was fast enough for real time control.

Another possibility for the Handler is MTBASIC which includes a flash compiler, has the ability to call machine language subroutines with argument passing, and can include in-line machine language code. You can even write a subroutine with CODE statements and then access it with a GOSUB.

The most likely of the current languages for the Handler is Forth with a special dictionary of machine language subroutines for the speed critical routines, but I can't seem to get started in Forth. The concept of the language is easy to understand, my problem is taking the time to learn to use the nonconventional editing, file saving, and user interface. It would help if there was a local Forth advocate who could take the time to get me started. Perhaps one of our readers would like to write an article on getting started with Forth – not a description of the language, but the mechanics of doing something with it.

APPLE ][, ][+,///,//e &//c OWNERS
UPGRADE THAT TIRED 6502 TO 16 BITS !!

65802 CPU $49.95
16 bit version of the 6502. Pin for pin and completely software compatible with the 6502 CPU. You can upgrade your Apple][,][+, ///,//e or//c to a 16 bit computer simply by replacing the 6502 with the 65802 without losing the ability to run any old software.

ProDOS ORCA/M (list $79.95) $69.95
This ProDOS version of ORCA/M comes with the complete 65802 instruction set. If you intend to develope software for this new CPU, then this package is a must. Chosen by the designers of the 65802 as the standard 65802 assembler.

16 Bit Upgrade Starters Package $109.95
This package includes 65802 CPU and the ProDOS ORCA/M. All you need to start.

Coming Soon: FORTH, PASCAL P-code Upgrade & MORE!

TO ORDER, SEND CHECK OR MONEY ORDER TO:
ALLIANCE COMPUTERS
PO BOX 408
CORONA, NY 11368

POSTAGE AND INSURANCE INCLUDED. (718) 426 2980
All CPU's will be sent by Postal Service, 1st class insured or UPS, insured. Please specify USPS or UPS. UPS doesn't deliver to POB's. Software will be sent by UPS Blue Label. If you want UPS Next Day Air, add $5.00 (CPU's only!). Most all orders sent out same day. COD add $3.00. APO's and FPO's welcomed.
Foreign orders: Please make payment in US dollars drawn on a US bank. Add $5 for Registered Mail and Air Mail Postage (except Canada). No foreign COD's.
PLEASE INCLUDE YOUR PHONE NUMBER WITH ORDER

**Surplus Parts Resource**

Here's a catalog any serious computer tinkerer needs. It's a treasure-trove of stepper motors, gear motors, bearings, gears, power supplies, lab items, parts and pieces of mechanical and electrical assemblies, science doo-dads, goofy things, plus project boxes, lamps, lights, switches, computer furniture, and stuff you might have never realized you needed.

*All at deep discounts cause they are surplus!*

Published every couple of months, and consecutive issues are completely different. Send $1.00 for next three issues.

JERRYCO, INC. 601 Linden Place, Evanston, Illinois 60202

One of the new developments is 'Structured Assembly Language Programming for the Z-80' from Hayden. Their approach is to use a large library (which they supply on disk) of macros for commonly used routines, and then write your specific routines in assembly. We have a copy of this for review, and will have more to say about it later.

**Turbo Pascal**

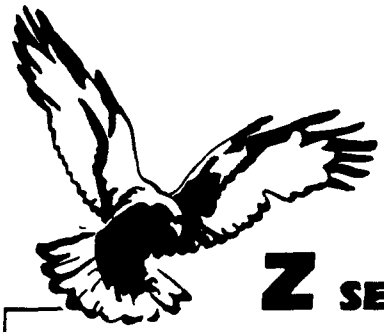
As you can tell from reading this, I've been doing a lot of thinking about languages—I haven't had time to do much programming, but I have been thinking about it while doing other things. When I first saw Pascal I decided that it had nothing to offer and I completely ignored it, but I recently tried Turbo Pascal from Borland and it changed my mind about Pascal. There has been a lot written about Turbo Pascal so I won't re-hash it, but you should be aware of its editor with WordStar like commands, ability to run programs from memory, compile directly to disk in one step, and a very helpful error detection and correction system.

Jerry Houston's article in this issue is the first of a series on Turbo Pascal, and we are interested in additional articles from other authors. We'd like to hear from you if you have an article in mind, and we'd also like to publish short notes, letters, information on public domain software, tips, routines, or questions on Turbo Pascal.

**CP/M is Dead, Long Live Z**

We have been told that Digital Research has classified their products A..D according to how much support the products will receive with D being the category with the least support. Concurrent CP/M is in group D and CP/M Plus is in group C, which means that in effect there is no support! Apparently they have also discontinued selling individual copies of uninstalled CP/M, and will only sell to hardware OEMs who will ship installed versions with their hardware.

How can you obtain an operating system for a Z-80 or HD64180 single board computer you design? You either pirate CP/M and end up with an illegal unsupported system, or you find another supplier. A better choice than CP/M (even if it was available) is the Z-System from Echelon, because it can do so much more than CP/M, and it is supported by people who care about YOU the user. It is time to change to an improved, up-to-date, supported system even if you already have CP/M up and running. Write to Frank at Echelon, 101 First Street, Suite 427, Los Altos, CA 94002 for information on the Z-System. And if you implement it for a new application, share the BIOS with other readers.

**Z SETS YOU FREE!**

**Z Operating System, an 8-bit OS that flies! Optimized HD64180/Z80 assembly language code — full software development system with proven linkable libraries of productive subroutines — relocating (ROM and RAM) macro assembler, linker, librarian, cross-reference table generator, debuggers, translators and disassemblers — ready to free you!**

High performance and flexibility! Productivity results from dynamically customized OS environments, matching operator, tasks and machine.

Real-time control kernel option allows quick software development for industrial control applications, other tools and utilities for office desk-top personal computing functions, local area networks to Ethernet, AppleTalk, Omninet, ArcNet, PC-Net (Sytek) — from micro to mainframe command, control and communications. Distributed processing application programs are easily developed.

- Extreme organizational flexibility, each directory another environment
- Multiple Commands per line
- Aliases (complex series of commands known by simple names) with variable passing
- Named Directories with absolute password security
- Full-screen command line editing with previous command recall and execution
- Shells and Menu Generators, with shell variables
- Command-file search Paths, dynamically alterable
- Screen-oriented file manipulation and automatic archiving and backup
- 512 megabyte file sizes, 8 gigabyte disks handled
- Auto disk reset when changing floppies
- TCAP database handles characteristics of over 50 computers and terminals, more easily added
- Tree-structured online help and documentation subsystem
- 76 syntax-compatible support utilities

**Your missing link has been found — Z! Now fly with eagles! Fast response, efficient resource utilization, link to rest of computing world — shop floor to executive suite, micro to corporate mainframe. Call 415/948-3820 for literature.**

**Echelon, Inc.** 101 First Street • Suite 427 • Los Altos, CA 94022 • 415/948-3820


**We Are The Pioneers of the Future**

What will computers be like five years from now?? Predicting the future for computers is very difficult. If you doubt it, check some of the forecasts made five years ago and see how off-base they were. I'll stick my neck out and make a forecast for the state of microcomputers in the year 1991 (I hope that you'll still be reading TCJ then). My prediction is that in the year 1991 microcomputers will be nothing like the ones we have today, and we have no basis on which to forecast what they will be because the technical features they will use are unknown (or at least not well known) and would not be feasible with current technology.

I realize that predicting that we can not make a prediction is a cop-out, so I'll try to forecast some trends or needs that should be met. One of the most obvious needs is for standardized easy to use hardware, operating systems, and software. Any reasonably intelligent person—someone who can use a telephone, a stero, or drive a car—should be able to use a computer

with a few minutes introduction. And they should be able to use any other computer from a different manufacturer without relearning anything. The development of the automobile is a good example of what will have to be done by the computer industry. In the beginning there were many companies making dissimilar products, but the market forced them to produce cars with standarized controls. Today we can get into a Ford, Chevy, Honda, Dodge, or Volkswagon, and travel to our destination over the same roads, using the same maps, stopping at the same gas stations, without having to be reeducated about how to operate the car. There are non-standard high performance vehicles which require special skills, but they are for the specialist and not the general public. My first prediction is that the unsophisticated user will be able to put their (I know that user is singular and their has been plural, but I refuse to use he/she or his/her so I am using their instead) program into any one of a large number of computers and run it using the same controls—they'll hardly know that they are using a different conputer.

Floppy disks are slow and easily damaged. Hard disks have more storage capacity and are faster, but they are not portable and are prone to catastrophic crashs. Some form of disk may be suitable for computer centers, but the normal user needs something which is non-mechanical, not easily damaged, fast, has very large capacity, and can be taken to another computer across the street or across the country. Optical write once disks sound like a temporary solution because they have large capacity and preserve the data because you do not write over the files, but I feel that the mechanical complexity and potential problems rule them out as a long term solution. My second prediction is that high capacity solid state non-volatile memory which does not require batteries will be in general use. Plug in cartridges will be used for program distribution, data back up, and data interchange, but the computer will have enough non-volatile memory to hold all the programs and data used by the average person so that they load the computer once and then just turn it on and use it without having to reload.

One of the current buzz-words is "user friendly", but programs are NOT easily used by the general public and a lot of work still remains to be done in this area. One of the problems is that people have been over-sold on what a computer can do. Small business owners who can not even balance their checkbook have been led to believe that if they buy a computer and an accounting program they can take care of their books. Accounting programs are a tool which enable someone capable of doing accounting to do it more easily and more accurately, but they don't enable someone with no training to set up and maintain their books. The disapointment caused by these unrealistic expectations (which are due to misleading advertising) are the primary reason for the current anti-computer backlash. It would be like me buying a scalpel today and expecting to able to perform brain surgery tomorrow just because I bought the same tool that a trained surgeon uses. My third prediction is that programs will be able to be used by their intended audience without any special computer training.

**What do YOU Predict?**

We are the pioneers of the future! The ideas for advancements will come from individuals who are not satisfied with what is available—people who can see a better way and figure out some way to accomplish it. I challenge you to send your predictions for publication in TCJ. ■


# TURBO AUTHORS WANTED!

**for the**

## Turbo Pascal Handbook

**Know a good set of Turbo routines? Perhaps you have developed your own libraries. What do you know about using Turbo to manage assembly language libraries? Operating system calls? I/O? Write about it! The Turbo pascal handbook needs your article-now.**

*Write to* The Computer Journal *for more information and an author's guide.*

**The Computer Journal**

190 Sullivan Crossroad, Columbia Falls, MT 59912

Phone (406) 257-9119


**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, Macintosh, DOS 3.3, ProDOS; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. MBASIC; Microsoft. Wordstar; MicroPro International Corp. IBM-PC, XT, and AT; IBM Corporation. Z-80, Zilog. MT-BASIC, Softaid, Inc. Turbo Pascal, Borland International.

Where these terms (and others) are used in The Computer Journal they are acknowledged to be the property of the respective companies even if not specifically mentioned in each occurence.

**Circuit Designer**

*(Continued from page 40)*

**Memory Board Parts List**

D1, D2, D3 – 1N914 switching diode.
DIP 1, DIP 2 – 8 Position dip switch.
DISP 1, DISP 2 – MAN 704 common cathode display.
IC1, IC2, IC3 – 74LS373 octal tristate latch.
IC4, IC5 – F9368 hexidecimal decoder.
IC6, IC7 – 2114 1K by 4 random access memory.
R1, R2, R3 – 1K ¼ W resistor.
R4 – 4.7K ¼ W resistor.
S1 – SPST switch.
S2 – SPST momentary switch.
SOC 1 – 28 pin I.C. socket.
T1 – 2n3904 silicone transistor.

**Figure 16:** Parts list for Memory Board.

possible. The device which I am referring to is a 2716 EPROM emulator. The 2716 EPROM emulator is actually a read/write memory (with a 2716 EPROM pinout) connected to the serial port of any personal computer through a standard UART. In addition to programming the CD-1, MPB, you will find that the EPROM emulator has many other uses. ■


OVERCOME
**FORTRAN | PASCAL | C**
LIMITATIONS WITH

$129 NO LICENSE FEE

NO LIMIT

Ver. 2.0 For:
MS FORTRAN/Pascal/C
IBM Professional and R-M FORTRAN
SuperSoft FORTRAN

A library of over 90 Assembler routines transform FORTRAN, Pascal and C language compilers into the flexible, responsive, complete languages needed for the microcomputer environment. Hundreds of NO LIMIT owners are creating highly interactive software systems, often utilizing existing mainframe code and saving time and money in the process. Ver. 2.0 features include:

| | |
|---|---|
| EXTENSIVE GRAPHICS | (Get, Put, Paint, Color, Dot, Line, Box, Circle, Ellipse, Large Characters) |
| FULL SCREEN CONTROL | (Windows, Cursor, Read/Write Screen) |
| STRING MANIPULATION | (Match, Compare, Concatenate/Extract, Pack, Justify, Zero Fill) |
| KEYBOARD CONTROL | (Read Key During Execution, String Read With Edit) |
| FILE MANAGEMENT | (Exist?, Rename, Delete) |
| COMMUNICATIONS | (Full Interrupt Driven to 9600 Baud, Set Com Line, Send/Receive, Line/Modem Status) |
| INTERRUPTS | DOS Execution, Program Execution |
| OTHER FEATURES | (Command Line Read, DOS 3.0 Directories, Peek, Poke, Random Numbers, System Status, etc.) |

And to complement NO LIMIT, the I/O PRO development system allows creation and editing of FORTRAN/Pascal/C callable screens. This word processor type system pays for itself several times over on your first project. $260.

For immediate solutions to your programming needs call
**(800) 562-9700**
**(512) 251-5543 (Texas)**
**M | E | F Environmental Inc.**
P.O. Box 26537 Austin, Texas 78755
© Trademark of Microsoft Inc., Supersoft Inc., IBM Corporation, Ryan-McFarland and Microsoft Inc., respectively.

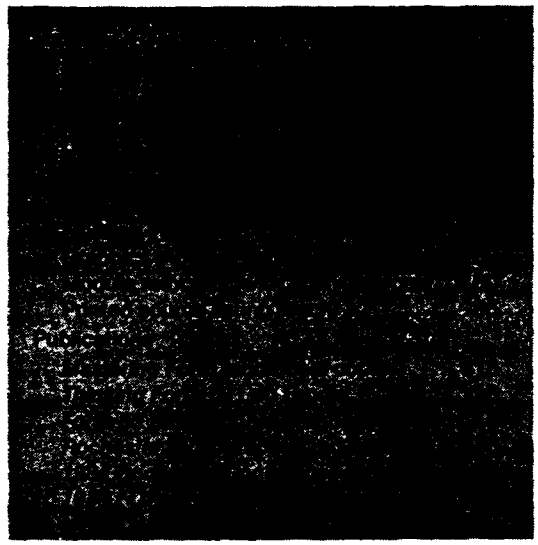


Portable Forth

Whether you program on the **Macintosh**, the **IBM PC**, an **Apple II** series, a **CP/M** system, or the **Commodore 64**, your program will run unchanged on all the rest. If you write for yourself, MasterFORTH will protect your investment. If you write for others, it will expand your marketplace.

MasterFORTH is a state-of-the-art implementation of the Forth computer language. Forth is interactive – you have immediate feedback as you program, every step of the way. Forth is fast, too, and you can use its built-in macro assembler to make it even faster. MasterFORTH's relocatable utilities, transient definitions, and headerless code let you pack a lot more program into your memory. The resident debugger lets you decompile, breakpoint, and trace your way through most programming problems. A string package, file interface, and full screen editor are all standard features.

MasterFORTH exactly matches the Forth-83 Standard dialect described in *Mastering Forth* by Anderson and Tracy (Brady, 1984). The standard package includes the book and over 100 pages of supplementary documentation.



MICROMOTION
12077 Wilshire Blvd., #506

# Back Issues Available:

**Volume 1, Number 1 (Issue #1):**
- *The RS-232-C Serial Interface, Part One*
- *Telecomputing with the Apple][: Transferring Binary Files*
- *Beginner's Column, Part One: Getting Started*
- *Build an "Epram"*

**Volume 1, Number 2 (Issue #2):**
- *File Transfer Programs for CP/M*
- *The RS-232-C Serial Interface, Part Two*
- *Build a Hardware Print Spooler, Part One: Background and Design*
- *A Review of Floppy Disk Formats*
- *Sending Morse Code With an Apple][*
- *Beginner's Column, Part Two: Basic Concepts and Formulas in Electronics*

**Volume 1, Number 3 (Issue #3):**
- *Add an 8087 Math Chip to Your Dual Processor Board*
- *Build an A/D Converter for the Apple][*
- *ASCII Reference Chart*
- *Modems for Micros*
- *The CP/M Operating System*
- *Build a Hardware Print Spooler, Part Two: Construction*

**Volume 1, Number 4 (Issue #4):**
- *Optoelectronics, Part One: Detecting, Generating, and Using Light in Electronics*
- *Multi-user: An Introduction*
- *Making the CP/M User Function More Useful*
- *Build a Hardware Print Spooler, Part Three: Enhancements*
- *Beginner's Column, Part Three: Power Supply Design*

**Volume 2, Number 1 (Issue #5):**
- *Optoelectronics, Part Two: Practical Applications*
- *Multi-user: Multi-Processor Systems*
- *True RMS Measurements*
- *Gemini-10X: Modifications to Allow both Serial and Parallel Operation*

**Volume 2, Number 2 (Issue #6):**
- *Build a High Resolution S-100 Graphics Board, Part One: Video Displays*
- *System Integration, Part One: Selecting System Components*
- *Optoelectronics, Part Three: Fiber Optics*
- *Controlling DC Motors*
- *Multi-User: Local Area Networks*
- *DC Motor Applications*

**Volume 2, Number 3 (Issue #7):**
- *Heuristic Search in Hi-Q*
- *Build a High-Resolution S-100 Graphics Board, Part Two: Theory of Operation*
- *Multi-user: Etherseries*
- *System Integration, Part Two: Disk Controllers and CP/M 2.2 System Generation*

**Volume 2, Number 4 (Issue #8):**
- *Build a VIC-20 EPROM Programmer*
- *Multi-user: CP/Net*
- *Build a High-Resolution S-100 Graphics Board, Part Three: Construction*
- *System Integration, Part Three: CP/M 3.0*
- *Linear Optimization with Micros*
- *LSTTL Reference Chart*

**Volume 2, Number 5 (Issue #9):**
- *Threaded Interpretive Language, Part One: Introduction and Elementary Routines*
- *Interfacing Tips and Troubles: DC to DC Converters*
- *Multi-user: C-NET*
- *Reading PCDOS Diskettes with the Morrow Micro Decision*
- *LSTTL Reference Chart*
- *DOS Wars*
- *Build a Code Photoreader*

**Volume 2, Number 6 (Issue #10):**
- *The FORTH Language: A Learner's Perspective*
- *An Affordable Graphics Tablet for the Apple ][*
- *Interfacing Tips and Troubles: Noise Problems, Part One*
- *LSTTL Reference Chart*
- *Multi-user: Some Generic Components and Techniques*
- *Write Your Own Threaded Language, Part Two: Input-Output Routines and Dictionary Management*
- *Make a Simple TTL Logic Tester*

**Volume 2, Number 7 (Issue #11):**
- *Putting the CP/M IOBYTE To Work*
- *Write Your Own Threaded Language, Part Three: Secondary Words*
- *Interfacing Tips and Troubles: Noise Problems, Part Two*
- *Build a 68008 CPU Board For the S-100 Bus*
- *Writing and Evaluating Documentation*
- *Electronic Dial Indicator: A Reader Design Project*

**Volume 2, Number 8 (Issue #12):**
- *Tricks of the Trade: Installing New I/O Drivers in a BIOS*
- *Write Your Own Threaded Language, Part Four: Conclusion*
- *Interfacing Tips and Troubles: Noise Problems, Part Three*
- *Multi-user: Cables and Topology*
- *LSTTL Reference Chart*

**Volume 2, Number 9 (Issue #13):**
- *Controlling the Apple Disk ][ Stepper Motor*
- *Interfacing Tips and Troubles: Interfacing the Sinclair Computers, Part One*
- *RPM vs ZCPR: A Comparison of Two CP/M Enhancements*
- *AC Circuit Anaysis on a Micro*
- *BASE: Part One in a Series on How to Design and Write Your Own Database*
- *Understanding System Design: CPU, Memory, and I/O*

***Issue Number 14:***
- *Hardware Tricks*
- *Controlling the Hayes Micromodem II From Assembly Language*
- *S-100 8 to 16 Bit RAM Conversion*
- *Time-Frequency Domain Analysis*
- *BASE: Part Two*
- *Interfacing Tips and Troubles: Interfacing the Sinclair Computers, Part Two*

***Issue Number 15:***
- *Interfacing the 6522 to the Apple ][ and ][e*
- *Interfacing Tips and Troubles: Building a Poor-Man's Logic Analyzer*
- *Controlling the Hayes Micromodem II From Assembly Language, Part Two*
- *The State of the Industry*
- *Lowering Power Consumption in 8" Floppy Disk Drives*
- *BASE: Part Three*

***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 and Troubles: Build a "Jellybean" Logic-to-RS232 Converter*

***Issue Number 17:***
- *Poor Man's Distributed Processing*
- *Base: Part Five*
- *FAX-64:Facsimile Pictures on a Micro*
- *The Computer Corner*
- *Interfacing Tips and Troubles: Memory Mapped I/O on the ZX81*

***Issue Number 18:***
- *Interfacing the Apple II: Parallel interface for the game port.*
- *The Hacker's MAC: A letter from Lee Felsenstein*
- *S-100 Graphics Screen Dump*
- *The LS-100 Disk Simulator Kit: A product review.*
- *BASE: Part Six*
- *Interfacing Tips & Troubles: Communicating with Telephone Tone Control*
- *The Computer Corner*

**Issue Number 19:**
- Using The Extensibility of FORTH
- Extended CBIOS
- A $500 Superbrain Computer
- Base: Part Seven
- Interfacing Tips & Troubles: Part Two
Communicating with Telephone Tone Control
- *Multitasking and Windows with CP/M:* A review of MTBASIC
- The Computer Corner

**Issue Number 20:**
- Build the Circuit Designer 1 MPB: Designing a 8035 SBC
- Using Apple II 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
- The Computer Corner

**Ordering Information:** Back issues are $3.25 in the U.S. and Canada. Send payment with your complete name and address to The Computer Journal, 190 Sullivan Crossroad, Columbia Falls, MT 59912. Allow 3 to 4 weeks for delivery.

# Classified

Classified ads are 25 cents per word. All Classified ads must be paid in advance, and will be published in the next available issue. No checking copies of proofs are supplies, so please type your ad or print legibly.

**KEYBOARDS FOR COMPUTER BUILDERS.** Full ASCII, numeric pad, UC/lc, CAPS-LOCK, REPEAT, SELF-TEST! Brand new, hundreds sold to builders of Apples, Xerox 820s, Big Boards, etc. Parallel TTL output, strobe. 5 volts/100 ma. Custom case available. Keyboard $35. Documentation (21 pgs.)/cable pkg. $5. Spare CPU/ROM $4. UPS included. Detailed specs on request. **Electrovalue Industrial Inc.**, Box 376-CJ, Morris Plains, NJ 07950. (201)-267-1117.

**Voice Processor for the KAYPRO Computer.** Unlimited speech contains all software. Call or write **Busch Computer**, PO Box 412, West Haven, CT 06516. Phone (203)484-0320.

**S-100 68008 CPU BOARD.** Detailed description in issue 16 of The Computer Journal. A&T $260, Kit $210, Bare Board $65. Prices include shipping. **INTELLICOMP, INC.**, 292 Lambourne Ave., Worthington, OH 43085, Phone (614)846-0216 after 6 p.m.

**Morrow Decision I** S-100 system with MPZ-80 CPU, DJ/DMA floppy disk controller, 256K static ram, Wonderbus I/O on mother board, Disk Jockey Hard Disk (HDCA) Controller, Sugart 800 floppy drive, 10MB hard disk, CP/M, Micronix Multiuser system, unconfigured MP/MII, dBase II, Wordstar, Accounting Plus. Excellent condition. $3500, some trades considered. TCJ, 190 Sullivan Crossroad, Columbia Falls, MT 59912 Phone (406)257-9119.

**S-100 Bus IEEE-488 Interface Card** with cable, manuals, and software for North Star Horizon. Purchased new from Pickles and Trout in 1979 and used once. $100. Call **Phil Wells** at (406)755-1323 days or (406) 844-3240 evenings.

**Corvus 10MB Hard Disk** for the Apple II plus, $888.00; Apple III Second Disk Drive, $199.00; Apple III SOS Device Driver Writer's Guide $19.95; BPI Accounting for Apple III (Requires Hard Disk) $99.00; Apple Writer 1.1, 16 Sector, $8.88; Apple DOS User's Manual (II, II plus, IIe), $8.88; Apple DOS Programmer's Manual (II, II plus, IIe), $6.88; KAYPRO-Home Accountant by Continental, $49.00; Soroc IQ 130 Terminal, $399.00. All plus shipping. **The Computer Place**, 36 2nd Street East, Kalispell, MT 59901, Phone (406)755-1323.

**THE SECURITY DISK.** PROTECTED VS UNPROTECTED. At last, the best of both worlds. Here is software designed to PROTECT YOUR PRIVATE FILES. SIMPLE PROTECTION TO MULTI-LEVEL CRYPTOLOGY. Plus "DYSUN::THE DISK UNLOCKER" WILL ANALYZE & UNLOCK COMMERCIAL "COPY-PROTECTED" DISKS, then recopy them to standard DOS 3.3 format. "DYSUN" will also RECOVER LOST DATA & REPAIR BLOWN DISKS. A special "SECRETS-TIPS-TECHNIQUES" program is also included. Not locked-up. Listable. Machine Language Source Codes included. Supports Apple II, II Plus, IIc, IIe. To order send $29.95 CHECK/MO to **B.M.E. Enterprises**, Box 191-J, Kila, MT 59920.

**Book Sale** — These books are offered at this price while the supply lasts.
Zilog Z80-CPU Technical Manual, $1.50
CBASIC Users Guide
by Osborne, Eubanks, and McNiff, $14.00
Introduction to FORTH
by Ken Knecht, $9.00
FORTH Programming
by Leo J. Scanlon, $13.00
These prices are postpaid in the U.S. only. TCJ, 190 Sullivan Crossroad, Columbia Falls, MT 59912.

**COMMODORE 64 INTERFACING BLUE BOOK.** Build 36 useful interfaces for your computer. 196 pages. Complete, detailed information. Satisfaction guaranteed. $16.95 postpaid. **Microsignal Press**, PO Box 388, Goleta, CA 93116.

**CMOS PROTOYTPING SYSTEM.** Uses NSC-800 I.C. for Z-80 Compatibility, 4 STD-Bus Cards (Dual Serial I/O, CPU, 32K Static Memory, 24-bit Parallel I/O), battery powered 6-slot card cage, and standalone debugging monitor. Mates with Industrial I/O Racks (Opto22, DuTech, etc.) Complete package only $795 plus S&H, originally cost over $1200. Brochure sent on request. **Input Video**, PO Box 20, Randolph, MA 02368, phone (617) 961-4197.

**STD-Bus Cards.** Wide assortment of STD-Bus cards, CPU,s, Memories (Static & Dynamic types), Serial I/Os, CRT controllers, etc. Quantities limited, send SASE for complete listing and prices. **Input Video**, PO Box 20, Randolph, MA 02368, phone (617) 961-4197.

## Advertiser's Index


Alliance Computers..... 44
Apropos..... 26
Artec..... 17
Barnes Research..... 28
BD Software..... 7
John Bell..... 31
Bersearch..... 33
Blankenship Basic..... 21
BV Engineering..... 12
Classifieds..... 49
Computer Trader..... 39
Digital Images..... 30
Echelon, Inc..... 13, 45
Intellicomp..... 18
Jerryco..... 44
Lawson Labs..... 50
MEF..... 47
Miller Microcomputer Services..... 22
Micromotion..... 47
Micro Systems Research..... 43
Next Generation Systems..... 20
Poor Person Software..... 26
Public Domain Software..... 39
Remote Measurement..... 4
SC Software..... 34
SLR Systems..... 15
Softaid..... 10
Southern Pacific..... 37
Turbo Pascal Handbook..... 46


## New Products

*(Continued from page 43)*

### HD64180 Communications Software

Echelon, Inc. announces further support for Hitachi's HD64180 high-integration 8-bit microprocessor chip with release of TERM III, next generation communications software package which offers the following features.

- Interactive communications with remote computer systems.
- Reliable file transfer between the user's computer (host) and a remote computer.
- Control of an auto-dial/auto-answer modem.
- Access control for remote system applications (the user's system can be set up to be dialed into by outside users).
- Rapid and easy reconfiguration of the communications environment for different applications.

Term III was designed to meet the communications needs of the serious Z-System user, and knowledge of that operating system is required to make effective use of the combination. The program works with ZCPR3 and Z-System and runs on Z-80, NSC800 and HD64180 based microcomputers. Three books, *ZCRP3: The Manual, ZCPR3 and IOPs, and ZRDOS Programmer's Manual*, provide detail descriptions of all Z-System functions.

TERM III is priced at $99, and is available with the other Z-System items from Echelon, Inc., 101 First Street, Los Altos, CA 94022, phone (415) 948-3820.

### Technical Engineering BASIC

TransEra has announced TBASIC, a new technical BASIC for CAD, scientific, and engineering applications with emphasis on graphics and instrument control. They state that it is the only language for PC's with built-in GPIB syntax for both graphic and instrument type peripherals. It permits easy adaption of software written for HP and Tektronix engineering computers to the PC environment. TBASIC's enhanced instruction set conforms to the new ANSI proposed standard for BASIC and runs on many popular operating systems including: PC-DOS, MS-DOS, CP/M, UNIX, and VMS. Some of the supported computers include: IBM PC/XT/AT, AT&T 6300, HP-150 & Intergral, Tektronix 6130 & 4170, and DEC VAX and Rainbow.

TBASIC has a unique assortment of user-friendly features such as an on-line HELP facility, cross-reference facilities for listing variables and referenced line numbers, a command for renaming program variables that aids in documenting new and existing programs by making simple variable names more meaningful, MOVE and COPY commands that enhance program editing capability by assigning new line numbers intelligently, and syntax checking that is performed automatically as lines are entered to give instant feedback.

TBASIC features a full compliment of binary, scalar, array, matrix, and scientific math, including standard trig, transcendental, and other functions using both integer and double precision floating point data types. It also supports 8087 coprocessors. Special array functions perform area, circumference, centroid and other calculations for polygons. Other features include line labels, block IF statements, true sub

*(Continued on page 51)*


**DON'T PAY $700 FOR DATA ACQUISITION SOFTWARE**

We offer full-featured, reasonably priced data logging software. Our software is well documented, unprotected and fully listable.

64 - CHANNEL CAPACITY DATA LOGGING PACKAGE for IBM PC® ..... $150

128 - CHANNEL CAPACITY DATA LOGGING PACKAGE for APPLE II® ..... $100

HARDWARE SUPPORTED:

IBM: Model 140 15-bit A/D 7.5 sam/sec ..... $265

APPLE: Model 34 12-bit A/D 40,000 sam/sec ..... $325
Model 38 8-bit A/D 111,000 sam/sec .... $165
Model 14 13-bit A/D 15 sam/sec ..... $175
Model 40 Timer/Clock 1 ms resolution ..... $175

BOTH: Model 20 Differential Thermocouple Thermometer with true cold junction compensation ..... $175
Model 17 16-channel differential Multiplexer ... $165

**LAWSON LABS, INC.**
**5700 Raibe Road**
**Columbia Falls, MT 59912**
Phone: 406 387-5355

problem if you haven't assembled your own Forth compiler. While figuring out how it is done, I also considered writing or designing my own language as the structure and concept are pretty simple. Forth is most confusing only due to the use of the cryptic words. For those, like myself, who mostly do assembly work, we have already learned all the three letter codes that represent data movements in assembly. Forth words are no different than these, and the comparison can be used to help understand how it all works.

Suppose we want to get data to a terminal (in assembly this is usually called CONOUT). The routine would first check to see if the device is ready, then get the data and output it to the device. Next it would flag to indicate completion, then return to the main program. The way Forth works is to break each individual operation into words (think of MACROS). Our assembly program would then be CONOUT: TEST GETDAT OUTDAT FLAG RETURN. When assembling the macro program, the assembler would actually write the code and assemble it as if you coded it by hand. Forth, however, would assemble the addresses of the machine code portions that perform the functions and then CALL them in sequence. The difference in code sizes is due to the fact that Forth only assembles the machine code once and then references that location there after. The macro assembler, however, would insert machine code whenever the routine was used. Suppose that CONOUT used 25 machine code instructions and our program called it 10 times. If one call instruction is considered as one instruction, then the Forth program would be 25 + 10 for a total of 35 machine instructions compared to the macro's 250 instructions (10 times 25). The macro version might actually be faster without all the jumping, but the Forth version would be much smaller. Optimizing the Macro code would have you calling the routine where the code was placed instead of repeating the code every time it was used. The size of the optimized Macro code would be the same as the Forth code, but faster than Forth's threading.

I have been considering creating a language based on Forth's threaded features but using the microprocessor type architecture. We could take the Z80 assembly code and call this the language, with the kernal being actual Z80 machine instructions. For use in other machines, simulation of the Z80 instructions in the kernal is all that is needed. Think about it, what it means is being able to disassemble a program (or use a PROGRAM.ASM file), load the program into Z80 Forth and have it run directly on a 68000 machine! In reality I am borrowing Moore's concept of a universal design except that mine would be register oriented instead of stack oriented. All other features would be the same.

Enough speculation, I'll have more on the ROMing of Forth later and will see how many people can find errors with my idea of a new language.

**Televideo TPC-I**

A recent release of the discontinued Televideo protable Z80 computers, called TPC-I, at $795 has caught my attention. I purchased one for my mother-in-law to use at college and decided to check it out. The case is all plastic, so getting into it was a big chore. It all snaps together even though there are screws (mainly for looks). A board that covers the whole end of the unit has the computer. It has two Teac FD55B drives, a nice 9 inch monitor and 64K of memory expandable to 128K. The disk controller is a 1770 (5 inch only), and there is an R6545 video controller for the GEM package. There is a Z80, with several special chips handling unknown functions (no schematic provided). My unit is clean with only one jumper on the board (rev. B1) and it physically looks good except for the case. Other than adding more memory I would guess the unit could not be changed or modified (no adding 8 inch drives!).

If you are not concerned about the hardware, the units seem to work pretty well and come with a full complement of software. I am not necessarily recommending that you buy one of these, but only letting you know that they work, appear to be made OK, but will not let you expand on them, which is what I always need to do.

**Portapac**

What I can recommend for those who carry data from one place to another is the Portapac by Cryptronics (11711 Coley River Circle, Suite 7, Fountain Valley, CA 92708, phone 714-540-1174). This rather small $500 unit can intercept serial data and store it in memory for playing back later or elsewhere. We bought this unit to help with an industrial problem, mainly slow and troublesome tape cassettes. We needed some way of taking data from one location, where the controller was, and bring it to our shop where a regular computer could debug it and make changes.

Now that we have the unit in our hands, all sorts of uses come to mind. It can fit in between a computer and a terminal to trap data for later use (could prove useful with mainframes that will not talk to other computers). The units also can help with changes in baud and protocol operations by allowing you to store the data (the unit has battery backup), change formats, and play the data back. One use I can think of is in a busy office where the printer is located in a quiet area and the computers are all over the office. You could get one of these for everybody and just have them take it to the printer for use, no messy or long cables going to all systems. Also let's not forget the protocol problems (most LAN's will cost more per unit than this). The Portapac can handle up to 256K of memory (ours came with 32K and can be expanded to 64K by adding 6264LP-15's). It looks rather simple inside and well built, so give Cryptronics a call if this fits any of your needs. I may build one for myself along similar lines and if so will write it up in *The Computer Journal.* ■

---

*(Continued from page 50)*

programs, DO/FOR loops with "clean" exits, long variable names, multi-statement lines, dynamic dimensioning, a powerful assortment of string commands, and more.

Demonstration Disks are available with a full 30 day money back guarantee. Complete documentation is provided including a modern Keyword Dictionary and User's Guide. The cost is $495 for micro-operating systems (CP/M, MS-DOS, and PC-DOS), $795 for most UNIX workstations, and $1995 for multi-user minis such as VAX. For further information, call (801) 224-6550, or write to: TransEra Corp., 3707 North Canyon Road, Provo, UT 84604. ■

# THE COMPUTER CORNER

## A Column by Bill Kibler

I made it back alive from the SOG (Semi-Official Get-together sponsored by Micro Cornucopia) in Bend, Oregon. This beautiful recreation area of central Oregon was the site for a get-together that brought in several people from outside the U.S. At the barbeque on Thursday night, a showing of hands made it clear that the largest part of the attendees were from east of the Rocky Mountains (rather impressive for a free event).

Things got started Friday with seminars and product displays. About 300 people were there Friday with more people expected on Saturday and Sunday (those poor working types). While we were only able to attend the Friday stuff, there is no question that all three days were a success. Some of the speakers had a lot of information to give (even though the titles of seminars were a bit cryptic). Only a few of the seminars had conflicting times so you could stop in anyway to see if one fit your needs. Jack Denton, the author of RPM (which I reviewed some time ago), started the session with an overview of CP/M which really turned out to be a discussion of the known bugs in CP/M and how to correct them. For us system type programmers, this was great. Later on there were talks on OS9, a panel on 32 bit systems, and on, and on. A sample of what you missed if you weren't there was the new Z80 replacement (Hitachi's HD64180).

### Z80 Plus

It was most unfortunate that the speaker at the SOG had only been in the United States for THREE days and his english therefore, was rather poor. This talk has since been supplemented with documentation, which makes this new chip by Hitachi, the HD64180, a real plus to the 8 bit world. The unit is capable of addressing 512K of memory, has two serial ports, a timer, DMA functions and high speed clocked I/O for chip to chip communications. All this is placed on top of the Z80 architecture and in CMOS for higher speeds (10MHz possible, 8MHz nominal, 6MHz now!) and less power consumption.

The speaker was representing the MSC BOARDS imported by Southern Pacific Limited (Box 4427, Berkeley, CA 94704-0427). This company has produced a board having 512K of memory, serial and parallel ports, disk controller, video controller, and HD64180, all on a Five inch disk drive size board (same as AMPRO). As if the size and features were not enough, the price was even better—$400, and all available in a month or so. If this product doesn't get Zilog to produce their long awaited Z800, nothing will.

The HD64180 has a few new instructions, including an 8-bit multiply, a sleep command, and some special I/O block moves. Add this to their extensions of the bit test functions, and you have a faster Z80 machine. The MMU (memory management unit) part of the HD64180 is pretty standard and will make it possible to program any 4K segment ( of the 512K space) into the 64K operating system space. This is done by loading registers with offset values, with the necessary register selection based on the logical upper four bits (12 thru 15). The lower 12 bits are added with the MMU registers (7 bits) and the upper logical 4 bits to create the physical address. The preliminary data sheets didn't have the actual code needed to do this, so when I get the big manual, I will cover more of the features in greater detail then.

The Main drawback of the unit is the 64 pin socket. There will most likely be a shortage of these at the start, so using the chip will require headers at first. A project that I will undertake when more information is available (and the units are for sale here in the U.S. at a reasonable price) will be an adaptor for Z80 machines to make use of the new features (such as extended memory addressing). I am thinking of some kind of paddle board that will have a Z80 header, a socket for the HD64180, jumpers for extended memory (or sockets for more memory), and serial buffers to make use of the extra serial ports. This shouldn't require more than five or six components (not including memory). The biggest problem will be getting the HD64180 and enough information to make it work.

### Forth ROM

An ongoing project of mine is putting Forth in ROM, and I have some information on doing that. It didn't take me long to understand Forth (although learning all the words may take forever), but seeing how it works in real code took some investigation. Now I haven't figured everything out yet, but the way it is put together is quite nice. It reminds me of an article I published in my local club's newsletter about writing your own language. The idea was that a good macro assembler could be used to create enough macros to do all your functions. Writing a program would involve calling only macros and no actual code. Forth is similar to that idea with some other features, and of course the threading is different.

I will leave threading for a later date but point out some important thoughts now. From what I have read about Moore (the author of Forth), it would appear that he based the design of the language on an HP calculator. What he did was to create a processor architecture in software. The original work was done on big mainframes and all his early calculations were probably being done by hand on an RPN calculator (Reverse Polish Notation machine). My biggest problem with Forth is the RPN logic, which is a carry-over from the early calculators. The idea, however, of making a processor in software is pretty terrific as it can be transported to any physical system. The actual codes to create the basics of the design (stacks and math functions) are done in the main kernal, with more advanced functions calling these routines.

The ROM code of course doesn't have any buffer spaces inside the ROM. Other than that, it would be the same as regular Forth code. While this concept is pretty well known, the actual putting of code into ROM can still be a