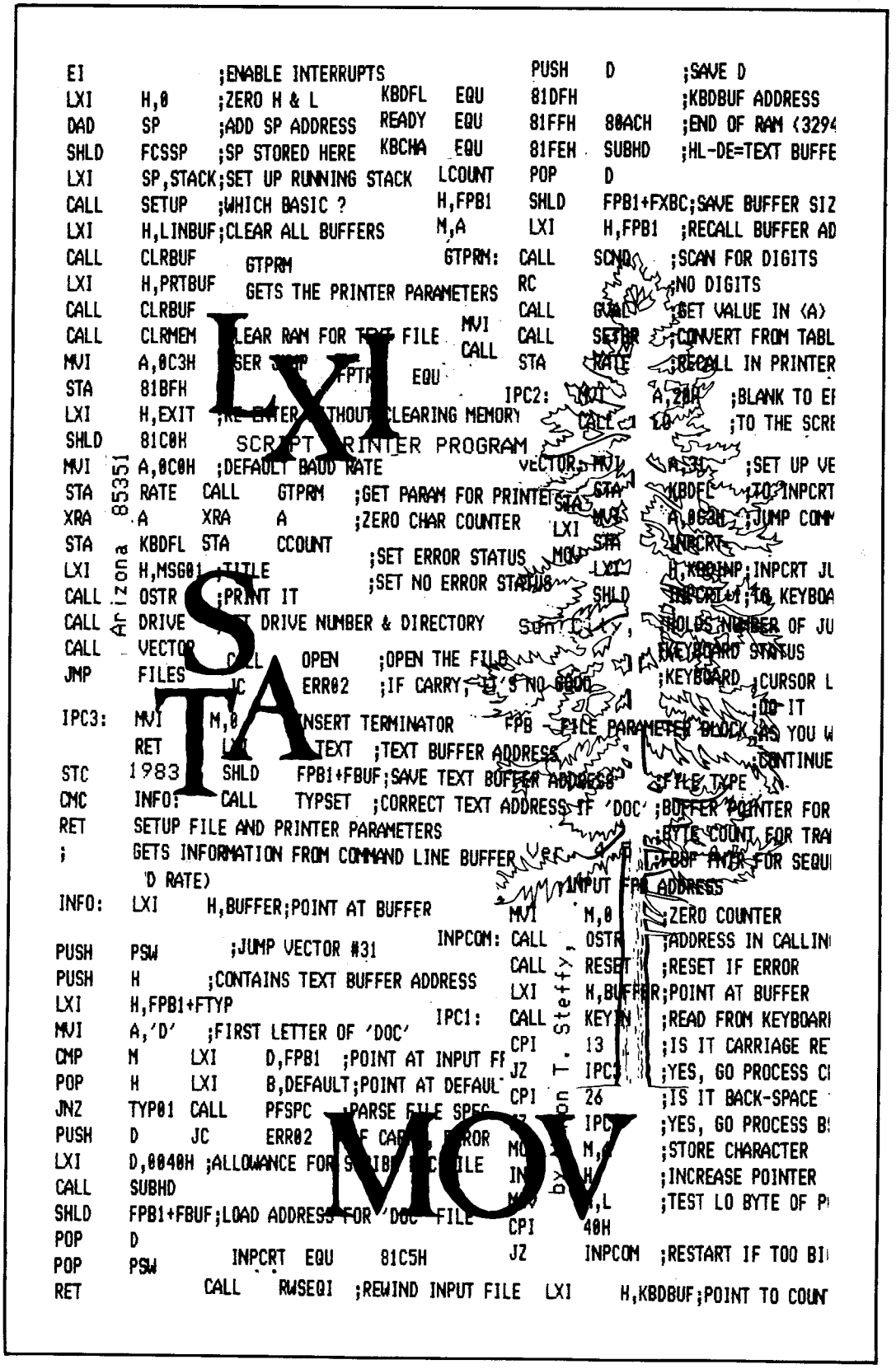
COLORCUE

VOLUME VI
NUMBER 1

A BI-MONTHLY PUBLICATION BY AND FOR INTECOLOR AND COMPUCOLOR USERS



FASBAS

BASIC VARIABLES

FORTH

BOOK REVIEWS

MODEM

Colorcue

VOLUME VI, NUMBER 1 JANUARY/FEBRUARY 1984

CONTENTS

COVER: A montage of SCRIPT by Myron Steffy.

EDITOR: JOSEPH NORRIS COMPUSERVE: 71106, 1302

COLORCUE is published bi-monthly. Subscription rates are US$18/year in the U.S., Canada, and Mexico (via First Class mail), and US$30 elsewhere (via Air Mail). All editorial and subscription correspondence should be addressed to COLORCUE, 19 West Second Street, Moorestown, NJ 08057, USA. (609-234-8117) Every article in COLORCUE is checked for accuracy to the best of our ability but is not guaranteed to be error free.

A CALENDAR PROGRAM is available from Christopher Zerr at 14741 NE 31st Unit, Bellevue, WA 98007. Written in Basic, it will print a calendar for any year neatly on an 8½"x 11" paper through your printer, or on the CRT. Mr. Zerr will make a copy on your submitted, formatted CD disk, or send you a printout of the program. Please include $2.00 either way for packaging and mailing.

ROBOT: "Scorpion" is a 9" x 12" mini-wonder that resembles a NASA Lunar Landing Module built from an advanced Erector Set. It is powered by a 6502 CPU, 8K-bytes of EPROM and 2K-bytes of RAM. Two 6522 chips provide 32 I/O lines and four programmable timers. It has sensors to detect obstacles in its path, a two-axis optical scanner, with 1.5 degrees of scan per step, which will move over a 300 degree plane both vertically and horizontally.

Visual patterns can be displayed on a monitor. Additional hardware includes sensing bumpers, loudspeaker, two ground tracks, two "eyes" and four motors - two of which are drive wheels. Operating from a 12 volt DC power supply, "Scorpion" may be programmed through any RS232 bus. As a kit, with complete assembly and programming instructions, it is priced at $660.

Rhino Robots, Inc. PO BOX 4010, Champaign, IL 61820. 217-352-8485.

"...we do it all for YOU!"

Welcome to another volume of Colorcue. The magazine has been brought to maturity by the dedication of Ben Barlow and David Suits, our previous editors. The new staff can only hope to equal their success, and with a little help from our friends - you the subscribers - we have a good chance

We have some ideas for the coming year but it is only your ideas that can give Colorcue a meaningful vitality. While the number of subscribers experiences a gradual decline, the decrease in membership provides an opportunity for greater responsiveness to individual needs, and we hope you will assist in bringing this about by increasing your communication with us.

Many of you are in possession of splendid material for an article which, for one reason or another, never saw the light of day. The editorial staff can help! We will take your rough drafts and expand them, collaborating with you on the final copy. If that seems too forbidding then throw caution to the wind and ask a few questions. We can probably put you in contact with someone who has an answer. This simple exchange can lead to good articles; you may be sure someone else has wanted the answer too. It may help you to know that our readership contains those with all levels of proficiency, from beginners in Basic to accomplished assembly language hackers. Colorcue serves us all.

There are users involved with other computers, or thinking about "moving on." It would be valuable to hear how you are choosing your next machine. What will you be looking for, and how will your experiences with CCII affect your choice?

Some of us are privileged to be near active user groups and know and share with others on a regular basis. Many, many more are isolated and working on their own. Let this be the year for knowing one another better. The comment I have heard most often, since I became editor, has been an expression of desire for more interaction with other CCII users. Several authors have told me they wish they could hear more from readers - some response to an article, or just a note to share ideas or ask questions. The hobby belongs to us all, and you'll find most authors pleased at your responsiveness.

We continue to be impressed with the savings offered by THE COMPUTER SHOPPER. If you spend more than $50 each year on some kind of computer supplies you will spend nothing for this magazine. There are interesting things popping up in its pages from time to time that you won't read about anywhere else.

Several requests have been made in the past for your network mailing address. A list of subscriber network numbers would be a valuable asset for isolated CCII users. We are repeating the request, and for those of you who are curious but don't know how to begin we have included an introduction to modem communications.

Colorcue is offering some new services to readers which are described in this issue. They are there for you to use. We are prompt with our correspondence and we can guarantee every reasonable effort will be made to provide a useful response to your questions and comments. Take note of the SOURCEBOOK, to be printed as VOL VI, No. 3. and the pamphlets we have been preparing for specific applications, such as using the modem and FORTH.

We will try, too, to keep you informed about user group activities. You can become part of a user group by joining yourself, even if you cannot attend meetings. A $10 membership fee provides access to interesting newsletters and often a valuable free disk library. So make a few waves and see what a difference a little risking can make. It's going to be a good year!

Joseph Norris

LINKUP is a new magazine dedicated to network users and small computer communications in general. A monthly, printed in Minnesota, LINKUP will contain the latest information on data bases —new and old, reviews of pertinent hardware and software, protocol descriptions (!), book reviews, news of upcoming shows, user group notes, and tips on using telephone lines at minimum cost. Charter subscription is $19.83 for 12 issues. Newstand price is $2.95/copy. Credit card orders may be phoned to 1-800-543-1300, or write to LINKUP, PO Box 26345, 3938 Meadowbrook Road, Minneapolis, MN 55426.

FORTH COMPUTER: The Juniper Ace 4000 is a dedicated FORTH computer with keyboard (no monitor or media storage) for $175. It may be used as an intelligent programmable control for AC and DC devices with a suitable interface. Using the Z80 microprocessor at 3.25 MHz , the Juniper uses standard FORTH-79 in ROM, with 51K-bytes maximum RAM capacity. The computer may be connected to monitors and displays 32 columns by 24 lines. Low resolution graphics are available. Storage is by means of a cassette recorder. A FORTH programming manual is also available at $14.95. Computer Distribution Associates. 17 South Main Street, Pittsford, NY 14534.

COMPILING BASIC

—Part Two

Peter Hiner
11, Penny Croft
Harpenden
Herts, AL5 2PD
ENGLAND

In Part One we looked at how some of the simplest BASIC statements are handled by the Basic Compiler (FASBAS) and at the same time noted how these statements are handled by the Basic interpreter resident in ROM. In this part we continue to compare the Basic compiler and interpreter but consider more complicated functions.

We have looked at evaluating mathematical functions like A+B-C from left to right, but in contrast the function A+C*C requires that B should be multiplied by C before A is added. This is achieved by assigning priorities to the arithmetic operators, as described in the CCII Basic Manual. The order of priority is such that the contents of the brackets are evaluated first, followed by the power function, followed by multiply and divide, etc. Within a pair of brackets the same priority sequence is applied, and if two operators have the same priority, then evaluation is from left to right. Both the interpreter and the compiler use the same fundamental technique for handling evaluation, which is to compare the priority of the next operator with that of the one currently being handled, and if the next has higher priority, then all current information is pushed onto the stack to be retrieved after the higher priority function has been dealt with. So in our example of A+B*C, the procedure would be as follows. Move the contents of variable A to the Basic Accumulator, and when the + sign is encountered, move the contents of the Basic Accumulator to stack. Move the contents of variable B to the Basic Accumulator and then look at the next operator. So far the operation is the same as for the simple A+B evaluation.

If the next operator were an end of the line marker then everything would be set up ready to pop variable A from stack to registers BC and DE, and to proceed with addition (as described in Part One.) But in this case the next operator has a higher priority, so we must push the + instruction and the contents of the Basic Accumulator (i.e. variable B) to stack.

Now we move the contents of variable C to the Basic Accumulator and check the next instruction, which is the end of the line. So we can go ahead with multiplication, popping variable B from stack to registers BC and DE and calling the multiplication subroutine. This leaves the result B*C in the Basic Accumulator. Now we pop the previous instruction (+) from stack and compare its priority with that of the next instruction (end of line.) The comparison tells us to perform the addition next, so we pop variable A from stack to registers BC and DE, and call the addition subroutine, giving the result A+B*C in the Basic Accumulator. If the Basic statement had been longer, with another instruction (such as *D) having higher priority than addition, then we would have pushed the + instruction back onto the stack, pushed the contents of the Basic Accumulator (B*C) to stack and carried out *D next.

So how do we know when we have finished? The answer is that before we start an evaluation we push a zero value to stack, and we treat this just like the other operators (+, *, etc.). The zero value has lowest priority, equal to that of an end of statement or end of line marker, so it will not be dealt with until we have completely evaluated all the other mathematical functions. Then it will cause us to return from the evaluation subroutine to the routine we were in previously (e.g. return to a PRINT routine with the result of evaluation in the Basic Accumulator).

This is a convenient point to pause and consider the overall structure of Basic, which influences the design of both an interpreter and a compiler. We can identify three classes of Basic instructions, which I will call command, sub-command and operator.

Commands can be simply defined as those instructions which can appear as the first instruction in a line, and which include GOTO, GOSUB, PLOT, PRINT, IF, INPUT, etc. This class also includes LET, which never appears explicitly in our version of Basic, but is implied in statements such as A=B. These commands immediately put limits on the structure of the rest of the Basic statement. For example GOTO or GOSUB must be followed by a line number and then an end of statement or end of line marker. PLOT must be followed by a mathematical expression (variables, constants, etc.), and after that either an end of statement/line marker or else a comma to indicate that more PLOTS are required.

A command can therefore be used to direct the interpreter or compiler to a routine specifically designed to handle all the types of structure which are allowed in the rest of the Basic statement. The GOTO routine wil automatically treat the next character(s) as a line number. The PLOT routine will automatically treat the next character(s) as a mathematical expression to be evaluated and plotted, and will look for a comma to cause it to loop back and perform the same task again.

"We can identify three classes of Basic instructions...."

Sub-commands are instructions like THEN, TO, STEP, etc., which can only appear in statements following specific commands. Therefore they will be handled within the routines designed to handle the commands that must precede them. For example the IF routine includes a routine to handle THEN.

Operators are all those instructions which can appear within an expression that requires evaluation. This class includes the mathematical and logical operators like +, -, AND, OR, =, SIN, COS, etc., and also some less obvious operators such as PEEK and CALL.

The importance of this distinction between commands and operators is that this concept allows for a very simple structure within an interpreter or compiler. The initial command is used as a key to a table of addresses for the routines which handle commands. When the appropriate command routine has been entered, it will be used as the main program for interpreting the rest of the Basic statement. The command routine will call other subroutines as required, and in particular will call an evaluation subroutine (which I will name EVAL) to evaluate any combination of variables, constants, or operators that may be present between the command and the end of the statement (or other termination point such as a comma or a sub-command). The EVAL subroutine must handle this evaluation completely and leave the results set up in a suitable form for immediate use by the command routine. Evaluation includes not only mathematical functions, but also all forms of string and string functions, and all forms of comparison (for example in the statement IF A = 2 THEN..., the portion A = 2 is evaluated as a comparison and results in a value of -1 in the Basic Accumulator if true, or 0 if false).

The EVAL subroutine is extremely powerful and is central to both interpreter and compiler. There are major differences in implementation of the EVAL subroutine in the interpreter and compiler, but both apply the same fundamental principles. We have already looked at the principles of evaluation from left to right and in priority order. Now I will describe evaluation of brackets, and in particular nested brackets.

When a bracket is opened, evaluation of other parts of an expression must be suspended (everything pushed onto stack) until the contents of the bracket have been evaluated as this has the highest priority. To avoid any confusion in priority order between items inside and outside the bracket, the EVAL subroutine treats the contents of a bracket as a completely new evaluation, and to do this the subroutine calls itself. Subroutines using this technique are described as being recursive, and great care must be taken to prevent their vanishing inside themselves like the fabled Oozlum bird. Each time a bracket is opened, EVAL will call itself, and it will then unravel itself as the brackets are closed again. Although this technique makes it difficult to follow through the flow of a program, it does have the virtue of making the program very compact (for example the core of the EVAL subroutine in the Basic interpreter is less than 500 bytes long). The EVAL subroutine in the compiler sorts out nested brackets and presents them in the required order for computation in the final compiled program, so that, for example, ABS(INT(COS(A))) is presented for evaluation in the order A, COS, INT, ABS.

"...of practical value in speeding up programs."

I mentioned above that EVAL must also handle strings and string functions. For a proper understanding of this subject we should start with some fundamental properties of strings. The interpreter stores the references to strings in the same area as it stores variables, and distinguishes them by adding 80H to the second character of the name (e.g. variable A is stored as 4100H and string A$ is stored as 4180H). The other 4 bytes of a numeric variable location contain its floating point value, but for a string these bytes contain references to the address at which the string itself is stored and its length.

If, for example, line 100 contains the statement A$ = 'OK', then the reference address will point to the location within line 100 at which the first letter of the string is stored, and the length will be 2 characters. If line 110 contains B$ = A$, the EVAL subroutine will be used to evaluate A$, and will return with an address in the first two bytes of the Basic Accumulator (80DE-F). This address will point to the four bytes of the A$ storage location containing the references to the address of the string itself (i.e. OK) and its length. Now B$ can be made equal to A$ by simply moving these four bytes to the storage location for B$, so that both A$ and B$ point to the same string.

If we then have a statement PRINT B$, the EVAL subroutine will return with the address of the four byte storage location for B$, and this time the information stored there will be used to find the string (OK) and to PRINT two characters.

The compiler makes use of the same string subroutines as the interpreter in the final compiled version of a Basic program. It saves some time by loading the string name (e.g. 4180H for A$) directly to the registers and then it uses the ROM subroutines to search for the six byte string location, which is stored dynamically (i.e. at run time) in exactly the same manner and the same memory area as in a normal Basic program. This is still faster than in Basic, because the compiler allocates a separate storage area for variables at compilation time and therefore the search for a string does not get slowed down by having to sort through a large number of variables.

Generally the most that I can hope is that this series of articles will satisfy your intellectual curiosity, but the next section may also be of practical value in speeding up programs. This description of string contatenation applies equally to the Basic interpreter and to compiled programs.

When strings are contatenated (as in A$ = B$ + C$), the resultant string is stored in the string manipulation area at the top of memory. If B$ equals 'GOOD' and C$ = ' MORNING', a string 'GOOD MORNING' will be created and the A$ storage location will contain a reference to the storage address and length of this new string. If we now have A$ = A$ + ', SIR', another new string will be created in the string manipulation space and A$ will point to this string instead. But the string 'GOOD MORNING' will be left as a piece of garbage until the string manipulation space has been filled. Then the interpreter will sort through the strings throwing out garbage and making available as much free string space as it can. This can happen at unex-

pected times during a program run and cause a complete suspension of other activities for up to several seconds (particularly if you have increased the amount of string manipulation space from the minimum allocation of 50 bytes, by using a statement such as CLEAR 500). If you set the string manipulation space at the smallest size that will allow the program to run without OS ERROR messages, then you will spread the effect, causing a lot of short interruptions, but the overall result will be a longer run time. For example you might cause 100 delays of 0.1 seconds (total 10 seconds) instead of 1 delay of 5 seconds.

You can force garbage collection at any time you want by putting in a statement like W = FRE(X$). A suitable time would be when you have a message on screen which will require some time for the user to read. Beware of collecting garbage when you want a keyboard response from the user, as the interpreter will not be able to accept a key entry via a normal INPUT routine, and the user may respond before the interpreter is ready.

The best solution is to minimize the number of string concatenation operations, which are slow operations in themselves, apart from the delays caused by garbage collection. It is faster to PRINT A$;B$ than to PRINT A$ + B$. When you really need to concatenate strings, do it all at once rather than as a series of statements repeatedly tacking bits on the end of a string.

I once wrote a program which included a subroutine for converting numbers from decimal to hexadecimal, dividing the decimal number by 4096 to generate the first character of a string, then dividing the remainder by 256 to get the second character and concatenating them to make a two character string, then dividing by 16 and so on until I had generated a 4 character string. Now I know why it was so slow!

In Part Three of this series of articles I will describe how the compiler deals with arrays and the Basic commands. □

[FASBAS is available from the author for $25US. Ed]

FOREIGN LANGUAGE DEPARTMENT:

The indescribable language, FORTH, has been issued in three different versions for Compucolor computers. One issue is available for the 3651. It is probably the most-avoided language invented, yet the experience of FORTH has much to add to one's programming education.

FORTH is a language that combines high-level and low-level language functions. It is fast, self-'compiling', may be used in 'immediate mode' during the learning process—just like Basic—and provides a programming grammer of unusual proportions. FORTH has a vocabulary of 'key' words—just as Basic does—that may be invoked to perform specific predetermined functions. These words are arranged in sequence, by the programmer, to provide an ordered set of instructions.

FORTH uses a 'stack' just as assembly programming, but in a more immediate way. It also uses 'reverse Polish' notation, much as the Hewlett-Packard calculators. As an example, here is a FORTH instruction line to add the numbers '3' and '5' and print the sum to the screen:

```
3 5 + .        ( Type (3) (space) (5) (space)
               (+) (space) (.) )
```

Here is what happens as FORTH reads the above line:

```
3              ( '3' placed on top of stack )

5              ( '3' moved down the stack and 5
               placed on top )

+              ( The symbol '+', which means
               "add the top two numbers on the
               stack". These two numbers are
               popped off the stack as they are
               read, and their sum, '8' is placed
               on the top of the stack. '+' is a
               FORTH command word. )

.              ( The symbol '.' means "print the
               top of the stack to the CRT. After
               this operation the stack is empty. )
```

FORTH gives you the power to make your own commands. If you feel more comfortable using the word 'PRINT' to print, then an instruction early in your program can make this possible. It is very easy to define a new word in FORTH. On a separate line, one types a ':' (colon) to indicate a new word definition. Next the name of the new word to be defined is entered, followed by the list of instructions the new word is to signify. Finally, a semicolon is entered to terminate the definition. Here is an example (note each element of the definition is separated by spaces):

```
: PRINT . ;   ( This means "hereafter when I type
                PRINT, perform the '.' function." )
```

Or I might want to make a word that adds and prints both:

```
: PRINTSUM + . ;   ( This means "hereafter when I
                     type PRINTSUM, add the top two
                     numbers of the stack and print
                     their sum on the screen." )
```

This introduction of 'new words' to FORTH provides the programmer with a set of instructions (subroutines of a sort) tailored specifically to his needs. You can actually design your own programming language and never use a single word of 'real' FORTH, once your own words have been defined. For this experience alone, FORTH is worth some investigation.

You may combine your own new words to make still another valid command. Here is a word defined to perform a carriage return and a linefeed:

```
: CRLF 10 13 EMIT EMIT ;   ( My command word is CLRF. )
```

The command word 'EMIT' is like the Basic 'PRINT CHR$(x)' word. Numbers on the stack will be printed in their ASCII character equivalent rather than as numbers. Now I may define a FORTH word which I will call 'GO', which will simulate line 20 in the following Basic program:

```
10 A=1 : B=6
20 C=A+B : PRINT : PRINT C : PRINT
```

First I define my special FORTH word, equivalent to line 20 -

```
: GO CRLF + . CRLF ;
```

And this is how I would simulate the Basic program above, in FORTH:

```
1 6 GO    ( Put '1' and '6' on the stack,
            then do "GO." )
```

Another consequence of the FORTH experience is that 8080 assembly programming takes on a fresh and creative aspect. Many stack manipulations of FORTH are transferrable to assembly routines. Since most of us don't use the 8080 stack fully, FORTH provides us with valuable insights for increasing the vitality of stack-related instructions.

I was captivated by the method FORTH uses to store disk files. There need be no disk directory since FORTH uses the REAd and WRIte disk routines, storing files in 1024 byte blocks directly to the uninitialized disk. When you want to call a program to the screen you do so by specifying what start block it may be found on. If you keep no records on paper about the storage location of programs, it can be interesting trying to find them again. Nevertheless, there is a quality of magic about the process.

We are probably inclined to stay away from things that are 'good' for us but, for me, the primary value of FORTH has come from the discipline of total submission to a new, strange and demanding programming pattern. It is frustrating at first to lose the facility we have in other languages as we plod through the elementary learning procedures all over again. But second and third languages build on our previous experiences and facility does seem to come more rapidly with each new language.

FORTH may not become your favorite form of relaxation, but it is just plain fun to play with - and it's a good way to rekindle your excitement at the keyboard. Furthermore it is inexpensive. The Rochester Users Group has a version, free for the price of a $10 annual membership (Gene Bailey, 28 Dogwood Lane, Rochester, NY 14625). Bill Greene has just released a version for CCII and 3651 at no charge (but PLEASE send him a formatted disk and about $5 to cover costs: 3601 Noble Creek Drive, Atlanta, GA 30327). There is little documentation from either source but you will not have too much trouble until it's time for more advanced procedures.

There are three tutorial books that can be recommended. (I needed at least two!) The most accessible is by Thom Hogan: Discover FORTH. Osborne/McGraw Hill, 1982. The second is a little more formal—by Paul Chirlian: Beginning FORTH. Matrix PUblishers, 1983. Both of these volumes are available at leading bookstores or computer stores. The third volume is very recent: The Complete Forth by Alan Winfield, published by Sigma/Wiley, NY, 1983. For me, this third volume was the most appealing since it spoke to my level of programming savvy—not too little, not too much, but just right. I was disappointed that none of them discussed assembly or debugging, but I'm not that far yet! While none of these volumes is specific to the CCII, you will find that the FORTH basic vocabulary is fairly constant among different versions and that few special commands are required. All you need to get started is one version of FORTH on disk and one of the above tutorials.

[COLORCUE is preparing an 'interface' pamphlet to help you get started with Bill Greene's FORTH. You may write for it; the price is $2.00. If you are interested in reading more about FORTH in COLORCUE, drop us a line.]

A SERIAL TO PARALLEL INTERFACE is available for $90.00 which connects to the RS232 output of the CCII and produces a parallel "Centronics" output for a peripher device. It works well with pen plotters, printers, paper punches, and robots. Handshaking is provided as well as 8 Baud rates. It may be ordered from Engineering Specialties. Phone 805-487-1665 for information (CA). [Also see Ben Barlow's do-it-yourself article in Vol IV No. 3, DEC/JAN 1982.]

A PROGRAM TO LOAD SRC FILES INTO COMP-U-WRITER

Myron T. Steffy

The COMP-U-WRITER series of word-processors provide three modules for converting their own brand of text files into a type labeled 'DOC' that may be stored with the usual FCS disc SRC files generated by the COMPUCOLOR DOS. The DOC file retains the first 40H bytes of the file as necessary instructions to COMPUWRITER for reloading. In this form, the file could not be read by the Screen Editor but could be printed by a program named 'SCRIPT'. The first 40H bytes were simply discarded and the remaining text loaded and printed. There has been no easy way of performing the converse operation, that is, loading an original SRC type file into the COMP-U-WRITER framework.

The routine labeled 'LDFCS' is a machine language method of supplying the missing information and then loading the SRC file into the COMP-U-WRITER system. It will operate with versions 3.4, 3.5 and the current 3.6 generally known as the 'EXECUTIVE'.

The SRC files generated with the Screen Editor will have their lines terminated with a carriage return and a line feed. If the file has had its lines justified by a program similar to 'SCRIPT', there will be extra spaces inserted between words to stretch out the lines to a uniform right margin. When 'LDFCS' is used to load the file into COMP-U-WRITER, the line feeds are automatically removed as they are not required. There is an option in LDFCS that will allow you to alter the file so that the COMPU-WRITER's justification mechanism can operate.

If this option is selected, the carriage returns are replaced by 0A0H when they occur singly. This code is a terminator used by COMP-U-WRITER at the end of a line prior to justification. When double C/R's are encountered, they are retained to provide paragraph spacing as originally intended. All of these operations are more or less predicated on a 60 character line which is all that will fit on the CCII screen. If the file has longer lines, LDFCS will still work but the lines may double back upon themselves on the screen.

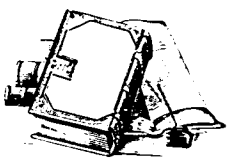
Since 'LDFCS' uses some functions within COMP-U-WRITER itself, the latter must be in place prior to running 'LDFCS'. This is also true of the other three modules. The method of operation is very much like the other modules. Be sure to do a [COMMAND/RESET] before loading the COMP-U-WRITER. After entering the usual preliminaries, the date and the drive number, exit the program with [CPU/RESET]. Then do an [ESC D] and 'Run' LDFCS in the default drive, not the drive number selected for COMP-U-WRITER text files. It will come up with a heading 'COMP-U-WRITER FILE LOADER' and the option question whether you want the C/R's and extra spaces removed. This defaults to 'Yes' as will usually be the case to allow COMPU-COLOR's internal justification mechanism to operate. If the file is an assembly language source file or other material where ordered columns are desired, answer 'No'.

The program will then prompt you for the file name and type. If it is an SRC file, you must so state as the default is 'DOC'. If it should be a 'DOC' type, the file will be loaded intact and the option rendered inactive. After typing in the file title, press [RETURN] and in a second or so, COMP-U-WRITER's usual heading will appear with the file you have selected. All of the usual functions are available and the text may be treated as any other file. Files of either type may be concatenated and the loading will take place at the point where the cursor is located when you exit COMP-U-WRITER. However, make it a rule to place the cursor at the end of the residual text and move the appended file with the [MOVE/BLOCK] function of COMPUWRITER after loading. When L/F's and other unwanted characters are removed from the file, the text will be shortened somewhat. Under certain conditions, you may find vestiges of the original text at the end of the file just loaded. Usually exiting COMP-U-WRITER with [CPU/RESET] and re-entering with [ESC] [USER] will straighten things out. Extraneous garbage may be readily removed with the [DELETE/BLOCK] facility if necessary.

With a 32K system there will be something over 17,000 (decimal) bytes available for text. If the file you are loading exceeds your RAM capacity, a message will appear on the screen to that effect. Pressing [RETURN] will take you back to COMP-U-WRITER and you will find a partial loading of the text file. A file too large for the system will have to be edited in sections. Again, if the tail end of the text is garbled, exit with [CPU/RESET] and re-enter with [ESC] [USER].

.The source code for LDFCS that follows is set up for assembly at 4000H where it may be retained and re-used for multiple file conversions. If you do not have a Devlin RAM board at that address, you may move it into the normal RAM area by changing the ORG to 8300H. Then at the very end, change the line 'DBSIZ EQU 5F00H-DBUF' to 'DBSIZ EQU 8E00H-DBUF'. LDFCS will have to be reloaded each time it is used since COMP-U-WRITER uses that area for I/O buffers and will overwrite it. □

```
;    LOAD COMP-U-WRITER TEXT FROM FCS SRC and DOC FILES
;=======================================================

;Removes L/F's and C/R's and excess spaces from SRC files.
;       by Myron T. Steffy, Sun City, Arizona  3/15/83

        ORG     8300H

START:  CALL    SETUP   ;WHICH BASIC ?
        LHLD    8F60H   ;HAS TEXT START ADDRESS
        SHLD    RAMST
        MVI     A,1     ;SET FLAG FOR SPACE REMOVAL
        STA     LFLG
        LXI     H,TITLE
        CALL    OSTR
        CALL    GETANS  ;WANT EXTRA SPACES REMOVED ?
        CPI     'N'     ;SAY NO FOR ASSEMBLER SRC FILES
        JNZ     QUERY
        XRA     A
        STA     LFLG

QUERY:  LXI     H,8EFFH
        SPHL            ;RESET STACK PTR
        CALL    912AH   ;INIT FOR RE-ENTRY
        CALL    RESET   ;RESET DISK IF ERROR
        LXI     H,MSG01 ;FILE SPECS
        CALL    OSTR
        LHLD    VHLAD
        XCHG
        LXI     H,NOCUR ;MOVE CURSOR OFF SCREEN
        CALL    OSTR
        LXI     H,BUFFER;POINT AT BUFFER
        MVI     B,20
        CALL    9F47H
        MVI     M,0     ;REQ BY OPEN
        LDA     91C6H   ;LOAD ENTRY
        CPI     1EH     ;UNMODIFIED  SCRIBE ?
        CNZ     9E51H   ;NO. CALL SAVE CURSOR
        LXI     H,FPB
        MVI     B,38
        CALL    CLEAR   ;CLEAR FPB
        LXI     H,BUFFER
        LXI     D,FPB   ;INPUT FPB
        LXI     B,DFDOC ;DEFAULT TYPE(DOC)
        CALL    PFSPC   ;PARSE FILE SPEC
        JC      E02     ;IF CARRY THEN ERROR

        LXI     H,FPB   ;INPUT FPB
        MVI     A,0     ;OLD FILE
        MOV     M,A
        CALL    OPEN    ;OPEN FILE
        JC      E02

        LXI     H,DBSIZ ;SEQ DISK BUFFER SIZE
        SHLD    FPB+FXBC
        LXI     H,DBUF  ;ADDRESS
        SHLD    FPB+FBUF
        LDA     FPB+FTYP;FILE TYPE ?
        CPI     'S'     ;SRC FILE ?
        JZ      SRCFIL  ;LOAD FILE ATTRIBUTES
        LXI     H,FPB   ;DOC FILE SPECS
        CALL    RWSEQI  ;REWIND FILE
        LXI     H,8F00H ;PARAMETER AREA
        MVI     B,40H
DSPEC:  PUSH    H
        PUSH    B
        LXI     H,FPB
        CALL    GTBYT
        JC      ERROR
        POP     B
        POP     H
        MOV     M,A
        INX     H
        DCR     B
        JNZ     DSPEC

TXT10:  LXI     H,FPB
        CALL    GTBYT
        JNC     TXT15
        JNZ     ERROR
        JMP     TXT20

TXT15:  LHLD    8F60H   ;CURRENT LOAD LOCATION
        MOV     B,A
        XCHG
        LHLD    8F62H   ;HIGH RAM LIMIT
        CALL    CMPDH
        XCHG
        JC      TXT17   ;NOT FULL YET
        CALL    90D2H   ;CLEAR SCREEN
        CALL    9DF3H   ;ERROR MSG - RAM FULL
        CALL    RESET
        JMP     TXT20   ;RETURN TO SCRIBE

TXT17:  MOV     M,B     ;STORE INPUT BYTE
        INX     H
        SHLD    8F60H   ;CORRECT COUNT
        JMP     TXT10

TXT20:  LDA     FPB+FTYP
        CPI     'S'     ;SRC FILE ?
        JZ      ST01

TXT30:  CALL    90D2H   ;CLEAR SCREEN
        CALL    0A650H  ;RE-JUSTIFY TEXT
        JMP     9270H   ;RE-ENTER

SRCFIL: LXI     H,PARA  ;PARAMETER LIST
        LXI     D,8F00H ;PARAMETER AREA
        MVI     B,40H
```



```
        CALL    MOVDH   ;PUT IT IN PLACE
        LXI     H,FPB   ;NOW GET TEXT
        CALL    RWSEQI
        JMP     TXT10

EXIT:   MVI     M,0A0H
        LHLD    SAVADR  ;END OF CORRECTED TEXT
        DCX     H
        SHLD    8F60H
        JMP     TXT30

E02:    PUSH    B
        CALL    90D2H   ;CLEAR
        POP     B
        CALL    EMESS   ;EMIT ERROR MESSAGE
        CALL    RESET
        CALL    WAIT
        JMP     TXT20

ERROR:  CALL    90D2H   ;CLEAR
        LXI     H,MSG02 ;PRINT ERROR MESSAGE
        CALL    0A1EEH
        CALL    WAIT
        JMP     TXT20

;       REMOVE C/R AND EXTRA SPACES FROM TEXT

ST01:   LHLD    RAMST   ;START OF TEXT RAM
        PUSH    H       ;SAVE IT
        XCHG
        POP     H       ;GET IT BACK
ST02:   PUSH    D
        XCHG
        SHLD    SAVADR  ;END OF CORRECTED TEXT
        XCHG
        POP     D
        MOV     A,M
        ORA     A
        JZ      EXIT
        CPI     0AH     ;L/Fs NOT WANTED
        JZ      SKIP
        ANI     7FH     ;CONVERT TO ASCII
        STAX    D       ;PUT CHARACTER AT DE ADDRESS
        MOV     B,A
        LDA     LFLG
        ORA     A
        JZ      NEXT    ;IF ZERO, SKIP THE REST
        MOV     A,B
        CPI     20H
        JZ      SPACE
        CPI     0DH
        JNZ     NEXT
        MVI     A,0A0H
        STAX    D
        INX     D
        INX     H
        MOV     A,M
        CPI     0AH
        CZ      NOPAR
        MOV     A,M
        CPI     0DH     ;SECOND C/R ?
        JNZ     ST02
        DCX     D
        STAX    D       ;TWO C/Rs FOR A PARAGRAPH
        INX     D
        STAX    D       ;SECOND C/R
        INX     D
        INX     H
        JMP     ST02

SKIP:   INX     H
        JMP     ST02

NEXT:   INX     H
        INX     D
        JMP     ST02

NOPAR:  INX     H
        RET

SPACE:  CPI     20H     ;SPACE ?
        JNZ     NEXT    ;NO
        INX     H
        MOV     A,M
        DCX     H
        CPI     20H     ;TAKE OUT ALL BUT SINGLE SPACES
        JNZ     NEXT
        INX     H
        MOV     A,M
        JMP     SPACE   ;DO IT AGAIN

WAIT:   LXI     H,9E28H ;'HIT RETURN'
        CALL    0A1EEH
        CALL    09E48H  ;WAIT FOR CR
        RET

CLEAR:  XRA     A
        MOV     M,A
        INX     H
        DCR     B
        JNZ     CLEAR+1
        RET

GETANS: MVI     A,50H
        STA     READY
GETCHA: CALL    0024H
        JNZ     GETCHA
        RET
```

```
SETUP:  LDA     0001H   ;VERSION 8/79?
        CPI     0BAH
        RZ              ;NO,  6.78
        LXI     H,NEWTAB
        LXI     D,OLDTAB
        LXI     B,LENTAB
MOVE:   LDAX    D
        MOV     M,A
        INX     H
        INX     D
        DCX     B
        MOV     A,B
        ORA     C
        JNZ     MOVE
        RET

;       SYSTEM ADDRESSES(6.78)

OLDTAB: JMP     262DH   ;EMESS
        JMP     3077H   ;PFSPC
        JMP     26A5H   ;RESET
        JMP     2DABH   ;OPEN
        JMP     30C6H   ;RWSEQI
        JMP     322CH   ;GTBYT
        JMP     33F4H   ;OSTR
        JMP     3453H   ;CMPDH
        JMP     343BH   ;MOVDH

        LENTAB  EQU     $-OLDTAB

;       SYSTEM ADDRESSES(8.79)

NEWTAB:
        EMESS:  JMP     0AD6H
        PFSPC:  JMP     14ADH
        RESET:  JMP     0B48H
        OPEN:   JMP     11E1H
        RWSEQI: JMP     14FCH
        GTBYT:  JMP     1662H
        OSTR:   JMP     182AH
        CMPDH:  JMP     1889H
        MOVDH:  JMP     1871H

NOCUR:  DB      3,64,0,239
DFDOC:  DB      'DOC'

TITLE:  DB      15,13,10,10,17,9,9,'COMP-U-WRITER FILE LOADER'
        DB      19,13,10,10,'REMOVE L/Fs  and EXTRA SPACES ? '
        DB      20,'(DEFAULTS TO YES) ',18,239

MSG01:  DB      13,10,10,19,'FILE NAME ',17,'(SRC or DOC) '
        DB      21,'DEFAULTS TO DOC ',18,239

MSG02:  DW      7290H
        DB      'INPUT ERROR ON DOCUMENT FILE',0

PARA:   DB      66,0,60,0,10,0,0,0,128,37,1,0,8,16,24
        DB      32,40,48,255,0,0,0,0,0,0,0,0,0,0
        DB      0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
        DB      0,0,0,0,0,255,0,0,0,0,0,0,80,0

FPB     EQU     80F7H

READY   EQU     81FFH
VHLAD   EQU     81D2H
LFLG:   DS      1
RAMST:  DW      1
SAVADR: DW      1

EMFN    EQU     15H     ;MISS FILE NAME ERROR CODE

;FPB OFFSETS

FTYP    EQU     08
FBUF    EQU     32
FXBC    EQU     34

BUFFER: DS      20
        ORG     ($+127)/128*128
DBUF:   DS      0
DBSIZ   EQU     8E00H-DBUF

        END     START
```

We note with sadness the passing of Myron Steffy who died shortly after Christmas. Many of you will remember his steady stream of stimulating articles in Colorcue and Forum, particularly the evolution of SCRIPT, his word processor. Many others of us have known him as a supportive, knowledgable friend and helper. Myron was the first to respond to a call of assistance, whether it be help with a personal programming problem or the need for an article to enrich a not-quite-complete journal edition. Although in his seventies, Myron was in the front rank of those working to expand the utility of the CCII. As a programmer he was thorough and articulate. We will yet see more from him as some later projects are brought to completion by his associates. It has been a fortunate thing to have him in the Colorcue community.

David Suits
49 Karenlee Drive
Rochester, NY 14618

BASIC FROM THE GROUND UP. David E. Simon.Hayden Book Co.; 219 pp.; pbk

The reader who wants to know a little bit about internal operations of the computer, as well as learn a high level programming language, would benefit from this book. It starts on an elementary level, assuming the reader knows little about computers or even algebra, and clearly and concisely develops a knowledge about BASIC to a great extent. The book covers many aspects of the language and gives good sample programs to follow along with. The examples and the text will help the reader develop good structuring techniques for his/her algorithms and programs. Exercises are included at the end of each chapter to review key concepts, and there are appendices providing ASCII codes, a glossary, and a brief summary of BASIC statements.

Although each concept does not have an individual program which illustrates it, there are several good programs to illustrate many points at once. Most of these are immediately adaptable for use on the Intecolor or Compucolor computer. The book does base its applications on a time-sharing system, and some programs would require a little modification (omitting PRINT USING, random file commands, etc.). The programs cover such subjects as sorts, list look-ups, function graphing, and other mathematical problems.

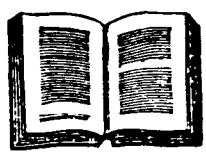
BASIC FROM THE GROUND UP is an excellent general work for someone who is seriously interested in learning BASIC for practical applications. The book concentrates on providing the text information on BASIC syntax and on descriptions of the BASIC vocabulary. It does not cover microcomputer-oriented BASIC as such, but it does provide enough knowledge of the language to enable the reader to program on microcomputers. This is not a book for someone who is simply interested in obtaining a list of programs to be copied and applied; instead, it gives the reader the tools necessary to be an effective BASIC programmer. It is an excellent reference work for BASIC.

DISCOVERING BASIC. Robert E. Smith,Hayden Book Co.203 pp,; pbk

This book begins as a brief introduction to BASIC, using many sample programs which illustrate points the author is currently covering. The work deals with the fundamental BASIC vocabulary without going into detail about the statements. It is intended to allow the reader to begin programming immediately, experimenting with each new concept as it is introduced. As the book progresses, the example algorithms become more complex but still presented at the neophyte level. Also, each chapter is summarized with a brief quiz which is graded by a corresponding BASIC program which the reader actually types in and runs.

Discovering BASIC contains over 50 programs, many of which could be entered directly into the Intecolor or Compucolor without modification. The last 70 pages are entirely devoted to algorithmic problems (solutions are given in the back of the book) that include computing pi, dice throwing, amortization scheduling and annuities. Although originally written with time-sharing in mind, the programs not directly applicable to Intecolor or Compucolor computers can be easily modified with a little scrutiny (ignoring MAT statements, entering BASIC from the time-sharing operating system, etc.).

In general, this book tends to stress algorithms as opposed to devoting much time to the finer points of BASIC. Within the text, the author has underlined key words to stress important points, increasing conceptual understanding. The immediate hands-on experience idea is also a very practical approach to the subject matter. However, the book is geared to time-sharing systems—not to microcomputers—and does not use many of the features of microcomputer-oriented BASIC (multiple statement lines, PEEK, POKE, etc.). Furthermore, the book does not go into computer fundamentals such as memory configuration, but instead takes a problem solving approach to learning. This would probably not be a good book for the person who wanted to learn BASIC in depth and to know about microcomputer BASIC in particular.



Introduction to 8080/8085Assembly Language Programming by J. N. Fernandez and R. Ashley. John Wiley and Sons, 1981. 303 pp. pbk

Here is a most excellent introductory book for those persons who wish to begin learning about 8080 assembly language programming. The book assumes only a minimum of experience with computers. (A little BASIC is fine.) The topics include binary, decimal and hexadecimal numbers, binary arithmetic, the ASCII code, the 8080 instruction set, pseudo operations, plus tutorials on writing 8080 programs for simple number crunching.

The book is a 'self-teaching guide'; it has numerous exercises which are to be done in the spaces provided. (Solutions are given, too) This is a good approach, since, after all, learning to write assembly language programs can be done only by actually writing them. And the large format of the book (6-3/4" x 10") provides plenty of space for working out the frequent exercises and samples.

The front cover photo is of an Intecolor 3600 computer. Alas, there is nothing in the book which addresses itself specifically to ISC machines. In fact, most of the exercises will have to be altered (in very simple ways) if they are to be actually entered and run on an ISC computer, since the RAM addresses used in the text are not RAM addresses on an Intecolor or Compucolor.

There is a lot the book does not cover, but an introductory text such as this is not meant to anyway. And all those features specific to ISC machines (such as ROM routines and color graphics) are, of course, not covered. The book is meant to introduce the reader to 8080 assembly language programming, independently of the particular machine one uses. And this it does very well. □

REFRESHER COURSE

You may create and read non-random files from BASIC. The only restriction is that the format of the file must be expressible in terms of the FILE 'N' and FILE 'R' statements. You may supply your own extension code.

```
Example: FILE "N", "NOTES.PDQ",2,128,1
         FILE "N", "QUACK.SRC",1,256,1
```

To recover data from a .SRC file, created anywhere, you might do this;

```
100 FILE "R",1,"TEXT.SRC",1;2,64,1
120 GET 1;A$[64],B$[64]
```

This feature is very handy for accessing a data base in .SRC form, updated by some other program, and passing parameters to BASIC. You cannot extract numerical data directly since BASIC uses a special form for numbers, but you can retrieve a number in string form and use A = VAL(A$) to convert.

A typical example of use might be a catalog of items with a description, weight and price. This data is updated on a screen editor in strict format, but called by a BASIC program which writes invoices. (Expanded from a note in CUWEST, via FORUM.) □

GETTING STARTED WITH THE MODEM

Joseph Norris

It's a whole new world with a modem connected to your Compucolor II. What a wonderful way to 'reach out and touch someone' or dispel the 'Basic blues.' While the initial cost is significant, maintaining a modem connection with others is relatively inexpensive and a continuing source of entertainment.

With a modem you may communicate with other computers (not just CCII) through the telephone lines, or you may subscribe to a 'network' and utilize their facilities for work and play. A popular beginning network is CompuServe, which offers moderate resources at a low maintenance fee. This summary will describe the initial steps to establishing a modem and network connection for yourself, and define the approximate costs for putting it in operation.

You will need a modem. This device accepts output from the 'printer port' of the CCII and sends it over an ordinary telephone line. It also accepts signals from the telephone line and puts them on the CCII input port. (Both these ports are on the same connector.) Some lower-priced modems transmit only at 300 Baud while others add a 1200 Baud capability at a significantly higher cost. We use the Hayes Smartmodem(T), a 300 Baud device, which may be purchased for about $240. (300 Baud is fine for most purposes.) The modem 'input' connects to the MODEM or RS232 bus connector on the CCII. Unlike many other peripherals for the CCII, your modem can be used with most any computer you may purchase in the future.

The 'output' of the modem connects to a standard telephone outlet, usually in 'parallel' with a telephone handset which is used to dial numbers in the usual way. You will need to use the modem near such an outlet, or run one to your modem location. We bought an inexpensive ($9.95) telephone to use with ours for verbal communications (the modem does the dialing!).

Modems are available with a large variety of features, too many to describe here in detail. Some modems will store telephone numbers for you, dial automatically, turn on your computer if a call is received and you're not home, etc. Our recommendation is that you 'keep things simple.' CCII users have had good experiences with the Hayes and the CAT Novation modems. You will need to do some homework if you propose to add a 'fancy' device to the CCII.

You will also need some kind of 'terminal' software to enable your CCII to act like a terminal only. Comtronics has a nice program for this (TERMII) which adds some capabilities such as saving incoming data to disk for later use, and transmitting disk contents over the modem. TERMII costs about $70, and is available from CCII dealers.

At this point, with the connections made, you may communicate with another modem operator directly. The cost will be the telephone charges for the connection, just as you would pay for an ordinary telephone call. Assuming you had made arrangements with

SUBMITTING ARTICLES

You may submit material to Colorcue in a number of ways. Although we use a 3651, we can usually back up CD disks written with v8.79. If you use COMPUWRITER make a DOC file from your text and send that. You may submit SRC files as well. Modem owners may reach us through COMPUSERVE or by direct connection if a prearrangement is made by mail or telephone. For 3651 users, we can read MD, FD and DF disks (but not DM). If none of these methods is suitable, just send your text on paper. If program listings are a part of your material, please use a fresh ribbon to print them (white typewriter paper preferred) so we won't have to copy them and face the possibility of errors.

another party to talk at a specific hour, one of you will dial the other's telephone. You may speak to one another on the phone and by computer. With your terminal program running, you speak to him/her by typing on the keyboard. What they reply is printed on your screen. Both your message and the return message can be displayed on the screen—in different colors— so the entire dialogue is before you. You may also transfer files of various types. This means much of your transmission may be prepared in advance, thus lowering the expense of the actual connect time.

NETWORK CONNECTION. Possessing the equipment discussed so far, and using CompuServe as an example, here are the steps necessary to begin network communications. If you write a request letter to CompuServe they will send you a 'beginners kit' at a small charge. This kit contains general information about the network, a CompuServe log-on number for you to use and a 'password' that is exclusively yours. One hour of connect time is included in the price. Another easy way to begin is to buy, from RADIO SHACK, a TRS-80 Videotex Universal Sign-up Kit (#26-2224, $19.95). This kit contains a CompuServe number, a password, explanatory material, and includes one free hour of CompuServe time. (This is often the most inexpensive way to get a first membership.) Some modems come from the factory with 'access' packages enclosed, so, if you have the hand set and line connections, you can begin right away. Detailed descriptions of procedures are included. From either source, instructions are provided for continuing your membership beyond the first hour, if you wish. At 'log-on' time your modem will indicate to the network which Baud rate you are using. The hourly charge for 1200 Baud is higher than for 300 Baud.

CompuServe provides a number of interesting services, including a network mailbox, 128K of memory storage, user group bulletins, programs and games, and news items. There are 'hundreds' of data bases in this service, some requiring additional fees. For example, you may expand your access to Dow Jones reports, national news wire services and other more specialized information, including news releases by Federal agencies (which can be fascinating). Accumulated charges will be billed to you monthly by the network. If they are billed to Master Card or Visa charge accounts there is no billing charge (*).

CompuServe has installed many 'local' telephone access numbers across the North American Continent so that 'long distance' calls are not usually required, or at least available at reduced rates. Our CompuServe connection is only three miles away. A list of access telephone numbers will come with your information kit. The charges include a yearly membership charge, a 'log-on' time charge of about $6.00 per hour (if you use the service between 6PM to 5AM), billed in one minute segments, and the use of the telephone line from your local CompuServe exchange to the CompuServe computer complex (about $5/hr.) Since actual connect times can be very small, these charges are not a great burden. For example, we access our mailbox three times each week for a total log-on time of about 2 minutes. This includes the time I use sending messages as well as receiving them. Since messages can be prepared in advance, saved on disk, and transmitted from disk at maximum speed, little actual connect time is required.

"...a monthly expense of $12 will provide a goodly amount of entertainment."

Terminal programs will allow you to save all incoming data for later examination at your leisure. If you are playing a CompuServe game 'on-line' in real time, the 300 Baud rate, billed at one-fourth the cost of 1200 Baud, is the better method. All that time you spend 'thinking' on an open line costs money. Since we do not think faster at 1200 Baud, the 300 Baud rate and costs are the way to go. The only time 1200 Baud is cost effective is when long data files of many pages are being sent. COLORCUE goes to the typesetter by modem and at 300 Baud it's a bit costly. But this only happens once every two months and represents about 1/50 of total modem use for COLORCUE.

Colorcue has prepared a small users manual for connecting and operating the Hayes SmartModem using TERMII by Com-tronics. The price is $2.00. This manual details step-by-step procedures for those who don't want to unravel the puzzle for themselves. To summarize expenses, your initial investment will be about $350 for modem, handset, cables, software and CompuServe 'starter kit.' After that, a monthly expense of about $10 will provide a goodly amount of entertainment. The COLORCUE editorial office welcomes CompuServe mailbox communications, so you already have one place to 'call.' Here are addresses for two popular networks.

COMPUSERVE: 5000 Arlington Centre Boulevard, PO Box 20212, Columbus, Ohio 43220.

THE SOURCE: 1616 Anderson Road, McLean, Virginia 22102

* Note: For a detailed survey of networks available and the services they offer, see PERSONAL COMPUTING magazine, January 1984]

....*MORE BLUE SKIES*

Some primary objections to the CCII might be a) a limited software base (we don't have the rich variety that CPM or IBM/PC users have); b) only 63 characters per line (we can't have 120 column screen displays); c) 'poor' graphic resolution (no spectacular moving pictures); d) an inadequate disk storage system (not enough capacity, non-reliable operation); e) frequent failures from aging equipment (no service facilities, hard-to-get replacement parts.)

Sounds depressing, doesn't it! Yet, the fact is that I use my Intecolor equipment most of every day to work and play and I am not feeling deprived in any sense of the word. Sometimes I look with longing at ALT keys, 600 X 400 screen resolution and feel, frankly, a desire to escape to something new. It must be related to all the other feelings of disenchantment we sometimes feel—with our jobs, our spouses, our children, and.... ourselves. Yet, after spending any amount of time on another system, I'm always glad to get back. A closer look will show the picture isn't so bleek after all.

Tom Devlin has given us access to CPM and, with that, a good disk drive system. He has made satisfactory resolutions with the 63 character limitation of the CCII as a monitor for many, many programs. (CP/M was designed for a 64 character display, by the way.) With CPM programs and the supurb CCII utilities available, we have a software base more rich, and less expensive than most. (Have you ever added the cost of your software equivalent on CPM?)

Equipment failures are another matter. Compucolor (Intecolor) still provides factory service on the CCII. I have used it recently, accompanied with a carefully-worded plea for mercy, with satisfactory results. There are several fine service facilities still available from selected dealers and service stations across the country. (We'll provide an update on this for you very soon.) A major cause of CCII failure has been connected with the analog board. Tom Devlin has a 'saver' board for installation in the CCII which prevents one major failure mode. Write to him for information. We plan to print a review of this device in the next issue.

The graphics resolution we can't do much about, but if you have been keeping up with the spectacular releases in CCII game software you're not feeling too deprived.

Now to the blue skies..... with the current access to 4000H, we have a place to put, in ROM, disk handling routines for MD and FD standard drives in the 40 track, 128 byte/sector format. What we need is a System Chip that is modified to bypass CD disk handlers and jump to routines in the 4000H space (in ROM or RAM) for disk handling of MD (5" standard 92K floppy drives. 4000H might also contain a CD to MD backup program for transferring existing files. The keyboard edge connector is a satisfactory interface for such a system. If bank selector boards are in use, Bank 0, under software control, can be assigned the disk handling function, with automatic power-up to these routines. True, some software modifications might be required for existing programs, but not much.

Our hardware and software experts might consider this as a new product venture. To operate a CCII with MD reliability and speed would shed a new light on our 'old, outdated' machines. One or two chips and some drives, that seem to be reaching very low price levels, would do it. (8" drives are selling for as low as $150 each!) Let's have some response. Can it be done?!

QTZ

The following program, submitted by Tom Napier, is our first CUTIES in assembly language and must be entered with precision. It's great in a darkened room! Use [COMMAND/RESET] to stop, and reset Basic's pointers with [ESC] [W] before calling another program.

```
12 PLOT 6,3,12,3,20,10 : PRINT "DYNAMIC ELLIPSE DOODLER"
15 PLOT 3,25,15 : PRINT "BY TOM NAPIER" : GOSUB 300 : B=33282
40 POKE B,195 : POKE B+1,0 : POKE B+2,144 : PLOT 12
65 A=INT(65535*RND(1))
70 Z=CALL(A) : A=A+1 : IF A>65535 THEN A=A-65536
80 GOTO 70

100 DATA 229,33,0,0,66,14,127,83,30,127,205,128,144
110 DATA 205,0,145,124,181,194,10,144,225,201,256
115 REM
120 DATA 229,122,230,124,111,38,0,41,41,41,41,41,120
130 DATA 230,126,213,95,22,112,25,209,213,120,230,1
140 DATA 135,135,0,95,122,230,3,131,60,95,151,55,23,29
150 DATA 194,165,144,209,174,119,227,35,124,227,35,230
160 DATA 28,15,15,246,128,119,225,201,256
165 REM
170 DATA 229,120,167,31,47,103,121,31,0,111,35,25,84,93
180 DATA 122,167,31,103,123,31,111,9,68,77,225,201,256
190 REM
300 X=36864 : GOSUB 350 : X=36992 : GOSUB 350 : X=37120
350 READ Q : IF Q=256 THEN RETURN
360 POKE X,Q : X=X+1 : GOTO 350
```

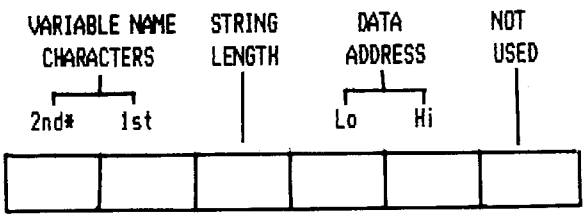
HOW BASIC STORES VARIABLES

Gary Dinsmore
32695 Daisy Lane
Warren, OH 97053

The Compucolor Programming Manual lists the address of the pointer to the start of Basic variables. This is a two-byte address at 32983 and 32983 (80D6H and 80D7H). If you use a debug program to extract this pointer address, you can look at the values in the variable storage area after a Basic program has been run, and while the variable data is still intact. You will find the variable names and other important information about them, stored in 6-byte groups.

Like most things having to do with the 8080 microprocessor, these bytes are stored "basackwards." To find the start of variable storage add 256 * PEEK(32983) to the value of the byte in 32982.

When you begin to examine the actual variable data you will need to know how to interpret it. The first two bytes are the variable name—second byte first. The first byte is coded to indicate whether the variable is a "string" or a numeric variable. A value greater than 128 indicates a "string variable", and the format in FIG 1. will be used.



*For 2nd character of variable name (ASCII), 128 = no 2nd character. Subtract 128 to get ASCII value of character

FIG 1.

If the value of the first byte is less than 128 then the variable is numeric. In this case, the value of the number stored in bytes 3, 4, 5, and 6 is a binary floating-point representation. Bytes 1 and 2 still represent the second and first characters of the variable name respectively. If a single letter variable name is used, then the first byte of the six bytes is a "0." The last four bytes (32 bits) of numeric value include a sign bit, 23 bits of floating-point precision, and 8 bits of ~~mantissa.~~ exponent.

Lets take a decimal analogy. The decimal number 132 can be represented in exponential notation as +1.32 E2. A binary number can also be represented in exponential form. In this case it will be +1.00001B E111B. We can interpret these two versions of the same number position by position.

First, decode the fractional parts of both numbers:

DECIMAL: +1.32 E2	**BINARY: +1.00001B E111B**
1 * 10^0 = 1	1 * 2^0 = 1
3 * 10^-1 = 3/10	0 * 2^-1 = 0/2
2 * 10^-2 = 2/100	0 * 2^-2 = 0/4
	0 * 2^-3 = 0/8
	0 * 2^-4 = 0/16
	1 * 2^-5 = 1/32

Next, decode the exponent for each number:

E2 in base 10 = 100 **E111B = 2^7 = 128**

Now multiply each number by the exponent:

1 * 100 = 100	1 * 128 = 128
3/10 * 100 = 30	1/32 * 128 = 4
2/100 * 100 = 2	
SUM = 132	**SUM = 132**

To see how this data fits into four bytes, lets draw a picture. This is how the six bytes for variable "X7" would look with the value 132 stored:

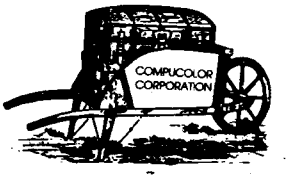
Byte	1	2	3	4	5	6
Binary	00110111	01011000	00000000	00000000	00000100	10001000
Hex	37	58	00	00	04	44
ASCII	'7'	'X'				

Lets rearrange the bytes in a more logical order:

Byte 2	Byte 1	Byte 5	Byte 4	Byte 3	Byte 6
01011000 'X'	00110111 '7'	00000100	00000000	00000000	10001000
		Sign bit + 23 bits of fractional precision for value of variable. 1.fffffffffffffffffffffff (The '1' is implied.)			Add 127 to this byte to get the power of 2 for the fractional part, = a number from 127 to -128.

The following Basic program segment can be typed into the last few lines of any program and used to list the variables and dump their values. Remember, only variables

that have been assigned are listed. The variable "A" is assigned first and then used to transfer the values of the other variables into a form that can be interpreted and reported by Basic—without resorting to an elaborate decoding scheme. (I let Basic do it!) [If "A" is used as a variable in your program, substitute an unused variable for "A". Ed.]




```
0 CLEAR : A=0
1 REM This assigns A first so we can find it.

59999 END : REM Protective end statement.

64000 Q3=PEEK(32982) + 256 * PEEK(32983)
64001 REM List out all variables except arrays

64010 FOR Q1=Q3 TO PEEK(32984) + 256 * PEEK(32985) - 1 STEP 6
64020 IF PEEK(Q1) > 127 GOTO 64200 : REM Handle strings.
64030 PRINT CHR$(PEEK(Q1+1)); CHR$(PEEK(Q1));"  ";
64040 POKE Q3+2, PEEK(Q1+2) : POKE Q3+3, PEEK(Q1+3)
64041 POKE Q3+4, PEEK(Q1+4) : POKE Q3+5, PEEK(Q1+5)
64042 REM Transfer value to A.

64050 PRINT A
64060 NEXT : RETURN

64200 PRINT CHR$(PEEK(Q1+1));
64210 A= PEEK(Q1) : IF A>127 THEN A=A-128
64220 PRINT CHR$(A);"$  ";
64230 FOR Q2=0 TO PEEK(Q1+2)-1
64231 PRINT CHR$(PEEK(PEEK(Q1+4)+ 256*PEEK(Q1+5)+Q2));
64231 NEXT Q2 : PRINT : NEXT Q1 : RETURN
```

The SOURCEBOOK is coming.....

Colorcue's **SOURCEBOOK** will be printed in Vol VI, No. 3, May/June 1984. It is designed to be a compendium of all materials available for the CCII and 3651 computers. If you have any material available for these machines, here is your opportunity to let Colorcue readers know who and where you are. It is particularly important that dealers who have not advertised for a few months or more make their continued presence known. Please contact our office promptly if you wish to contribute materials. There is no charge in the **SOURCEBOOK** for advertisements or copy of any kind. Editorial discretion will be used and space allocated as it is available.

The planned contents includes in part: an index of Colorcue. Forum, Data Chip and as many other publications that are made available to us; a complete documentation of ASCII; a cross reference of printer commands among Epson, IDS, Okidata, and C. Itoh (Apple and NEC); comparative ROM listings for v6.78, v8.79 and v9.80; a descriptive catalog of currently available "commercial" software; user group disk holdings; RS232 definitions and "standards"; list of current hardware available for CCII and 3651; up to date dealer list and repair center list; bibliography of tutorial books applicable to the CCII and 3651; a directory of current Colorcue subscribers with network access numbers (you may have this information withheld upon written request.)

ASSEMBLY LANGUAGE PROGRAMMING

PART XIII Printing the File / Finishing Up!

Joseph Norris
19 West Second Street
Moorestown, NJ 08057

[This article makes reference to Listings published in Colorcue, Jun/Jul 1983.]

If you have been constructing SOURCE.SRC with these articles, you know by now that the text format is not particularly elegant. Words at the end of the first text line 'wrap-around' to the second - sometimes with inglorious division of syllables. Well, it's up to you to deal with that problem. My point is that the print routine won't be any better. What you see on the CRT is what you'll get. The important topic for our consideration is how to get those bytes to the RS232 port.

THE PRINT ROUTINE (See Listing V.) It is only appropriate to print text from an open file, so we begin by verifying an 'open' file status. Sending our text characters to the outside world requires a setting of the BAUD rate (rate of character bit transmission per second)[1], setting the number of stop bits, and feeding the characters to be sent to the operating system subroutine.

BAUD RATE GENERATOR. The Baud Rate is set by addressing Output Port #5 in the CCII and 3600 series computers. It expects to see a number between 01H and 40H (see BAUDTB, Listing V) corresponding to baud rates between 100 and 9600. In the fifth through tenth lines of module PRINT (Listing V) we get the ASCII number entered at the keyboard representing the selected Baud Rate, and subtract 30H to convert it to the correct decimal number, 1-7. What happens next may be new to some readers; we will change our decimal number (1-7) to the proper code number for setting Baud Rate using a table of equivalents we have installed in memory at address BAUDTB ('baud table').

We will use the system routine **ADHLA** to add the contents of the accumulator (our selected baud number code) to the HL register pair. This has the effect of indexing the HL memory pointer to the correct corresponding conversion number in BAUDTB. [In Fig. 1, examination of the binary equivalents makes it clear that we are setting one of seven bit switches, one for each Baud Rate available.] We move the converted number into the accumulator and send it to Port #5, which is internally wired to the baud rate generator in the computer. [2]

STOP BITS. In BAUDTB, bit 8 is shown as '0.' This bit sets the number of stop bits; '0' = one stop bit, '1' = two stop bits. We can 'set' this bit by adding 80H to our number code in A (the same as adding 10000000B, thus setting bit 8.) Before addressing BAUDTB, our program is already set at two stop bits because we executed a PLOT 15 in string CLR. Since the data in BAUDTB will have been transmitted to the printer just before printing, it will supercede any previous settings.

THE PRINTER SETUP STRING. A 'setup string', here, means a string of characters you may want to send to your printer to set characters/inch, margins, paper positioning, etc. The actual characters used vary from printer to printer but are usually Escape sequences. Here is a string I might use with my C. Itoh 'Prowriter' to set 12 cpi and 6 lines/inch index spacing -

```
SETPR:  DB  27,69,27,65,0 ; ESC E, ESC A
```

The '0' has been added as an 'end-of-line' code so I may send these bytes using **S1OUT**, with 'CPI 0' to detect the end of the string. However, if I replace the '0' in the string with '239', and set **LOFL** to '0EH' (output to serial port), I can use **OSTR** to send the bytes (see COLORCUE, FEB/MAR p13 and be sure to change **LOFL** back to 00H afterward.) It is helpful to keep GTCHA in the routine, even if a setup string is used, to allow for aligning paper, etc. before printing commences.

S1OUT, the Send Routine. The label means 'send one out' and the second character is the numeral one, not the letter 'I'. This routine requires that the E register contain the character to be sent, before CALLing. **S1OUT** will obey the 'clear to send' line (the 'handshake' line) on the RS232 port, waiting patiently (forever, if necessary) for the printer to OK the send (digital '0'). If there is no printer connected, the byte will be 'sent' into thin air and the routine will RETurn.

Different printers may require some modification of the send routine with respect to carriage returns (cr) and line feeds (lf). My printer may be programmed to do a cr-lf both when either character is received (in non-graphics mode.) Technically, a carriage return only moves the carriage to the left side of the paper. Only a line feed indexes the paper up one line. In lines 19 and 20 of module PRINT, I send a line feed (and cr, because of my printer characteristic) to print a space between printouts of the text.

The C register, in the print routine, counts the number of characters printed per line and the B register counts the total characters to be printed. We point to INBUF with HL, move each character into the E register, send it, and then adjust the counters and pointer until we are finished. At the end of the first line (when C = 0) we branch to subroutine XX8 which indexes the paper for a new line. Here, the cr and lf are sent separately - as an example.

We said in Article XI, footnote [1], that printers interpret the ASCII characters between 1 and 31 differently than the Intecolor or Compucolor CRT. Fig. 2 is a table of the printer's interpretation with a brief explanation. While this chart shows the expected function of each code you must check your printer instruction manual for possible exceptions.

We have only three more modules to enter in SOURCE - those that provide the exit from our program to FSC, BASIC, and CRT mode. The BASIC exit is a routine I have used in these pages before, by M.A.E. Linden. With a 'D' in the H register, the operating system routine **ESC1** takes us back to

FIG 1. BAUDTB CONVERSION

===== POINTER =====	== ADDRESS ==	======= CONTENTS =======		
		Hex	Decimal	Binary
HL + (A=00H) ->	8507 BAUDTB:	00H	0	00000000
HL + (A=01H) ->	8508	01H	1	00000001
HL + (A=02H) ->	8509	02H	2	00000010
HL + (A=03H) ->	850A	04H	4	00000100
HL + (A=04H) ->	850B	08H	8	00001000
HL + (A=05H) ->	850C	010H	16	00010000
HL + (A=06H) ->	850D	020H	32	00100000
HL + (A=07H) ->	850E	040H	64	01000000

FCS (notice we are simulating the key presses 'ESC' and 'D')[3] if we have saved and restored the FCS stack pointer (which we have!) The CRT jump is not so elegant and may give unpredictable results. It should result in a blank screen - without the 'CRT mode' label, but, depending on what has taken place in the computer beforehand, may display a few extraneous characters. You **WILL** be in CRT mode, however. It would probably be best to restore the FCS stack before making this jump. You can add those program lines yourself, using FCSJMP as an example.

INITIAL ROUTINES. We begin by saving the FCS stack pointer for later use in exiting SOURCE.PRG. We also assign the beginning of our own stack [4]. This location is not to be left to chance, and if you have been lucky 'till now, your days are probably numbered. From its starting location, the stack works backwards toward 0000H, so adequate room must be provided for it. Normally numbers that go on the stack come off in equal quantity. A major exception occurs when a subroutine is terminated by a JMP instruction and not a RET. While provision for this kind of error is not an elegant justification for ample stack space, there are very clever ways of using the stack for storage of constants (as used in the FORTH language, for example) which sometimes justify the allocation of 'unusual' stack space.

We now initialize our data space. When the computer is turned on, the contents of memory, outside the control of the operating system initialization routines, is unpredictable. If the contents of buffer spaces and other data spaces is critical (control characters could cause program failure) it is best to clear them. We, therefore, clear our data spaces with '0's (SETUP) but our file buffer with 'spaces' (CLBF), which seems more appropriate for a text buffer.

There is one provision not made in this program which I consider essential. You might want to add it yourself. A good programmer will always provide some means of terminating an operation in progress if it may be done so with safety. Operations which do not lend themselves to this auto-termination must certainly include disk I/O, for directory damage can, indeed - will, result. But suppose you began a print cycle and found you had green paper in the printer and wanted violet paper? Unless the system interrupts are disabled, the keyboard will continue to be sampled many times each second, and could intercept your keypress. As a minimum feature, I like to provide each instance of required keyboard input with an 'escape' feature, not necessarily included in the option line. The HOME key could be used for this purpose. As an example, when the Baud Rate option line appears and one wishes to abort the print mode, the HOME key could return the operator to OPTION. The presence and functioning of such a procedure must be mentioned in your instruction manual.

An instructive modification to SOURCE.SRC is the redesign of the file name entry box to allow space for a full file specification. MCHAR will now be set to 15 so the following specification could be entered - `0:XXXXXX.YYY;NN` but keep the final cr in case the version number is not entered.

How's your subscription?

Past issues of Colorcue have been delayed in part because subscription renewals came in too late. We are few in number and each renewal is critical to our existance. Keep a close watch on your shipping label and when it reads "0" (or before it reads "0") send your renewal check promptly. Since we plan one full Volume at a time, send $3.00 for each issue of the current Volume due you. (There are six issues per Volume.) We are continuing our guarantee of six full issues or a refund for each unpublished issue. Those of you who were "oversubscribed" at the $2.00/issue rate will have that lower rate honored for VOL VI.

COMPUTER SHOPPER: FASCINATING ADVERTISING CIRCULAR WITH INTERESTING ARTICLES AND NEWS UPDATES. SAVINGS YOU WON'T BELIEVE! ANNUAL SUBSCRIPTION FOR 12 ISSUES IS $15, SENT 3RD CLASS MAIL. EACH ISSUE IS APPROXIMATELY 160 PAGES LONG AND CONTAINS EQUIPMENT, SOFTWARE AND BOOK REVIEWS, APPLICATIONS AND CONSTRUCTION ARTICLES ("HOW TO BUILD YOUR OWN ROBOT"), USER GROUP INFORMATION, AND CLASSIFIED AND COMMERCIAL ADVERTISEMENTS FOR EVERY POSSIBLE COMPUTER-RELATED ITEM. HIGHLY RECOMMENDED! [COLORCUE HAS NO CONNECTION OF ANY KIND WITH COMPUTER SHOPPER.] COMPUTER SHOPPER. PO BOX F. TITUSVILLE, FL 32781.

Another modification (which I highly recommend) might be to reformat the display of the 128 bytes so they make labelled fields appropriate to a data base. This was suggested in the last article and some of you have already done so. One reader changed his file extension default to 'ADR' for 'address book.' His field assignments were as follows:

```
NAME FIELD: 40 Characters
STREET ADDR: 40 Characters
CITY FIELD: 28 Characters
STATE & ZIP:  8 Characters
TELEPHONE: 12 Characters

TOTAL = 128
```

It is only a short step from this point in SOURCE.SRC to a working data base program that can search fields for specific ranges of data and display only targeted items.

This completes construction of SOURCE.SRC. Add the modules of Listing V. and test the PRINT and EXIT routines. If you have run into space limitations on your screen editor because you do not have a 32K memory, you may omit the comment fields. The project is hardly over because there is ample room for both the suggested modifications and those you create yourself (the most important ones!) It is true that 90% of your file requirements are contained in this simple program. As with most everything else in this life, success with it will be, primarily, the result of risk.

OTHER DISK ROUTINES. Here is a brief overview of some additional disk routines you can experiment with. There is a provision for 'getting' and 'putting' strings of bytes—or records—in disk files. A record, in this case, is defined as a string of bytes terminated by a line feed or form feed character, and the records are 'stacked' sequentially, one after the other, in the file. You are, of course, not limited to only 128 bytes.

PTREC (16BBH-v9.80,v8.79)(3285H-v6.78) will write such a string to disk. You must follow our previous procedure through CALL OPEN and then set these parameters:
a) Place record buffer address in BC; b) place the record length in DE, and c) place the FPB pointer in HL. d) Call PTREC.

Upon completion, A will be gone, BC will point just pass the last byte written, DE will be 0000H if no errors occurred, and the FPB pointer will still be in HL. If Carry and Zero flags are both set, the end of the file was reached before all bytes were transferred.

Another routine, **PVREC** (16B1H-v9.80,v8.79)(327B-v6.78) will behave in the same way as **PTREC** except that it will write the byte count of the DE registers into the file as the first two bytes, low byte first. These bytes may be retrieved when 'GETting' to simplify local storage or printing.

To 'get' bytes from either record routine above, use **GAREC** (168DH-v9.80,v8.79)(3257H-v6.78). The OPEN and RWSEQI routines must be performed before the first call only, as described in these articles. Then place a) the record buffer address in BC, b) the record buffer length in DE, and c) the FPB pointer in HL. At completion the registers will be as **PTREC** except that the DE register will show how many bytes were read. If both Carry and Parity flags are set a valid terminator was not seen. In this case, the specified number of bytes will be read, but the next call to **GAREC** will start reading at the next byte (and end at the first terminator it sees.)

Beyond the scope of this series are routines for reading and writing 'image' files to and from memory, and routines for reading and writing complete blocks (128 bytes) to disk file. (See Dale Dewey's publication, referenced in the FEB/MAR issue, or the ICS System Listing for details)□

[1] Each character transmission consists of a single 'start' pulse which tells the printer a character is to follow, an eight-bit 'burst' of pulses, representing the ASCII character (with the MSB unused - that is, set to '0') followed by one or two 'high' levels representing the 'stop' bits. These stop bits signal the end of a single character. The detection circuitry in some peripherals is 'fast' enough to be satisfied with one stop bit. Others require a longer pulse. If in doubt, two stop bits should be used. Theoretically, the faster the bits are transmitted, the faster they will print, but once baud rates become higher than 1200, there is usually no significant increase in printing speed with today's printers because the printing mechanism cannot respond that fast. 'BAUD' is an honorary unit of transmission rate, named after Baudot, inventor of an early character set for teletype transmission.

[2] Alert programmers will recognize another way to set baud rate and stop bits. Remembering the sequence 'PLOT 14,27,8,B', where PLOT 14 sets one stop bit and B is a number from 1 to 7, we may place such a sequence in a byte string, LXI H,BAUD and use **OSTR** to send it. If you arrange the byte string as shown here, the '7' (a default value for B) may be replaced by the selected number with an LXI H,BR and MOV M,A. However, conversion tables are important to assembly programming and great fun! If you're new to conversion tables, I suggest staying with the procedure in the listing.

```
BAUDTB: DB 00H,81H,82H,84H,88H
        DB 90H,0AH,0CH
```

If you want two stop bits, change BAUDTB as follows:

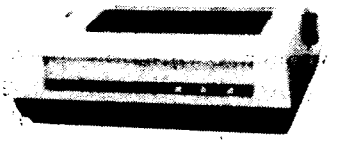
```
BAUD: DB 14,27,18 ;prepare ESC sequence
BR:   DB 7,239    ;add rate and 'end
```

[3] This method of re-entering FCS has engendered some controversy. It will work

#	NAME	FUNCTION
1	Null	No operation
2-7		Variable
8	BS	Back space print head
9	HT	Horizontal tab/next stop
10	LF	Line Feed
11	VT	Vertical tab/next set line
12	FF	Form Feed/top of form
13	CR	Carriage Return
14-23		Variable
24	CAN	Cancel prior text/this line
25-26		Variable
27	ESC	Escape Command
28-31		Variable

FIG 2. ASCII PRINTER CODES

'Variable' functions provide selection of type face, underline, boldface, elongated characters, graphics and all special functions. This group of codes will vry from printer to printer. Check your own manual for specific instructions.



the first time it's tried. But if you re-enter SOURCE with ESC T and attempt to exit to FCS a second time, it will fail.

[4] The stack provided is excessively generous! As a rule of thumb, for a program of this size, 80 bytes would be more than enough. The FCS stack, by the way, begins at 8044H in the 3651 computer, and at 8042H in the CCII. It works its way back toward screen memory which ends at 7FFF. It is interesting to observe the stack, which may be done with the MLDP (ICS), DBUG (Comtronics), or IDA (Bill Greene). The procedure is to insert a 'break point' at an appropriate place in the program running under the debugging software, and then make a 'dump' of the stack area for examination.. If the stack area is one designated by the user, it may be cleared with '0's before program execution to make tracing easier. □

PRINTER MATERIALS

Here are some sources you may want to tap for printer supplies such as continuous paper, labels, stationery, ribbons, and mailers. Catalogs are available, usually at no charge. If you suscribe to 'Computer Shopper' (see insert in this issue) you will find frequent 'specials' in this same category that can be real money-savers.

QUILL CORPORATION. 100 S. Schelter Road, Lincolnshire, IL 60069. Ribbons, bond papers, printwheels, continuous labels and paper, floppy disks, general stationery supplies.

NEBS COMPUTER FORMS. 12 South Street, Townsend, Massachusetts 01469. Continuous printed stationery, envelopes, file cards, labels (you name it!); computer furniture, disks, disk file devices, copystands, ribbons, computer hardware and furniture, business forms.

DELUXE COMPUTER FORMS. 530 North Wheeler Street, PO Box 43046, St. Paul, Minnesota 55164-0046. All kinds of custom forms, including checks, stationery, business utilities, labels, envelopes, mailers. They also sell disks. Somewhat more expensive.

VULCAN BINDER AND COVER. PO Box 29, Vincent, Alabama 35178. This is a great source for looseleaf binders in any variety you might want. They supply plastic sheets for holding disks in a 3-ring binder, disk carrying cases and files, and many other stationery items you're paying a lot more for now.

OBSCO. 11 Dalewood Lane, King Park, New York 11754. Labels and paper at extraordinary savings. Catalog may not be available, but you can telephone them at 516-360-1750. Sample of prices: 5000 labels 3-1/2 X 15/16, one across, continuous form, for $12.95; 1000 sheets continuous 'easy perf' 20- stock printer paper for $17.95.

BCCOMPCO. 800 South 17, Box 246, Summersville, MO 65571. I don't know the extent of their line, but printer ribbon sales are frequently advertised. Sample of recent prices; C. Itoh Prowriter and Epson: (new for C. Itoh) 12 for $96, (new for Epson) 12 for $66, (reloads—you send old cartridges for refill) $6 each for 2 or more, 'off-brand'—12 for $54. They advertise in COMPUTER SHOPPER.

MORROW USERS GROUP: A GROUP EXPLORING THE POTENTIAL OF THE MORROW MICRO DECISION HAS FORMED IN ORANGE, CONNECTICUT. CMDUG (CONNECTICUT MICRO DECISION USERS GROUP) MEETS ONCE EACH MONTH. A QUARTERLY, "CMDUG NEWSLETTER" COMES WITH THE MEMBERSHIP FEE OF $12 PER YEAR. FOR MEMBERSHIP AND DETAILS OF MEETINGS, WRITE DAVE MINTIE. CMDUG, 226 BOSTON POST ROAD , ORANGE, CT 06477.

COLORCUE BOOK SERVICE: Several major bookstores carry a line of computer books. Among these are Walton, B. Dalton, Doubleday, and Computerland stores. If you are having trouble gettings books and have exhausted your local resources, Colorcue will try to get them for you. We charge the price of the book plus a handling and mailing fee that runs between $2.00 and $5.00. You will be billed after the book(s) are sent. Just write to us with the author, title, publisher and number of copies.

NETWORK SUBSCRIBERS! Here's a beginning!]

COLORCUE: CompuServe 71106, 1302

Frepost Computers: Source TC1251, Micronet* 70210, 374

Intelligent Computer Systems: Source TCB610

David Suits, Ben Barlow: CompuServe 70045, 1062

Christopher Zerr: CompuServe 71445, 1240

*MicroNET is a part of CompuServe dedicated to personal computer user functions and activities. There is a surcharge for use of this service.

BASIC'S FILE STRUCTURE; A Review

Creating the File: The FILE "N" Statement.

Basic creates new files with the FILE "N" statement. The function of this statement is to define and reserve the disk space parameters for the file and to enter the file in the disk directory. This statement may be used to create any type of file. Those files with reserved type extensions, such as LDA, PRG, MAC, COM and so forth may be "created" but are not readily usable by Basic and should be avoided. Basic can create SRC, TXT, and DOC files and read and write to them. Any unreserved file extension type may be used including ones you specify yourself, but in cases other than the use of RND, the file will be a sequential file, best containing only ASCII characters. Non-ASCII characters read by Basic can cause disruption of Basic and should therefore be avoided. As a "reserved" file type, RND files will be recognized and treated in a special way by Basic.

The file to be created must not already exist on the disk in the addressed drive. By "exist" we mean that at least one parameter among file name, type and version number must be unique. If the file name and type already exist on the disk, Basic will create another like file with a higher version number.

The FILE "N" Statement has this general form -

FILE "N",F$,R,N,B

where F$ is the file name—a group of quoted ASCII characters, R is the number of records in the file, N is the number of bytes in each record, and B is the number of records that will be read into computer memory at one time.

The file name may have from one to six characters drawn from the alphabet "A" to "Z" and the numerals "0" through "9". The file name may not contain spaces or any other ASCII character not in the list above. Valid characters may be arranged in any order; for example, these are all valid file names —

12GOT.RND 4.RND IF56K.TRP I1TU.NOW 0.HAB

The file name may end with a specific version number if you wish to specify it. You may also preceed the file name with a disk drive specification, contained in the F$ string. For example —

F$ = "0:SAMPLE.FRK;09" or
F$ = "CD1:FROTH.CLD;1A"

Note that version numbers are in hexadecimal. The maximum number of versions varies with the system software. You may determine your maximum number through trial and error.

The number of records, R, to be contained in the file is a decimal number between 1 and 32767. For random files (RND), a "record" implies that you will construct a sequence of bytes which have meaning as a group, and that this same meaning is reflected in each and every record in your file. For example, a record might consist of a name, address and telephone number, meaning that every record will have these same data groups (and in the same sequence and each of the same field length). If you wish to store no more than 100 sets of names-addresses-telephones then you will assign the value 100 to R. For non-random files, a record may be anything you wish, with the requirement that each record will have the same defined maximum length. whether this entire length is used or not.

The number of bytes to be held in each record, N, is a decimal number between 1 and 32767. This number will be made large enough to contain the longest record you expect to store. If the "name" field can be no longer than 30 characters, the "address" field 40 characters, the "telephone number" field 12 characters, then the value assigned to N will be at least 82—all these fields in the same record.

When choosing the value of N, consider that Basic creates file space in integer multiples of the standard disk block of 128 bytes. If you assign a value of 1 to R, and 125 bytes to N, one entire disk block of 128 bytes will be assigned to your file, the last 3 bytes being "wasted." With the same single record, and a value of 129 bytes for N, two entire disk blocks will be assigned to your file with 127 bytes "wasted." A "wasted" byte cannot be used by another file.

When the value of R is greater than 1 however, bytes within a single disk block or several disk blocks may be shared among the records. Suppose R = 3 records and N = 85 bytes per record. The total number of bytes for three records is 3 * 85 = 255. This file will fit into two disk blocks (2 * 128 = 256) with one "wasted" byte. You can see how many "wasted" bytes exist in a disk block by examining the LBC (last block count) hexadeciaml number in the file directory. LBC tells you how many bytes are used in the last block of a file. If the number is 80H (=128) then every byte has been used. In this example, nothing would be gained by making N=84, because then three more bytes would be "wasted" in the disk blocks assigned to this file.

The blocking factor for the file, B, is a decimal number between 1 and 255 that infers how many blocks of file data you wish to be read into computer memory at one time, and, therefore, how much computer memory you wish dedicated to holding your file data. The maximum number 255 means that no more than 255 * 128, or 30640, data bytes may be read into computer memory at one time. Just as Basic allocates file space in integer multiples of the 128-byte disk block, so does it read blocks in multiples of 128 bytes. You cannot read only 67 bytes of data from disk into computer memory, nor can you read only 129 bytes of data into computer memory. Any bytes unused (not subject to GET) will still be in computer memory if they lie within a partially-specified 128-byte block.

When assigning the blocking factor it is customary to choose a number that will coincide with an integer number of records, and, at the same time, an integer number of disk blocks. If N=128 bytes, then each integer value for B will bring that number of records into computer memory. If N=64, then a value of 2 for B will bring two records into computer memory at one time. If N=64 you cannot have a blocking factor of 1 (=64 bytes—an error message will result) since disk blocks can only be read in integer multiples of 128. For N=64, B must be in multiples of 2 (2 * 64 = 128 bytes).

As the value for B determines the amount of user computer memory you want dedicated to file data storage (B * N) it also defines an additional memory space which will hold the file access parameters required by Basic. The Com-

pucolor Manual cites a formula for computing the exact memory space requirement. If there is sufficient room in user memory for many files, then the only penalty for reading a large number of records into computer memory is the disk access time. If all the records will not be needed (as they might be for procedures such as sorting) the needed record may be accessed without bringing all previous records into memory. (See FILE "R" next issue.) In summary, the smallest disk access time will be achieved by assigning a value to B that brings only the required number of records into computer memory.

All parameters in FILE statements may be expressed as variables. You may assign numbers to R, N, and B and a string to F$ and state FILE "N",F$,R,N,B to open a file.The string file name, F$, may be a concatenation of strings, and R,N, and B may be members of numeric arrays as in these examples:

FILE "N",N$+T$,L(3), K(2),MW(20), or
FILE "N",STR$(D)+T$,J(X),YM(A,B),BY(R,T,U)

Remember that the FILE "N" statement is only concerned with the file name and the file disk space requirement. If N * B permits the reserving of sufficient file space then you are free to distribute this space later in another way when the file is opened by FILE "R". For example, I can create TEST.RND with R = 1, N = 512, B = 1, but actually use it in a FILE "R" statement with R = 64, N = 8, and B = 16. In both cases, the reserved disk space is the same.

It is helpful to experiment creating files in "immediate mode", using different parameters, to see which Basic will accept. Apart from the specifics of parameters discussed here, you must only observe that all parameters, F$,R,N, and L are specified. As a further example, if you wished to create a SRC file 5120 bytes long, and access it all at once, then the create statement might read—

FILE "N","TEST.SRC",1,5120,1.

Next time we will continue with a discussion of the FILE "R" statement. □

NEXT ISSUE:Using Morrow MicroDecision with the CCII; An inexpensive pen plotter you can use; ASCII, Masks and BCD; Review of Basic's file structure; Simple encryption in assembly language; Peter Hiner's FASBAS and more!

Back issues of COLORCUE contain a wealth of practical information for the beginner as well as the more advanced programmer, and an historical perspective on the CCII computer. Issues are available from October 1978 to current.

DISCOUNT: For orders of 10 or more items, subtract 25 % from total after postage has been added. **POSTAGE:** for U.S., Canada and Mexico First Class postage is included; Europe and South America add $1.00 per item for Air Mail, or $ 0.40 per item for surface; Asia, Africa, and the Middle East add $ 1.40 per item for Air Mail, or $ 0.60 per item for surface. **SEND ORDER** to Ben Barlow, 161 Brookside Drive, Rochester. NY 14618 for VOL I through VOL V; and to Colorcue, 19 West Second Street, Moorestown, NJ 08057 for VOL VI and beyond.

1978	**VOL I**	**$3.50 each**
	No. 1-3:	OCT/ NOV/ DEC
1979	**VOL II**	**$3.50 each**
	No. 1-3:	APR/MAY/JUN
	No. 4-5:	JAN/FEB/MAR
	No. 6-7:	AUG/SEP/OCT
	No. 8:	NOV XEROX COPY, $2.00
1980	**VOL III**	**$1.50 each**
	No. 1	DEC/JAN
	No. 2:	FEB
	No. 3:	MAR
	No. 4:	APR
	No. 5:	MAY
	No. 6:	JUN/JUL
1981	**VOL IV**	**$2.50 each**
	No. 0:	DEC/JAN
	No. 1:	AUG/SEP
	No. 2:	OCT/NOV
1982	No. 3:	DEC/JAN
	No. 4:	FEB/MAR
	No. 5:	APR/MAY
	No. 6:	JUN/JUL
	VOL V	
	No. 1:	AUG/SEP
	No. 2:	OCT/NOV
1983	No. 3:	DEC/JAN
	No. 4:	FEB/MAR
	No. 5:	APR/MAY
	No. 6:	JUN/JUL
1984	**VOL VI**	**$3.50 each**
	No. 1:	JAN/FEB

UNCLASSIFIED ADVERTISEMENTS

FOR SALE: Compucolor II, v6.78, 32K, extended keyboard, manuals including Programming, Maintenance, "Color Graphics" and "Basic Training." 15 disks included. Excellant condition. Asking $1000. Art Tack. 1127 Kaiser Road, SW, Olympia, WA 98502.

FOR SALE: Compucolor II computer, v8.79. Basic keyboard only, 32K memory, switchable lower case character set, CRT filter, handshake option installed. Very good condition. $800. Joseph Norris c/o The David Hafler Company, 5910 Crescent Blvd, Pennsauken, NJ 08109. 609-662-6355. (No, I'm not abandoning ship—I still have three more!)

The following items are from the estate of Myron T. Steffy.

FOR SALE: Two Compucolor II computer systems, v6.78. Each computer has full keyboard, 32K memory, dual disk drives, Devlin Analog Protector, 2 Devlin Ram Cards with software switch, lower case character set, and character generator for FREDI. One of the above units has the Comtronics updated ROM. Excellant condition. $1200 each.

Morrow MicroDecision CP/M computer with dual disk drives, 64K memory and software package. No monitor. $900.

Novation CAT, 300 Baud modem, Votrax unit, TI SR52 calculator and TI programmer's calculator, 2 MFT transfer switches for RS232 outputs. All in good condition. Make offer.

Diablo Model 630 letter quality printer. Cost $1850 wher new. Excellant condition. Make offer.

Address inquiries to Bill Shanks. 1345 West Escarpa, Mesa AZ 85201. 602-962-0130.

Whence did the wond'rous mystic art arise,
Of painting Speech, and speaking to the eyes?
That we by tracing magic lines are taught,
How to embody, and to colour thought?

COLORCUE VOLUME VI, NUMBER 3 MAY/JUNE 1984

Colorcue

VOLUME VI, NUMBER 3 MAY/JUNE 1984

CONTENTS

COVER: Text set by Carl Remley in pen and ink. Unknown source, quoted from CompUKolour.

EDITOR: JOSEPH NORRIS COMPUSERVE: 71106, 1302

COLORCUE is published bi-monthly. Subscription rates are US$18/year in the U.S., Canada, and Mexico (via First Class mail), and US$30 elsewhere (via Air Mail). All editorial and subscription correspondence should be addressed to COLORCUE, 19 West Second Street, Moorestown, NJ 08057, USA. (609-234-8117) Every article in COLORCUE is checked for accuracy to the best of our ability but is not guaranteed to be error free.

"Welcome to the Sourcebook...

You will find we couldn't deliver all we promised for this issue of Colorcue. In spite of written requests for information, mailed in January, many respondents sent materials at press time—too late. Others replied in such a vague way that nothing less than a divining rod could have decyphered the meaning. Still others used a handwriting (with faint pencil) that could only defeat any mortal interpretation. The fault is not to be entirely externalized, however. We simply bit off more than we (and our bank account) could chew. Not to worry. The promise will be fulfilled in time. Meanwhile, I hope this issue is as interesting to you as it has been to me.

Welcome to those of you who have rejoined us, and to those who are new to the CCII by virtue of purchasing used computers. You are discovering that we have a lot to offer (at a reasonable price) in the Compucolor community. There are at least 200 of us at work with the CCII, and an unknown number of others with the 3651 and 8000 series who are just now finding us. I have been working with a few 8000 users in an attempt to adapt CCII software to their machines. They use FCS, but the memory mapping is very different. As of this writing, things look promising. This might mean a somewhat revitalized software market for Com-tronics, Bill Greene and anyone else still willing to try.

The highlight of my Compucolor experiences since the last issue was a visit to Rochester and the user group there. Imagine, if you can, eleven or so owners, all gathered together in the same place. They've been doing it steadily for years, and I imagine they have an inadequate appreciation for the comradeship and mutual support CHIP gives them. The topic for the evening was FORTH, one in a series of talks by

.....ah...that is....Part 1."

Jim Minor on the FORTH language. I delighted in putting a "face" on Gene Bailey, Doug Van Putte, Rick Taubold, Joseph Charles, Jim Minor, David Suits, and all the other names I've known only in print. Dr. and Mrs. Suits were my gracious hosts for a night. David and his wife have started a new magazine for the NEC APC computer, called NEXUS. It has a familiar format, and the contents remind one of early CCII days—starting all over again. It is an appropriate computer for David—spectacular graphics, rather complicated operating system, and a challenge for him for a few years, I imagine. My thanks to the CHIP group for letting me sit in, and I hope to see you all again soon. Wouldn't it be wonderful if CHIP could arrange a grand reunion for all of us? How about an International CCII Conference for a weekend in Rochester, open to all Compucolor and Intecolor users? Now that's entertainment!

There are some new authors coming on the scene. I guess rightly that our group has hidden talents. Witness the beautiful cover on this issue by subscriber Carl Remley of Mississippi. His subscription renewal came written in calligraphic form and I was prompt to enlist his services. (It is appropriate that we express more than our computer interest in these pages.) The text is reprinted from the first issue of COMPUKOLOR, of the user group in the United Kingdom. (I do not have the source of this quotation. Does anyone know it?

So, on to the first part of the SOURCEBOOK. Rest assured that all submitted materials will be published, and all promises fulfilled. We look forward to your new materials as we continue our sixth year of publication. Thanks to the many of you who have sent notes of appreciation, and... keep the cards and letters coming.

John.

A Pascal for the CCII

Part II.

Doug Van Putte
18 Cross Bow Drive
Rochester, NY 14624

In Part I, a subset of Pascal, called Tiny-Pascal, was introduced through the fig-Forth language. Since Tiny-Pascal uses the facilities of Forth, such as editing, compiling and disk handling, Part I told how to obtain and implement both Forth and Tiny-Pascal on the CCII. In addition, Part I presented the Forth editor commands required to enter, edit, store, compile, and run Tiny-Pascal programs.

We will proceed by introducing some programming techniques that relate to the structure of a Pascal program. Pascal is designed as a "block-structured" language. The blocks are the basic building units of the program, so it is important to have a good conception of them. Think of blocks as logical subdivisions of program functions. Writing a program to bake a cake? Then design blocks to handle each part of the job: read the recipe, measure the ingredients, mix the ingredients, and perform the baking. The effectiveness of this block structure might not seem apparent in a short program, but in very complex programs it provides a clear advantage over Basic command structures.

A block structure enables the distribution of a complex program task into simple, smaller program tasks, which may be tested and debugged in themselves as they are created. Then, as the blocks are connected together, debugging will be limited to problems in the interaction among the blocks, greatly reducing debugging time. This procedure may be followed in Basic as well, if you exercise self-discipline, but the structure of Pascal syntax dictates these good programming practices naturally. A properly-written Pascal program is more readily accessible by other readers, and years from now it will reveal itself with the same clarity it had at its conception. A typical block-structured outline of a program is shown in Listing 1. The middle column contains the actual program code. The first and third columns are comments.

As the listing begins, we see a program name (PROGRAM DUMPCAKE). We see a declaration of a constant OVENTEMP, and a declaration of the variables CUP which is to be an integer number, and TEASPOON which is also to be an integer. In Pascal, constants and variables must be declared (identified) before they can be referenced by the program. This treatment is distinctly different from many versions of Basic, where one can assign a non-dimensioned numeric or string variable anywhere in the body of a program without regard to its type (real or integer).

Next we see a procedure, PROC RECIPE (procedure "recipe"), and a function, FUNC PROPORTION (function "proportion") specified. RECIPE might tell how to mix the ingredients; PROPORTION might describe how to proportion the mixture for various sizes of cake. These are "subroutines" which can be "called" by the main program. The declaration of variables and the specifying of procedures and functions are placed at the head of the program so they may be "seen" by the program before it begins execution. (This is much like reading data statements in Basic, before using the data.) You will notice that both the procedure and function portions may be "sub-programs", of several lines, each marked with a "BEGIN" and "END" to define the scope of each.

Now we approach the main body of the program which schematically represents several groups of coded instructions. Some elements of the group are "nested" (like FOR....NEXT

statements in Basic) and others are sequential. We use indentation in the listing to help the eye separate the groupings according to the sequence in which they perform.

Notice the punctuation after the END statements. Those END statements that are within the body of the program are followed by a semi-colon to indicate that the program continues. The final END statement is followed by a period (.) to indicate the conclusion of the program. Of course a program may have a wide variety of these BEGIN...END sections in all combinations.

The listing also contains statements set within the reverse backslash marks, which are comment sections (like REM statements in Basic.) Pascal differs from Basic in that comments may be placed both before and after a code statement. The compiler will find the code statement and extract it.

LISTING 1. Schematic outline of a Pascal Program.

```
\PASCAL PROGRAM\     PROGRAM DUMPCAKE;              \MAIN BLOCK HEADING\

\DECLARATIONS\        CONST OVENTEMP = 350;         \BEGIN DECLARATION BLOCK\

                      VAR CUP: INTEGER;

                          TEASPOON: INTEGER;        \END DECLARATION BLOCK\

\PROCEDURES\          PROC RECIPE;                  \BEGIN PROCEDURE BLOCK\

                         BEGIN
                            ---------------
                            ---------------
                            ---------------
                            ---------------
                         END;                       \END PROCEDURE BLOCK\

\FUNCTIONS\           FUNC PROPORTION;              \BEGIN FUNCTION BLOCK\

                         BEGIN
                            ---------------
                            ---------------
                            ---------------
                         END;                       \END FUNCTION BLOCK\

\MAIN PROGRAM\        BEGIN                         \BEGIN MAIN BLOCK 1\
                        --------------------
                        --------------------
                        BEGIN                       \START BLK 2\
                          ------------------
                        END;                        \END BLK 2\

                        BEGIN                       \START BLK 3\
                          ------------------
                          BEGIN                     \START BLK 4\
                            ----------------
                            ----------------
                          END;                      \END BLK 4\
                          ------------------
                        END;                        \END BLK 3\
                        --------------------
\END OF PROGRAM\      END.                          \END MAIN BLOCK 1\
```

Let's see how to specifically design a Tiny-Pascal program through the FORTH language. We will do this by writing a program to compute the area in a 2 by 4 rectangle (Listing 2). The spaces and punctuation are required, as shown, both for FORTH and Pascal syntax. Since we must first proceed from FORTH, we will follow a FORTH convention by placing the name of our program in parentheses on the first line. Parentheses are "delimiters" (comment markers) in FORTH. This will be followed by the FORTH words "PASCAL", which invokes the Tiny-Pascal interpreter, and "DECIMAL", which tells FORTH that our numerical data will be entered in decimal format. (We could have said "HEX" instead, if we meant to use hexadecimal numbers.) This is all we need to do to satisfy FORTH's requirements.

Now we write line 2 for Pascal, the Pascal word PROGRAM followed by our program name RECTANGLE AREA. Observe the terminating semi colon. Next we can see that two con stants are declared (LENGTH and WIDTH) and values assigned to them. In choosing the names of these constants, and the names of all the elements of our program, such as the program name, variables, procedures and functions, Pascal allows us to use both letters and digits to a length of 31 characters. This permits the names to be precisely descriptive.

A variable is declared next (AREA) and it is declared as an integer value. Notice the colon following the declaration of a variable. Now the "main" program begins and we equate AREA to the product of LENGTH and WIDTH. NEWLINE advances control to the beginning of a new line, and WRITE, with its printed prompt, prints the product on the CRT. A BEGIN statement and END statement bracket this main program block.

While this program could be duplicated with a single statement in Basic, it serves as an example of Pascal.

As the program becomes more complicated, Basic soon loses it's advantage of brevity, and the clarity of Pascal emerges. Among the peculiarities of punctuation, notice the semicolon that terminates each statement line, the ":=" used to establish an equate, the use of '.....' to delimit text in the WRITE statement, and "-" to designate printing the value of AREA. NEWLINE is the only "cursor position ing" statement in Tiny-Pascal, and gives direction to commence subsequent printing on the next line.

You may enter the program in Listing 2 on a blank FORTH screen using the editor. Check it carefully for spaces and punctuation. FLUSH the screen to disk. To compile the program, type [n] LOAD, where [n] is the number of the screen containing this program. From this point, you may "run" the program by typing RECTANGLEAREA.

Our program would be more functional and more interesting if we could enter new parameters for LENGTH and WIDTH from the keyboard. The READ statement in Pascal makes this possible, and operates like the "INPUT" statement in Basic. To prompt an input, the WRITE statement is used, followed by READ. We will have to change LENGTH and WIDTH from constants to integer variables, and insert a few statement lines just after the BEGIN in Listing 2. The changes are shown in Listing 3. Add them to the FORTH screen with the editor, and proceed as before to compile and run the amended program. Press RETURN after entering "length", and again after entering "width."

If you want to try some more adventurous programing, here are some other arithmetic operations available in Tiny-Pascal. "+", "-", and "*" are used as in Basic. "DIV" is used in Tiny-Pascal for integer division. The "/" symbol is reserved for real number division, and is not supported by Tiny-Pascal. We will be looking at other Pascal constructs, such as IF...THEN, WHILE...DO, FOR...DOWNTO, CASE...OF, and REPEAT...UNTIL. These operators, along with some exploration into PROC and FUNC organization will provide many possibilities for practical, interesting programs. Good luck, and good learning! □

Listing 2. A working program to calculate and print area.

```
( RECTANGLEAREA ) PASCAL DECIMAL    \FORTH commands\

PROGRAM RECTANGLEAREA;              \computes and prints\
                                    \area=l*w\

CONST                               \Declare constants\
  LENGTH = 4;
  WIDTH = 2;
VAR                                 \Declare variable\
  AREA: INTEGER;

BEGIN                               \Main program\

  AREA := LENGTH * WIDTH;
  NEWLINE;
  WRITE ( 'The area is ', #AREA);

END.                                \End of program\
```

Listing 3. Changes to RECTANGLEAREA to permit keyboard entry.

```
\Change declaration of these labels to variables\

   LENGTH, WIDTH: INTEGER;

\Add these lines after BEGIN\

   NEWLINE;                 \Print crlf\

   WRITE ( 'Enter the rectangle length & width ' );

   NEWLINE;

   READ ( #LENGTH,#WIDTH );  \Read two variables\
                             \from keyboard     \
```

Errata: In Part I, the screen editor command to copy one screen to another was incorrect. "COPY [num1] [num2]" should be corrected to read "[num1] [num2] COPY"

This table is compiled from the system listing of three software versions of FCS. (- - -) = NA (not applicable); (• • •) indicates that the address is the same as in the preceeding column. Frequently used labels appear in boldface.

LABEL	v6.78	v8.79	v9.80
A7ON	38E8	0331	0300
ACRTSP	0036	• • •	• • •
ADDU	2144	1AE9	1B76
ADHLA	**3518**	**194E**	• • •
ADJTKS	- - -	- - -	1C43
AESCTB	000B	• • •	• • •
ANHD	351D	1953	• • •
ASCPL	3DFB	0859	07FB
AUCNT	81B3	• • •	- - -
AUTOX	0058	1F3E	• • •
B2HEX	**33AA**	**17E0**	• • •
B7ON	3A19	0462	0431
BA7OF	3946	038F	035E
BARTX	3D5F	07BD	075F
BARTY	3D57	07B5	0757
BARTZ	3D51	07AF	0751
BARXM	3C13	066A	060C
BARYM	3C42	0699	063B
BARYM1	- - -	- - -	063E
BARYM2	- - -	- - -	064A
BASEX	**0055**	**1F3B**	• • •
BASFL	**81F1**	• • •	• • •
BASICE	**0046**	**1F2C**	• • •
BASICI	0052	1F38	• • •
BASICW	0040	1F26	• • •
BASORG	0040	1F26	• • •
BASOUT	**0033**	• • •	• • •
BAUD	**0005**	• • •	• • •
BC01	35B2	19EB	• • •
BC2BK	35A8	19DE	• • •
BCCIX	3A05	044E	041D
BCHK	3292	16C8	• • •
BCHK1	32BB	16F1	• • •
BCRSX	3A2A	0473	0442
BCRSY	3A37	0480	044F
BEGEX	**0038**	• • •	• • •
BEGIN	**3768**	**01B6**	• • •
BEGOT	3A59	04B0	047F
BEL	**3AC3**	**051A**	**04E9**
BEL1	- - -	- - -	04EE
BFILL	81D0	• • •	• • •
BFSX	- - -	- - -	001F
BHLAD	**81D4**	• • •	• • •
BK2BC	35BA	19F0	• • •
BKCOL	3928	0371	0340
BLIND	**3A09**	**0452**	**0421**
BLINK	393F	0388	0357
BRAKE	3AB6	050D	04DC
BRATE	3A09	0452	0421
BRATX	3A6F	04C6	0495
BREAK	**003B**	• • •	• • •
BRTR	- - -	- - -	0010
BRTRY	80E0	• • •	- - -
BRTX1	- - -	- - -	049D
BS01	35ED	1A23	1E4E
BS02	35F1	1A27	1E52
BS03	- - -	- - -	1EB2
BS04	365F	1A95	1EFA
BS10	237E	1D01	- - -
BS11	2389	1D0C	- - -
BS12	239D	1D20	- - -
BS13	236B	1CEE	- - -
BSB01	35C7	19FD	1E26
BSB02	- - -	- - -	1E92
BSB03	- - -	- - -	1E98
BSB04	- - -	- - -	1E84
BSB05	- - -	- - -	1EBD
BSB06	- - -	- - -	1EC9
BSB07	- - -	- - -	1ED1
BSB08	3652	1A88	1EED
BSDHU	- - -	- - -	19DE
BSTR	33E9	181F	• • •
BUCNT	81B4	• • •	- - -
BUFP	**8047**	• • •	• • •
BXLOP	3C30	0687	0629
BYLOP	3C67	06BE	0660
BYLOP1	- - -	- - -	066F
C3C9	- - -	- - -	011F
C3C95	- - -	- - -	012A
CARET	000D	• • •	• • •
CARR	3B4E	05A5	056C
CBC	30F5	152B	• • •
CBC1	3108	153E	• • •
CBC2	3127	155D	• • •
CCI	3A09	0452	0421
CCIX	3A01	044A	0419
CD03	2215	1BC1	- - -
CD04	2219	1BC5	- - -
CDHD	211C	1AC1	- - -
CDK	3695	004A	- - -
CDMK	002E	• • •	• • •
CDNM	3691	0046	- - -
CDNU	3693	0048	- - -
CDMK	- - -	- - -	002E
CDRSET	218B	1B30	- - -
CDSEC	3694	0049	- - -
CEN01	2AA5	0F48	0F45
CENTR	2A9E	0F41	0F3E
CHAIN	3A09	0452	0421
CHDEL	000E	• • •	• • •
CHDLR	2EFC	1332	• • •
CHPLO	39FD	0446	0415
CHTIM	001C	• • •	• • •
CKEND	26E7	0B8A	• • •
CLOSE	**2F26**	**135C**	• • •
CLSEQO	**3136**	**156C**	• • •
CLX	2859	0CFC	• • •
CMASK	81E0	• • •	• • •
CMDTMP	- - -	- - -	81B1
CMPDH	**3453**	**1889**	• • •
CMPHD	**344D**	**1883**	• • •
CMT1	2AB4	0F57	0F54
CMT2	2ABF	0F62	0F5F
CMTAB	257A	0A58	08D4
CODE	3996	03DF	03AE
CODE2	39A2	03EB	03BA
COLFL	**81E6**	• • •	• • •
COLOR	3907	0350	031F
COLW	392C	0375	0344
COMND	0004	• • •	• • •
COMOF	393B	0384	0353
COMON	393A	0383	0352
COP00	2B20	0FC3	• • •
COP01	2B38	0FDB	• • •
CPLOX	3E03	0861	0803
CPYDV	2C77	10AD	• • •
CR	3872	02BB	028C
CR1	24B3	1E36	- - -
CR2	24B8	133B	- - -
CR3	24BF	1E42	- - -
CR4	24C3	1E46	- - -
CRATE	**81E2**	• • •	• • •
CRC	247D	1E00	- - -
CRC1	8043	• • •	- - -
CRC2	8044	• • •	- - -
CRCX	249F	1E22	- - -
CRET	3B56	05AD	0574
CRLF	338B	17C1	• • •
CRSLT	38F0	0339	0308
CRSRT	38B5	02FE	02CD
CRSUP	38F6	033F	030E
CRSUP1	- - -	- - -	0316
CRSXY	3809	0452	0421
CRSY	388D	02D6	02A7
CRT0	0060	• • •	• • •
CRT1	0061	• • •	• • •
CRT2	0062	• • •	• • •
CRT3	0063	• • •	• • •
CRT4	0064	• • •	• • •
CRT5	0065	• • •	• • •

LABEL	v6.78	v8.79	v9.80
CRT6	0066	• • •	• • •
CRTCHIP	0060	• • •	• • •
CRTMO	25B7	01A7	• • •
CRTMSG	25C6	0123	0040
CRTR	0003	• • •	• • •
CRTRAM	81AF	• • •	• • •
CRTRY	80E2	• • •	81AE
CRTSET	37C0	020F	01EB
CRTST1	- - -	- - -	01F0
CRTUBE	396B	03B4	0383
CRXDA	006C	007C	• • •
CRYDA	006D	007D	• • •
CTRK0	81B1	• • •	- - -
CTRK1	81B2	• • •	- - -
CTWO	2C69	109F	• • •
CTYP	- - -	- - -	109A
CUCNT0	81B5	• • •	- - -
CUCNT1	81B6	• • •	- - -
CURS0	3A0B	0454	0423
D1	358A	19C0	• • •
D2	359F	19D5	• • •
D5CNT	- - -	- - -	6F84
D8CNT	- - -	- - -	6F83
DATAM	005A	• • •	- - -
DBF	811D	• • •	• • •
DBFE	819D	• • •	• • •
DBLK	811D	• • •	• • •
DDFHD	- - -	- - -	1BAB
DDFHV	- - -	- - -	1B05
DDFNM	- - -	- - -	1B08
DDFNU	- - -	- - -	1B0A
DDFFC	- - -	- - -	1B0B
DDFTK	- - -	- - -	1B0C
DDMHD	- - -	- - -	1B23
DDMHV	- - -	- - -	1AED
DDMNM	- - -	- - -	1AF0
DDMNU	- - -	- - -	1AF2
DDMRT	- - -	- - -	1AF5
DDMSC	- - -	- - -	1AF3
DDMSF	- - -	- - -	1AF6
DDMTK	- - -	- - -	1AF4
DEL00	29B5	0E58	• • •
DEL01	29D6	0E79	• • •
DEL06	2A2B	0ED0	• • •
DEL07	2A3C	0EDF	• • •
DEL08	2A44	0EE7	• • •
DEL09	2A6F	0F12	• • •
DEL10	2A8A	0F2D	• • •
DELER	2A9B	0F3E	0F3B
DELTA	000F	0014	- - -
DEV00	2996	0E39	• • •
DFDHD	- - -	- - -	1BDD
DFDHV	- - -	- - -	1B11
DFDNM	- - -	- - -	1B14
DFDNU	- - -	- - -	1B16
DFDSC	- - -	- - -	1B17
DFDSF	- - -	- - -	1B1A
DFDTK	- - -	- - -	1B18

LABEL	v6.78	v8.79	v9.80
DFDV	80F0	• • •	• • •
DFH	- - -	- - -	1B03
DFUN	80F2	• • •	• • •
DIG	3476	18AC	• • •
DIP	2D80	11B6	• • •
DIPS	0006	• • •	• • •
DIR00	2791	0C34	• • •
DIR01	2799	0C3C	• • •
DIR02	27C5	0C68	• • •
DIR03	2816	0CB9	• • •
DIR04	282A	0CCD	• • •
DIR05	27D9	0C7C	• • •
DISPCK	81BC	• • •	• • •
DIVHD	3581	19B7	• • •
DMDHD	- - -	- - -	1B52
DMDHV	- - -	- - -	1AF9
DMDNM	- - -	- - -	1AFC
DMDNU	- - -	- - -	1AFE
DMDRT	- - -	- - -	1B01
DMDSC	- - -	- - -	1AFF
DMDSF	- - -	- - -	1B02
DMDTK	- -	- - -	1B00
DOWN	3A90	04E7	04B6
DSBUF	6000	• • •	• • •
DSBUFS	1000	• • •	• • •
DSEC	226A	1C10	1CE7
DUP00	2B93	0120	- - -
DUP02	2BD9	- - -	- - -
DUPLX	81DD	• • •	• • •
DX01	- - -	- - -	1C74
DX02	- - -	- - -	1C8F
DX03	- -	- - -	1CC6
DX04	- - -	- - -	1CE0
DX05	- - -	- - -	1CE3
DX10	2132	1AD7	- - -
DX11	2149	1AEE	- - -
DX12	2159	1AFE	- - -
DX13	2169	1B0E	- - -
DX13A	216E	1B13	- - -
DX14	21AB	1B50	- - -
DX14A	21C3	1B68	- - -
DX15	21D1	1B76	- - -
DX16	21E6	1B8B	- - -
DX17	21E6	1B8B	- - -
DX18	21F0	1B95	- - -
DXCM1	- - -	- - -	1C18
DXCM2	- - -	- - -	1C26
DXX	2201	1BA6	1C67
DZ01	- - -	- - -	1BC7
DZ02	- - -	- - -	1BD3
DZ03	- - -	- - -	1BD8
DZ04	- - -	- - -	1C0B
DZ05	- - -	- - -	1C13
DZ10	- - -	- - -	1C1F
DZ20	- - -	- - -	1C38
EARLN	3AF8	054F	051E
EARLN1	- - -	- - -	0527
EARS	3839	0282	0253

LABEL	v6.78	v8.79	v9.80
EARS0	- - -	- - -	0255
EARS1	- - -	- - -	025A
EBLF	004B	0045	• • •
EBLK	0009	• • •	0003
ECFB	000F	• • •	0006
ECOP	0045	003F	• • •
EDCS	0015	• • •	0009
EDEL	003C	• • •	003C
EDEN	0024	• • •	- - -
EDFN	- - -	- - -	0024
EDIR	000C	• • •	000C
EDRF	002D	• • •	• • •
EDFY	0012	• • •	- - -
EDUP	003F	- - -	- - -
EFCS	3372	17A8	17A6
EFNF	002A	• • •	• • •
EFRD	0030	• • •	• • •
EFWR	0033	• • •	• • •
EHCS	- - -	- - -	000C
EIVC	0003	• • •	• • •
EIVD	001E	• • •	• • •
EIVF	0003	• • •	000F
EIVP	000F	• • •	• • •
EIVT	0048	0042	• • •
EIVU	0006	• • •	0012
ELIN	334E	1784	1782
ELINE	3AEC	0543	0512
EMDV	0018	• • •	• • •
EMEM	0018	• • •	0015
EMESS	**262D**	**0AD6**	• • •
EMFN	0015	0015	0015
EMVN	0006	• • •	• • •
EMVR	001B	• • •	• • •
ENSA	0021	• • •	• • •
ENVE	0012	• • •	• • •
EOFOK	2947	0DEA	• • •
EPRM	- - -	- - -	4000
ERALP	3851	029A	026B
ERAS	**2B8D**	**1030**	**- - -**
ERASE	3836	027F	0250
ERS1	3631	1A67	0A3D
ERS2	3637	1A6D	0AF3
ERSYN	26EB	0B8E	0B8E
ERSZ	0039	• • •	0039
ES01	- - -	- - -	1D2F
ES02	- - -	- - -	1D53
ES03	- - -	- - -	1D5A
ES04	- - -	- - -	1D6D
ES05	- - -	- - -	1D76
ESCAP	3A09	0452	0421
ESCAX	3AAA	0501	04D0
ESCCRT	81BF	• • •	• • •
ESCD	**32C9**	**16FF**	• • •
ESCDG	32D1	1707	• • •
ESCG	32BE	16F4	• • •
ESCTB	36D8	008D	• • •
ESEC	2354	1CD7	1D22
ESIZ	0042	- - -	- - -
ESKF	000C	• • •	0018

LABEL	v6.78	v8.79	v9.80
ESYN	0009	• • •	• • •
ETYP	- - -	- - -	005A
EVEN	- - -	- - -	1DE6
EVFY	001B	• • •	• • •
EVOV	0027	• • •	• • •
EWRF	- - -	- - -	001E
EWSZ	0036	• • •	• • •
EXDHV	- - -	- - -	1B1D
EXE00	- - -	- - -	1030
EXE05	- - -	- - -	1A30
EXE10	- - -	- - -	1A61
EXE100	- - -	- - -	1ABA
EXE20	- - -	- - -	1A69
EXE30	- - -	- - -	1A6C
EXE40	- - -	- - -	1A76
EXE50	- - -	- - -	1A8A
EXE60	- - -	- - -	1A97
EXE70	- - -	- - -	1AA3
EXE80	- - -	- - -	1AA8
EXE90	- - -	- - -	1AB8
EXH	- - -	- - -	1B1B
EXTBF	81D6	• • •	• • •
EXTIN	0001	• • •	• • •
EXTOT	0007	• • •	• • •
F1	25EF	0A98	• • •
F2	25F2	0A9B	• • •
F3	25F9	0AA2	• • •
F4	2607	0AB0	• • •
F5	- - -	- - -	0AB4
FATR	**80F8**	• • •	• • •
FAUX	**810F**	• • •	• • •
FBLK	**8115**	• • •	• • •
FBUF	**8117**	• • •	• • •
FCS	**25EC**	**0A95**	• • •
FCSEM	**262A**	**0AD3**	• • •
FCSEX	**2622**	**0ACB**	• • •
FCSFL	**81E1**	• • •	• • •
FCSORG	25B7	- - -	- - -
FCSOUT	**3379**	**17AF**	• • •
FCSX	3301	1737	• • •
FDBK	**810D**	• • •	• • •
FDEN	**810E**	• • •	• • •
FDH	- - -	- - -	1B0F
FDRV	**8114**	• • •	• • •
FEED	3B3D	0594	055B
FERS1	2648	0AF1	• • •
FERS2	265A	0B03	• • •
FFCN	8113	• • •	• • •
FG0	226C	1C12	- - -
FG1	226E	1C14	- - -
FG2	2272	1C18	- - -
FG3	227F	1C25	- - -
FG4	2286	1C2C	- - -
FHAN	**8111**	• • •	• • •
FILL	00FF	• • •	- - -
FLAD	**8108**	• • •	• • •
FLBC	**8107**	• • •	• • •
FNAM	**80F9**	• • •	• • •

LABEL	v6.78	v8.79	v9.80
FNEW	0001	• • •	• • •
FPB	**807F**	• • •	• • •
FPBE	811D	• • •	• • •
FPBP	80F3	• • •	• • •
FPROM	0080	- - -	- - -
FPTR	**811B**	• • •	• • •
FREE	3A0A	0453	0422
FREEX	3A0A	0453	0422
FSAD	**810A**	• • •	• • •
FSBK	**8103**	• • •	• • •
FSIZ	**8105**	• • •	• • •
FTYP	**80FF**	• • •	• • •
FULL	3A4E	04A5	0474
FVER	**8102**	• • •	• • •
FXBC	**8119**	• • •	• • •
G01	3504	193A	• • •
GAR1	3258	168E	• • •
GAR2	3272	16A8	• • •
GAREC	**3257**	**168D**	• • •
GB1	3224	165A	• • •
GCMA	3488	18BE	• • •
GCTRK	256A	1EEB	- - -
GCUCNT	2570	1EF1	- - -
GDATAM	24C8	1E4B	- - -
GDRET	271E	0BC1	• • •
GETBC	37FC	024A	0226
GETBC0	- - -	- - -	0222
GETTO	2C0C	1042	• • •
GH1	22DB	1C5E	- - -
GH2	22E2	1C65	- - -
GH3	22E9	1C6C	- - -
GH4	22F3	1C76	- - -
GM01	2CCD	1103	• • •
GM02	2CD0	1106	• • •
GM03	2CBD	10F3	• • •
GMPRM	2CA4	10DA	• • •
GN1D	34E7	191D	• • •
GN1Z	34E4	191A	• • •
GN2D	34F9	192F	• • •
GN2Z	34F6	192C	• • •
GNDE	2D86	11BC	• • •
GNDE0	- - -	- - -	013E
GODBK	2FB5	13EB	• • •
GTBYT	**322C**	**1662**	• • •
GXOUT	2564	1EE5	- - -
HALF	3A4C	04A3	0472
HANADD	- - -	- - -	80E3
HANER	35FC	1A32	1E5D
HDCNT	- - -	- - -	6F85
HDVCT	368E	0043	1AEB
HER1	22A5	1C4B	- - -
HERR	- - -	- - -	1DDE
HEX	81B8	• • •	• • •
HOLD	- - -	- - -	002A
HOME	386E	02B7	0288
HOMEX	- - -	- - -	1CCB
HRTR	001E	• • •	- - -
IDEV	368B	0040	0A88

LABEL	v6.78	v8.79	v9.80
IDM	0055	• • •	- - -
INCXY	3D2C	078A	072C
INCXY0	- - -	- - -	072A
INI00	2721	0BC4	• • •
INI01	2761	0C04	• • •
INI02	2766	0C09	• • •
INI03	2768	0C0B	• • •
INIBAS	37BD	01EB	01E5
INITAD	0003	• • •	• • •
INPCRT	**81C5**	• • •	• • •
INPFL	81E3	• • •	• • •
INPTB	3728	00DD	• • •
INSEQO	**30E7**	**151D**	• • •
INVEC	3BC4	061B	05BA
INVY	3BE5	061B	05DE
INVY0	- - -	- - -	05DC
INVY1	- - -	- - -	05E2
IRBK	318E	15C4	• • •
IRBKI	3205	163B	• • •
IROLL	38C1	030A	02D9
IS1	2240	1BE6	- - -
IS2	224D	1BF3	- - -
IS3	2265	1C0B	- - -
IS4	2267	1C0D	- - -
ISEC	2235	1BE5	1CE5
ISERL	3974	03BD	038C
ISERX	3991	03DA	03A9
IUNT	368D	0042	0A8A
IVC	2604	0AAD	- - -
IWBK	318B	15C1	• • •
IWBKI	3202	1638	• • •
JMPD	2F01	1337	• • •
JMPHL	3FDD	0A3B	0F63
JUMP	**81E7**	• • •	• • •
KAP0	- - -	005C	• • •
KAP1	- - -	0079	• • •
KAP2	- - -	003B	• • •
KAP3	- - -	009F	• • •
KAP4	- - -	0021	• • •
KAP5	- - -	000C	• • •
KBCHA	**81FE**	• • •	• • •
KBDFL	**81DF**	• • •	• • •
KBR1	- - -	- - -	0378
KBRDY	**- - -**	**- - -**	**81FF**
KBREP	394F	0398	0367
KCHAR	**003E**	• • •	• • •
KEDEL	001D	• • •	• • •
KERDY	001E	• • •	• • •
KEYBD	3EB3	0911	• • •
KEYCO	002B	• • •	• • •
KEYOT	3A53	04AA	0479
KEYOT1	- - -	- - -	047D
KEYTST	- - -	- - -	0024
KEYTEST	**0024**	• • •	• • •
KTAB	3FDE	0A3C	08B8
L001	2D94	11CA	• • •
L002	2D82	11D8	• • •
L005	2DC9	11FF	• • •

LABEL	v6.78	v8.79	v9.80
L006	2E16	124C	• • •
L007	2E30	1266	• • •
L008	2E3D	1273	• • •
L009	2E40	1276	• • •
L010	2E4A	1280	• • •
L011	2E5C	1292	• • •
L012	2E75	12AB	• • •
L013	2E82	12B8	• • •
L014	2F14	134A	• • •
L017	2F60	1396	• • •
L023	2E7E	12B4	• • •
L025	2FAA	13E0	• • •
L1	32E9	171F	• • •
L106	2E15	124B	• • •
L2	32F7	172D	• • •
L3	332A	1760	• • •
L4	3334	176A	• • •
L5	3349	177F	• • •
LADSB	2F03	1339	• • •
LBYT	339B	17D1	• • •
LER1	294E	0DF1	• • •
LER2	2952	0DF5	• • •
LET	346A	18A0	• • •
LF	38BD	0306	02D5
LHXD	33A4	17DA	• • •
LINBF	**8046**	• • •	• • •
LINE	3B45	059C	0563
LINE1	- - -	- - -	0565
LKC	81E4	• • •	• • •
LL1	3E41	089F	0841
LL10	3EA4	0902	08A4
LL2	3E4D	08AB	084D
LL3	3E59	08B7	0859
LL4	3E65	08C3	0865
LL5	3E66	08C4	0866
LL6	3E7D	08DB	087D
LL7	3E80	08DE	0880
LL8	3E8C	08EA	088C
LL9	3E8D	08EB	088D
LLDA	28CD	0D70	• • •
LN5X	3897	02E0	02B1
LN5Y	389E	02E7	02B8
LO	**3392**	**17C8**	• • •
LOA00	2869	0D0C	• • •
LOCAL	3A4F	04A6	0475
LODG	3472	18A8	18A8
LOFL	**81F9**	• • •	• • •
LOL1	28C9	0D6C	• • •
LOL3	28E6	0D89	• • •
LOL4	28EA	0D8D	• • •
LOL5	291F	0DC2	• • •
LOL6	2936	0DD9	• • •
LOL7	293D	0DE0	• • •
LOLDA	288F	0D32	• • •
LOTO10	- - -	- - -	0138
LOTDUP	- - -	- - -	0130
LPJMP0	- - -	- - -	4803
LPJMP1	- - -	- - -	4806
LPJMP2	- - -	- - -	4809
LPJMP3	- - -	- - -	480C
LPJMP4	- - -	- - -	480F
LPJMP5	- - -	- - -	4812
LPJMP7	- - -	- - -	4818
LS1	3B93	05EA	0589
LS2	3B94	05EB	058A
LS3	3B9F	05F6	0595
LS4	3BA0	05F7	0596
LS5	3BAB	0602	05A1
LS6	3BAC	0603	05A2
LS7	3BBA	0611	05B0
LS8	3BBB	0612	05B1
LTIM	0B5A	• • •	• • •
LTNOR	347E	18B4	• • •
LTPEN	3B0A	0561	0530
LTYP	2D5A	1190	• • •
M1	356A	19A0	• • •
M2	357A	19B0	• • •
M8002	- - -	014A	• • •
MASK	0008	• • •	• • •
MCH01	2C2A	1060	• • •
MCH02	2C41	1077	1074
MCHNK	2C1A	1050	• • •
MDBLK	811E	• • •	• • •
MDDX10	- - -	- - -	1B3F
MDDX11	- - -	- - -	1B93
MDDX12	- - -	- - -	1BA3
MDH	- - -	- - -	1AF7
MFIOA	0000	• • •	• • •
MMEMO	- - -	018E	0175
MODE	0006	• • •	• • •
MOVDH	**343B**	**1871**	• • •
MOVHD	**3444**	**187A**	• • •
MOVXY	3D48	07A6	0748
MS1	2CF7	112D	• • •
MS150	81FD	• • •	• • •
MS2	2D04	113A	• • •
MS3	2D33	1169	• • •
MS4	2D42	1178	• • •
MST01	3496	18CC	• • •
MST02	34A3	18D9	• • •
MST03	34AB	18E1	• • •
MST04	34AD	18E3	• • •
MSTR	3495	18CB	• • •
MULHD	**3562**	**1998**	• • •
NEGH	**3524**	**195A**	• • •
NIBL	**33B3**	**17E9**	• • •
NKC	81E5	• • •	• • •
NOCHA	0040	• • •	• • •
NOLIN	0020	• • •	• • •
NOROL	3864	02AD	027E
NOTH	3525	195B	• • •
NROLL	3DA3	0801	07A3
NROLL0	- - -	- - -	07A2
NSEC	000A	• • •	- - -
NTRK	0029	• • •	- - -
NTYP	2D53	1189	• • •
OBC	80E3	• • •	8044
OCODE	80F5	• • •	• • •
ODDFL	81EE	• • •	• • •
OFFPB	26E5	0B88	011D
OPDIR	2D60	1196	• • •
OPEN	**2DAB**	**11E1**	• • •
OPENX	2C86	10BC	• • •
OPENY	2C89	10BF	• • •
OPOX	2C00	1036	• • •
OPX01	2CA2	10DB	10D8
ORAM	80F0	• • •	• • •
ORCHA	3D21	077F	0721
ORHD	352C	1962	1962
OSEC	80ED	• • •	• • •
OSERL	3A61	04B8	0487
OSTR	**33F4**	**182A**	• • •
OSTR1	3404	183A	• • •
OSTR2	340A	1840	• • •
OSTR3	340C	1842	• • •
OSTR4	3410	1846	• • •
OSTR5	3418	184E	• • •
OUTBL	3718	00CD	• • •
OUTCRT	**81C2**	• • •	• • •
OUTFL	**81F8**	• • •	• • •
OUTHL	81FB	• • •	• • •
OVERS	80F6	• • •	• • •
P2SNUM	34D2	1908	• • •
PAGE	3A86	04DD	04AC
PAGE1	- - -	- - -	04AF
PARTMP	- - -	- - -	81B5
PP1	3237	166D	• • •
PBINX	3C05	065C	05FE
PBINX1	- - -	- - -	060A
PBINY	3C8B	06E2	0684
PBINY1	- - -	- - -	0690
PBYT	34CD	1903	• • •
PCFSP	3087	14BD	• • •
PCOLN	34B8	18EE	• • •
PCRAD	81D8	• • •	• • •
PDV	**2FDE**	**1414**	• • •
PDV01	2FF3	1429	• • •
PDV02	3002	1438	• • •
PDV03	300F	1445	• • •
PDV04	3016	144C	• • •
PDV05	3019	144F	• • •
PDV06	3026	145C	145A
PDV07	3055	148B	148D
PDV08	3062	1498	• • •
PFS01	30A5	14DB	• • •
PFS02	30AB	14E1	• • •
PFS03	30BB	14F1	• • •
PFS04	30C1	14F7	• • •
PFSPC	**3077**	**14AD**	• • •
PLINC	3C7C	06D3	0675
PLOFL	81DA	• • •	• • •
PLOKB	3DE9	0847	07E9
PLOTX	3CB2	0709	06AB
PLOTX1	- - -	- - -	06BD

LABEL	v6.78	v8.79	v9.80
PLPTY	3BFD	0654	05F6
PLTAB	3D6F	07CD	076F
PNFSP	**306E**	**14A4**	• • •
PNUM	34D8	190E	• • •
POTON	3D99	07F7	0799
POUND	2511	1E8E	- - -
PPFSP	**3074**	**14AA**	• • •
PRCFPB	- - -	- - -	81AF
PRDEV	2CD4	110A	• • •
PRINT	3A96	04ED	04BC
PRM1	0080	- - -	- - -
PRM2	0081	- - -	- - -
PRM3	0082	- - -	- - -
PRM4	0084	- - -	- - -
PRM5	0085	- - -	- - -
PRM6	0086	- - -	- - -
PRMPT	3382	17B8	17BB
PROCES	398C	03D5	03A4
PROLL	0007	• • •	• • •
PSBYT	34CA	1900	• • •
PSFSP	**3068**	**149E**	• • •
PSNUM	34D5	190B	• • •
PSPAC	34B3	18E9	• • •
PSSTR	34BD	18F3	• • •
PSTAT	81DB	• • •	• • •
PSTR	34C0	18F6	• • •
PTBYT	**324A**	**1680**	• • •
PTREC	**3285**	**16BB**	• • •
PTYP	**2D57**	**118D**	• • •
PUP	81B7	• • •	• • •
PUTEZ	3DE6	0844	07E6
PUTEZ0	- - -	- - -	07E4
PUTXD	3C9B	06F2	0694
PUTXD1	- - -	- - -	069E
PUTXZ	3DC6	0824	07C6
PUTYD	3CEB	0749	06EB
PUTYD0	- - -	- - -	06E5
PUTYD1	- - -	- - -	06F5
PUTYD2	- - -	- - -	06E6
PUTYD3	- - -	- - -	06E8
PUTYZ	3DCD	082B	07CD
PUTZZ	3DAD	080B	07AD
PVREC	**327B**	**16B1**	• • •
PWRUP	37B1	01DF	018E
PWRUP1	- - -	- - -	01DF
PXYCH	3D13	0771	0713
PXYCH0	- - -	- - -	0712
PXZER	3BF3	064A	05EC
Q01	3EBS	0916	• • •
Q02	3EBC	091A	• • •
Q03	3EC9	0927	0929
Q04	3ECB	0929	092B
Q05	3EE7	0945	0947
Q06	3EEF	094D	094F
Q07	3EF7	0955	0957
Q08	3F07	0965	0967
Q09	3F2C	098A	098C
Q10	3F3F	099D	099F

LABEL	v6.78	v8.79	v9.80
Q11	3F43	09A1	09A3
Q12	3F54	09B2	09B4
Q13	3F81	09DF	09E1
Q14	3F85	09E3	09E5
Q15	3F9E	09FC	09FE
Q16	3FAC	0A0A	0A0C
Q20	3FC2	0A20	0A22
Q21	3FCB	0A29	0A2B
QLDA	2AAB	0F4E	0F4B
QTYP	2AAE	0F51	0F4E
RACM	- - -	- - -	00C4
RAMJMP	- - -	- - -	6F80
RATE	- - -	- - -	0002
RATEA	0016	• • •	• • •
RATEB	002F	• • •	• • •
RBLK	**3182**	**15B8**	• • •
RBLKI	31F9	162F	• • •
RBYTE	246A	1DED	- - -
RBYTEC	2474	1DF7	- - -
RCMD	- - -	- - -	0088
RCOMD	0017	• • •	• • •
RD	2EFB	1331	• • •
RD00	2411	1D94	1DC7
RD01	241C	1D9F	1DF3
RD02	2429	1DAC	1DFA
RD03	2435	1DB8	1E04
RD04	2438	1DBB	1E0D
RD05	- - -	- - -	1E11
RDAT	- - -	- - -	0023
REA00	26F1	0B94	• • •
READ	2EA3	12D9	• • •
READY	**81FF**	• • •	**- - -**
REN00	2AC1	0F64	• • •
RERR	2453	1DD6	- - -
RESET	**26A5**	**0B48**	• • •
RF1	2880	0D23	• • •
RF2	2883	0D26	• • •
RFLG	80E1	• • •	- - -
RGAPS	0003	• • •	- - -
RINT	- - -	- - -	0029
RNCOM	- - -	- - -	01A0
RNFIL	- - -	- - -	0ABB
ROLDA	0066	0076	• • •
ROLFL	81DC	• • •	• • •
ROLL	3A85	04DC	04AB
ROLLN	81CD	• • •	• • •
ROT	3EFE	095C	095E
RRTR	000A	• • •	- - -
RS01	26AB	0B4E	0B4B
RS02	26CE	0B71	• • •
RS03	- - -	- - -	0B78
RSCM	- - -	- - -	0004
RSEC	- - -	- - -	0022
RST1J	81C8	• • •	• • •
RSTBF	0002	• • •	• • •
RTST	33D5	180B	• • •
RTST2	33D8	180E	• • •
RUN00	2956	0DF9	• • •

LABEL	v6.78	v8.79	v9.80
RUN01	296D	0E10	• • •
RUN02	297C	0E1F	• • •
RUNC0	- - -	- - -	1A0E
RUNC10	- - -	- - -	1A1D
RUNC20	- - -	- - -	1A29
RWB1	31EC	1622	• • •
RWB2	31EF	1625	• • •
RWBCM	318F	15C5	• • •
RWE	2EC5	12FB	• • •
RWICM	3206	163C	• • •
RWSEQI	**30C6**	**14FC**	• • •
RXBUF	0000	• • •	• • •
RXSER	0020	• • •	• • •
S1OUT	**33C3**	**17F9**	• • •
SAV00	2833	0CD6	• • •
SAVE	3FD0	0A2E	0A30
SBC	8042	• • •	- - -
SBCHA	39BC	0405	03D4
SCMN	3554	198A	• • •
SEC	80EE	• • •	• • •
SEC15	81D7	• • •	• • •
SEEK	2222	1BCE	1C8B
SEEKF	22AA	1C50	- - -
SET00	- - -	- - -	0A76
SETC9	- - -	- - -	0144
SETEXE	- - -	- - -	0A71
SFR	2F96	13CC	• • •
SHIFF	3B85	05DC	057B
SHLHD	353A	1970	• • •
SHRHD	3544	197A	• • •
SIZBIT	- - -	- - -	80E2
SL1	353E	1974	• • •
SLDSD	- - -	- - -	1C50
SOK	0000	• • •	• • •
SP64C	39B4	03FD	03CC
SPNOR	**3460**	**1896**	• • •
SR1	3548	197E	• • •
SRTR	001E	• • •	0004
SSIDA	0008	• • •	• • •
SSOBE	0010	• • •	• • •
STACK	**8042**	• • •	**8044**
START	0000	• • •	• • •
STARTIT	006E	• • •	• • •
STARX	376C	01BA	• • •
STAT5	0003	• • •	• • •
STEP1	- - -	1BD3	- - -
STEPCD	0027	• • •	- - -
STEPS	2525	1E9D	- - -
STIM	003F	• • •	• • •
SPOPIT	006A	• • •	• • •
STOR0	- - -	- - -	0400
STOR1	39EB	0434	0403
STOR2	39F1	043A	0409
STP1	2529	1E9E	- - -
STP2	2536	1EB0	- - -
STP3	2558	1EB7	- - -
STP4	- - -	1ED8	- - -
STPIN	253D	1EBA	- - -

LABEL	v6.78	v8.79	v9.80
STPITN	- - -	1EC0	- - -
STPITO	- - -	1ECE	- - -
STPOUT	254D	1EC8	- - -
STPTIM	0014	• • •	- - -
STPWAT	255E	1EDF	- - -
STRTMP	- - -	- - -	81B3
STWO	2C67	109D	• • •
STX	- - -	- - -	1CAA
STYP	2D5D	1193	• • •
SUBHD	**3459**	**188F**	• • •
SUBU	214D	1AF2	1B97
SULD	2986	0E29	• • •
SVCHA	39DC	0425	03F4
SVCRS	3873	02BC	028D
SVCRS1	- - -	- - -	0290
SVCRS2	- - -	- - -	0293
SXO	2502	1E7F	- - -
SXO1	250C	1E89	- - -
SYSORG	211C	0120	• • •
TAB	38B1	02FA	02C9
TAU	0005	• • •	- - -
TBADDR	368B	- - -	- - -
TBC	80EB	• • •	• • •
TBL00	36A8	005D	• • •
TBL24	36C8	007D	• • •
TBLK	80E7	• • •	• • •
TDRV	80E6	• • •	• • •
TEMP0	81F2	• • •	• • •
TEMP1	81F3	• • •	• • •
TEMP2	81F4	• • •	• • •
TEMP3	81F5	• • •	• • •
TEMP4	81F6	• • •	• • •
TEMP5	81F7	• • •	• • •
TEMPHL	80DE	• • •	• • •
TESHI	39CE	0417	03E6
TEST	3A09	0452	0421
TESTB	3A45	049C	046B
TESTB0	- - -	- - -	0469
TESTB1	- - -	- - -	046C
TESTC	3A2D	0476	0445
TESTC1	- - -	- - -	044B
TESTX	3831	027A	024B
TFCN	80E5	• • •	• • •
TFILE	0002	• • •	• • •
TFREE	0001	• • •	• • •
THRUFL	81DE	• • •	• • •
TIM3X	3EA6	0904	08A6
TIM4X	3AD1	0528	04F7
TIM4X1	- - -	- - -	0501
TIM4Y	3AE0	0537	0506
TIM4Z	3AE6	053D	050C
TIME1	0009	• • •	• • •
TIME2	000A	• • •	• • •
TIME3	000B	• • •	• • •
TIME4	000C	• • •	• • •
TIME5	000D	• • •	• • •
TIMX1	0000	• • •	• • •
TIMX2	0008	• • •	• • •
TIMX3	0018	• • •	• • •
TIMX4	0030	• • •	• • •
TIMX5	0038	• • •	• • •
TMEM	80E9	• • •	• • •
TMODE	39FE	0447	0416
TMP1	81AB	• • •	• • •
TOFF	2154	1AF9	1B9E
TPROG	001B	• • •	• • •
TPROT	0001	• • •	• • •
TRAM	80DE	• • •	• • •
TRK	80EF	• • •	• • •
TWOUT	3B25	057C	0543
TXBUF	0006	• • •	• • •
TXCONT	3B1D	0573	053A
TXOUT	3B30	0587	054E
TXOUT1	- - -	- - -	0550
TXPEN	3B5D	05B4	- - -
TXSER	0028	• • •	• • •
UPDATE	3805	0253	022F
UPTIM	0640	• • •	- - -
VCRAD	81CB	• • •	• • •
VCRSY	3A1F	0468	0437
VE00	2305	1C88	1CF0
VE01	2322	1CA5	1D03
VE02	232A	1CAD	1D09
VE03	2336	1CB9	1D0F
VE04	233D	1CC0	1D1A
VECTO	3E2D	088B	082D
VECTO1	- - -	- - -	082F
VECTY	3BBE	0615	05B4
VERR	230B	1C8E	1CFC
VFILL	**81CE**	• • •	• • •
VHLAD	**81D2**	• • •	• • •
VISIB	3A0A	0453	0422
VRTR	0002	• • •	- - -
VTP	24DD	1E60	- - -
VTP1	24EC	- - -	- - -
VTP2	24F4	1E71	- - -
WATL	**3429**	**185F**	• • •
WATS	**341C**	**1852**	• • •
WB1	245F	1DE2	- - -
WBLK	**317F**	**15B5**	• • •
WBLKI	31F6	162C	• • •
WBYTE	245E	1DE1	- - -
WCMD	- - -	- - -	0024
WCMN	- - -	- - -	00A8
WDAT	- - -	- - -	0027
WDSL	- - -	- - -	0028
WF2	284F	0CF2	• • •
WIG1	22BA	1C53	- - -
WIG2	22C4	- - -	- - -
WIG3	22D2	- - -	- - -
WL1	342A	1860	• • •
WL2	342D	1863	• • •
WR	2EF8	132E	• • •
WR00	23A6	1D29	1D7F
WR01	23BF	1D42	1DA1
WR02	23C4	1D47	1DA7
WR03	23D0	1D53	1DAB
WR04	23D4	1D57	1DB9
WR05	23E6	1D69	1DC4
WR06	23F0	1D73	- - -
WRDIR	2F75	13AB	• • •
WRI00	26EE	0B91	• • •
WRITE	**2ECC**	**1302**	• • •
WRTR	0004	• • •	- - -
WS1	341F	1855	• • •
WSEC	- - -	- - -	0026
WTRK	- - -	- - -	0025
WXYZ	2573	1EF4	- - -
X80	7000	• • •	• • •
XDATA	81EC	• • •	• • •
XFBLK	81A1	• • •	• • •
XFBUF	81A3	• • •	• • •
XFDRV	81A0	• • •	• • •
XFER	219A	1B3F	- - -
XFFCN	819F	• • •	• • •
XFHAN	819D	• • •	• • •
XFXBC	81A5	• • •	• • •
XINTR	0010	• • •	• • •
XORHD	3533	1969	• • •
XOUT0	81AF	• • •	- - -
XOUT1	81B0	• • •	- - -
XTWO	81EA	• • •	• • •
XYMIT	3B13	0569	0530
XYTAB	3D67	07C5	0767
XZERO	81EF	• • •	• • •
YDATA	81ED	• • •	• • •
YTWO	81EB	• • •	• • •
YZERO	81F0	• • •	• • •
ZERFL	39A9	03F2	03C1
ZFATR	8083	• • •	• • •
ZFAUX	809A	• • •	• • •
ZFBLK	80A0	• • •	• • •
ZFBUF	80A2	• • •	• • •
ZFDBK	8098	• • •	• • •
ZFDEN	8099	• • •	• • •
ZFDRV	809F	• • •	• • •
ZFFCN	809E	• • •	• • •
ZFHAN	809C	• • •	• • •
ZFLAD	8093	• • •	• • •
ZFLBC	8092	• • •	• • •
ZFNAM	8084	• • •	• • •
ZFPB	8082	• • •	• • •
ZFPBE	80A8	• • •	• • •
ZFPTR	80A6	• • •	• • •
ZFSAD	8095	• • •	• • •
ZFSBK	808E	• • •	• • •
ZFSIZ	8090	• • •	• • •
ZFTYP	808A	• • •	• • •
ZFVER	808D	• • •	• • •
ZFXBC	80A4	• • •	• • •
ZH	355E	1994	• • •
ZPTR	30D9	150F	• • •
ZRAM	8082	• • •	• • •
ZZZZZZ	3FFA	- - -	- - -

PRODUCT REVIEWS

Joseph Norris

IDA, software debugger
COLORWORD, word processor, screen editor

These two software packages have been long overdue for review by a Compucolor publication. Part of the delay has been caused by revisional activity by the authors, but the real delay has been the lack of adequate marketing efforts for US buyers. Colorcue hopes both these programs will receive the attention they deserve as the latest products directed to the Compucolor/Intecolor owner.

IDA, BY BILL GREENE. $49.50
(INTELLIGENT COMPUTER SYSTEMS)

IDA is a software debugging program by Bill Greene. Bill is also the author of Super Monitor, Super Monitor Plus, Compucalc, and an edition of FORTH, as well a a few games. He has been a frequent contributor to Colorcue since its very beginning.

It's difficult to give a straightforward review of IDA, because after using it daily for several months, I can't think of anything but superlatives. IDA is the topic of the tutorial in this issue by W.S. Whilly, who begins a fascinating exploration of IDA's possibilities. He has only scratched the surface. To be straightforward, it is my opinion that should I lose all my other software, I would hope to keep IDA. It is brilliantly written. It is a flawless performer. It is the greatest timesaver imaginable. IDA is usually the first software I call upon at power-up, and the last I use before dragging my mortal remains to bed. Had I owned it three years ago, I might actually know something about the CCII at this point. It is certain I would have spent much less time head scratching and reconstructing lost disks.

IDA is an (I)nterpreter, (D)isassembler, and (A)ssembler, monitor, calculator and debugger all wrapped within 8K bytes of magic. It is available for loading at 4000H, 8200H, A000H, and E000H. IDA.REL is also available for macroassembler users for loading at any address in user memory. It will operate with v6.78, v8.79 and v9.80 software, and there is work in progress to make it available to 8000 computer users as well.

IDA does all of the following, and then some: a) makes an ASCII dump of memory, b) sets printer Baud rate, c) sets checkpoints, d) disassembles memory, e) prints SRC files, f) fills specified memory with a constant, g) runs assembly programs, h) makes a hex dump of memory, i) interprets assembly programs with a register dump, j) allows addiion of user-defined jump table parameters, k) compares designated memory locations, l) prints any screen display to printer, including displays you type there (in simulated CRT mode) for clarification of screen data, m) moves designated memory, n) makes decimal dump of memory, o) sets origins for pseudo-assembly of mnemonics from keyboard directly to memory, p) peeks, with an option to poke, at any memory location, q) exits to FCS, s) searches memory for byte string, with option to disassemble, replace with designated string, or present a peek/poke mode, t) types keyboard lines to printer, v) displays directory with usual display plus file length in hex and decimal, x) executes any FCS command without exiting IDA, z) sets lines/page and blank line format for printouts.

While IDA has more capability that any previously-issued program of its type, it isn't what it does that is so impressive; it's how it does it! This is difficult to describe in text (you'll do better if you follow W.S. Whilly's article). The large figure on the next page is a "snapshot" of an IDA screen following a disassembly from address 8200. Following the IDA prompt is my instruction to disassemble 10 (hex) program lines beginning at 8200. IDA displays, in order of column, the memory address, a grouping of hex digits associated with that address, a translation of those digits to mnemonics, and, in the last column, an interpretation of the hex digits as though they were not program code. Notice the lables "IPCRT", "ICRT1", and "KBDFL" in the last column. IDA has recognized these addresses from the disassembly and printed the system names for them —for my convenience. This is a colorful display column, each number being colorcoded to establish its ranking in the 256 possible characters available in the CCII. You can readily identify a printable ASCII character, a control code, or a special character.

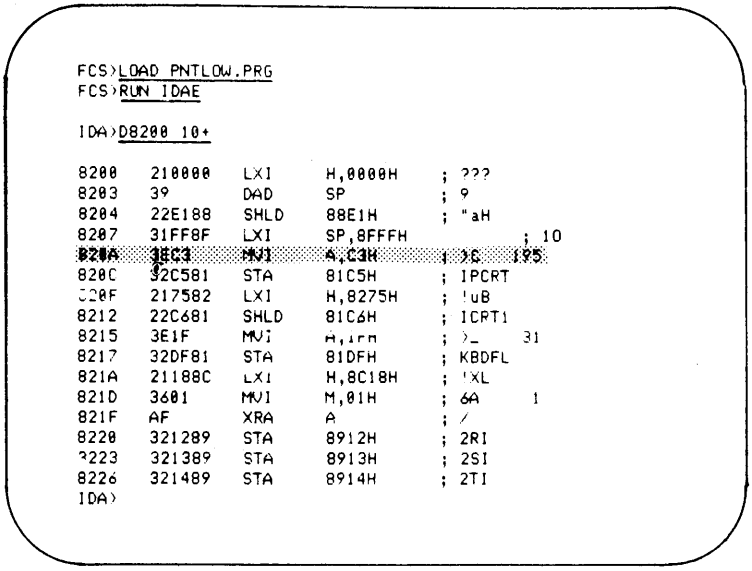
Is this particular instance, when the display reached the line of address 8226, I pressed the [UP/ARROW]. A blue bar cursor appeared on the screen. I held the [UP/ARROW] key until the bar cursor moved up to the line it is shown on in the figure, address 820A. IDA now lets me do a number of things at this address: I may use a Peek/Poke option to enter new hex digits at this address, I may set this address as an origin for assembly and actually type in mnemonics at the keyboard, which will be instantly assembled and inserted, beginning at address 820A.

The ease with which this kind of procedure is carried out is not adequately expressed with words. Similarly, all of IDA's functions act with "forethought" which is to say — the program "knows" what you are likely to want next, and presents itself for your service accordingly. It is clearly the work of a master.

The simple feature of making FCS commands available from IDA without a program exit is a remarkable time-saver. No perpetual reentries to IDA, no lost jump vectors from ESC USER. IDA reveals, with ease, any mystery the disk drives hold. All sectors are there to be read with a simple command. All sectors may be rewritten with a simple command.

Calculate with any combination of binary, hex or decimal numbers, using arithmetic operators, or the MOD, AND, OR and EXOR functions. Save yourself from tedious calculations with IDA's display of disk file size. Convert an LDA to a PRG in seconds. Repair bad Basic program lines in a jiffy. Correct a bad disk directory. Do anything you ever wanted to do but couldn't.

Selectively run an assembly language program, one instruction at a time, and



```
FCS>LOAD PNTLOW.PRG
FCS>RUN IDAE

IDA>D8200 10+

8200   210000    LXI     H,0000H     ; ???
8203   39        DAD     SP          ; 9
8204   22E188    SHLD    88E1H       ; "aH
8207   31FF8F    LXI     SP,8FFFH            ; 10
820A   3EC3      MVI     A,C3H       ; >C    195
820C   32C581    STA     81C5H       ; IPCRT
820F   217582    LXI     H,8275H     ; !uB
8212   22C681    SHLD    81C6H       ; ICRT1
8215   3E1F      MVI     A,1FH       ; >_     31
8217   32DF81    STA     81DFH       ; KBDFL
821A   21188C    LXI     H,8C18H     ; !XL
821D   3601      MVI     M,01H       ; 6A      1
821F   AF        XRA     A           ; /
8220   321289    STA     8912H       ; 2RI
8223   321389    STA     8913H       ; 2SI
8226   321489    STA     8914H       ; 2TI
IDA>
```

A SAMPLE IDA SCREEN

watch the registers change...or run sixteen lines at a time, or stop at preselected places of your choosing and examine the registers and memory to see what happened. Look at the stack and watch it change.

All these things are not only routine with IDA, but so logically executed and displayed, it's difficult to remain ignorant about anything for very long.

There must be a catch somewhere! There is. IDA's manual is painfully skimpy. Trying to find out how to make these provisions come to life reminds me of my early days with the Compucolor manual from ICS. The first IDA manual was a skeleton. Subsequent revisions have been improving, but you will be blind to IDA'S power without articles like Mr. Whilly's or a few daring sessions at your own keyboard, exploring.

There are plans at Colorcue to produce a manual for IDA worthy of its content. But, for goodness sake, don't let a poor manual stop you! Get IDA quick, and join us. IDA will be everywhere in the pages of Colorcue. It does for the entire computer what 'THE' Basic Editor and FASBAS have done for Basic. The price is a sinfully modest $49.95. Order from Intelligent Computer Systems in Huntsville, Alabama. (See their catalog, this issue.) If you want a version for 3651 or 8000, write to me at Colorcue. This is a program you'll cherish.

COLORWORD, BY CHRIS TEO. $50. (PROGRAM PACKAGE INSTALLERS)

CCII users have used Comp-U-Writer for years. It's a very respectable word processor at a very high price. One major drawback of Comp-U-Writer is its inability to process control and escape commands to take advantage of the various facilities of modern printers.

COLORWORD removes the price and printer support deficiencies of Comp-U-Writer in one $50 stroke. COLORWORD does not have Comp-U-Writer's "polish", and COLORWORD has its idiosyncracies, but it works, and if you court it appropriately it will serve you well.

The latest revision is v4.5 which contains a new sign-on display (I liked the old one better—less cluttered), keyboard buffering, and a capacity to reformat paragraphs. Current COLORWORD users might want to make the $15 investment to get these improvements if they were out-typing the software and were constantly losing lines of text from the screen.

COLORWORD offers the following, more or less standard, word processing facilities:

Line editing —delete character, delete line, delete all text, insert character, text, page markers and control characters.

Block editing —delete, move, copy, save, and print block.

File commands —save, load, initialize disk, print directory, delete file, rename file, change device.

Cursor control —up down, left right, down/up single or multiple lines, single page.

Printing —Screen preview of printout, print text, set printer parameters (including control and escape codes anywhere in text).

Special functions —string search with optional replace, operate with or without lower case character set (but print either to printer), HELP facility, typematic keys, compact file storage, generates SRC-type files, will process any SRC files, 21K or 27K text area, can be used with any CCII keyboard, available for loading from disk at 4000H.

COLORWORD's manual is thorough and clearly written (18 pages). The only difficulty with this program is learning to live with it's peculiarities. A double RETURN is required to establish a paragraph boundary. When editing (adding) words in the midst of a paragraph you will frequently encounter the "vanishing line" syndrome—entire lines of text disappear from the screen forever. They are in memory, and they print out, but they are invisable. There is a way out of this dilemma, faintly referred to in the manuals of earlier versions, more carefully annotated in v4.5, but one must be on one's toes to keep the demon away. COLORWORD is sometimes erratic in other ways, particularly if you have unusual margins set, or use tabs too much. I haven't found a problem I couldn't circumvent, but I have resented the time I had to spend courting the software when I had pressing work to do. I admit, I have frequently abandoned COLORWORD for Comp-U-Writer. At the same time, the price is right, and it works as advertised. If you've not been able to use a word processor because of a limited budget, COLORWORD is for you. It will be a good friend. COLORWORD works with v6.78, v8.79 and v9.80 computers. It is available from Program Package Installers in Australia. (See their advertisement in this issue.) Don't let their address deter you. PPI is prompt and supports its products conscientiously.

This article has originated from my experiences with the compiler, FASBAS, following the notion that some of the subroutines contained in the Basic interpreter in ROM might be of value in Assembly Language programs. It might be useful to access subroutines for square root, COS, TAN, etc, or the random number generator. The procedure for achieving this is somewhat complicated, but it can be done.

The Basic subroutines all operate on floating point numbers, to handle a wide range of values with accuracy to 5 or 6 decimal places. I assume that Assembly Language programs are going to treat all numbers as integers, expressed as 8 bit or 16 bit binary values. So we must establish a convention, and I propose that any number which is to be operated on by a Basic subroutine must be loaded as a 16 bit value in the DE register pair. Similarly the result of the operation will appear as a 16 bit value in the DE register pair. To accommodate negative values, the most significant bit will be a sign indicator (1 for negative) and thus a range of integers from -32768 to + 32767 can be expressed. This is the normal convention for signed 16 bit binary, in which positive values are counted upward from 0 to 7FFFH, and negative values are counted downward (FFFFH = -1, FFFEH = -2, etc).

So far the strategy is simple and can be described in the following list of steps:

1) Load 16 bit signed value into DE register pair, 2) Call subroutine to convert to floating point, 3) Call required Basic subroutine, and 4) Call subroutine to convert back to 16 bit signed value in DE register pair.

The primary problem is that the value returned by the Basic subroutine will, in many cases, lie between 0 and 1, and will appear as a value of 0 when converted from floating point to binary integer. We can circumvent this by multiplying the floating point number by some scaling factor (such as 100 or 1000) before converting back to integer form, but this is not as straightforward as it seems.

The Basic subroutine for floating point multiplication is not wholly self-contained in ROM. In order to use it we must first call another subroutine which loads instructions into the memory area 8200H to 8298H. Therefore, your program must, at an early point, include (just once) the following instructions (in this and subsequent listings, the Basic subroutine address is given for v6.78, followed by v8.79 in brackets):

CALL 17DEH [372DH] ; Load 8200H to 8298H

Your program cannot itself be loaded at any address lower than 8299H, as it would be overwritten by these Basic instructions.

We need a clear understanding of the order in which Basic carries out a series of mathematical operations, and where it stores intermediate results. Basic uses the memory area 80DEH to 80E1H as a 4 byte accumulator for floating point numbers, and results of all operations are stored here. If the expression COS(X)*100 is to be evaluated, the sequence of operations is as follows:

1. Put the floating point value of X in the accumulator,
2. Call the COS subroutine (result in the accumulator),
3. PUSH the contents of the accumulator onto the stack,
4. Put the floating point value 100 in the accumulator,
5. POP 4 bytes [=COS(X)] from the stack to registers BCDE,
6. Call the multiplication subroutine (result in the accumulator).

This example could be extended to divide the result by Y, in which case Steps 7 and 10 would be very similar to Steps 3 and 6:

7. PUSH the contents of the accumulator onto the stack,
8. Put the floating point value of Y in the accumulator,
9. POP 4 bytes (=COS(X)*100) from the stack to registers BCDE,
10. Call the division subroutine (result in the accumulator).

Two points are worth noting. If X is to be divided by Y, then X must be in registers BCDE and Y must be in the accumulator (and similarly for X-Y). The second point is that all the registers are used during these operations, which may help to explain why Basic moves values from the accumulator to stack and thence to registers BCDE, to keep them safe during Steps 4 and 8. It should also warn you that if you want to retain the contents of any registers, you must PUSH them onto the stack before calling any Basic subroutines.

Using Basic Subroutines in Assembly Language Programming

Peter Hiner 11 Pennycroft Harpenden Herts, AL5 2PD ENGLAND

We are now ready to consider the full sequence for accessing a Basic subroutine from an Assembly Language program. I will use as an example a routine to evaluate the Basic expression INT(RND(1)*100)+1, which generates a random integer value between 1 and 100. I assume that you have previously carried out the instruction CALL 17DEH [372DH] to load Basic routines into the 8200H-8298H memory field. (See Listing 1.)

Note that the subroutine at 134FH [329EH] should always precede CALL 114AH [3099H], even when you are sure the result will be positive.

Listing 2. is a sample of simple subroutines which could be used in the example of Listing 1. where RND was used. For the arithmetic functions, remember that X must be in registers BCDE and Y must be in the accumulator. (See Listing 3. for arithmetic functions.)

A routine to evaluate 3 to the power of 5 is shown in Listing 4. This routine returns a value of 729, so there is no problem, but 3 raised to the 10th power would yield 59049. The hexadecimal value in registers DE would be

Listing 1.

```
LXI   D,1              ; Load 1 for RND(1)
CALL  1A5AH [393ah]    ; Convert 1 to floating point
                       ; - value in accumulator
CALL  16D0H [361FH]    ; RND subroutine generates
                       ; RND(1) in accumulator
CALL  1380H [32CFH]    ; Put RND(1) on stack
LXI   D,100            ; Load scaling factor = 100
CALL  1A5AH [393AH]    ; Convert to fp in accumulator
POP   B                ; Pop RND(1) from stack to BCDE
POP   D                ;
CALL  12A3H [31F2H]    ; Multiply RND(1) by 100
CALL  134FH [329EH]    ; Test for positive/negative result
CALL  114AH [3099H]    ; Convert result to integer in DE
MOV   A,E              ; D=0, E=0 to 99
INR   A                ; Now A=1 to 100
```

Listing 2.

Function..v6.78..v8.79

SQR	15F8H	3547H
RND	16D0H	361FH
LOG	1263H	31B2H
EXP	163FH	358EH
COS	1704H	3653H
SIN	170AH	3659H
TAN	176BH	36DAH
ATN	1780H	36CFH

E6A9H, which, according to our convention for signed 16 bit binary integers, we should evaluate as -6487. Clearly you can, in practice, treat results between 8000H and FFFFH as positive numbers in the range 32768 to 65535, provided that you know what range of result to expect from the Basic subroutines. This does not apply to values at entry to Basic subroutines, which will always evaluate FFFFH as -1, etc. If you try to evaluate an expression which gives a result outside the range -32768 to +65535, then you will find that the program jumps into a Basic error subroutine which gives you a CF ERROR message. Intermediate floating point results outside this range are acceptable provided that you divide by a scaling factor (or add/subtract a suitable value) to bring the result back into range before converting to a binary integer.

A final point to remember is that the conversion from floating point to integer values always gives as a result the lower of the two adjacent integers, and does not round the result to the nearest integer (e.g. 5.9999 becomes 5 rather than 6). You probably expected this, but you may be surprised to learn that -5.001 becomes -6, which is at least consistent if not entirely logical. □

Listing 3.

Function..v6.78..v8.79

X + Y	1167H	30B6H
X - Y	1164H	30B3H
X * Y	12A3H	31F2H
X / Y	12E5H	3234H
X ^ Y	1603H	3552H

Listing 4.

```
LXI   D,3
CALL  1A5AH [393AH]    ; Convert to floating point
CALL  1380H [32CFH]    ; Put 3 on stack
LXI   D,5
CALL  1A5AH [393AH]    ; Convert to floating point
POP   B                ; POP value 3 to BCDE
POP   D
CALL  1603H [3552H]    ; Subroutine for ^
CALL  134FH [329EH]    ; Test for positive/negative result
CALL  114AH [3099H]    ; Integer in registers DE
```

one dimensional cellular automata

David Suits 49 Karenlee Drive Rochester, NY 14618

By now almost everyone who has worked with computers has learned of the game of *LIFE*, in which the screen is conceptually divided into hundreds of squares (cells), and each cell is treated as containing a living organism (represented, say, by an asterisk) or else treated as dead or empty (represented by a blank)..On the Compucolor/Intecolor it would be appropriate to set up the screen as a two-dimensional array of cells, 64 wide and 32 high. Given an initial number of live cells, placed hither and yon on the screen (usually at the discretion of the user), it is interesting to watch what happens when certain rules of transition are applied to each cell in the array and the new result displayed. In the original *LIFE* there were three rules (or four, depending on how you expressed them) for determining when a live cell "dies", when an empty cell "gives birth" and when live cells stay alive. For example, if a live cell is touching less than two other live cells, the cell dies; when an empty cell has exactly three live cells as neighbors, it gives birth to a new live organism.

A program to run the game of *LIFE* would set up the initial two dimensional array, then check each cell, counting the number of its live neighboring cells and applying the appropriate transition rule. When the entire array has been checked, the program would display the results and then start the transition process over. This would continue 'ad libitum.' It is fascinating to watch cellular patterns emerge and dissolve, especially if the program is implemented in assembly language.

There is nothing sacred, however, about using any particular transition rules which specify when a cell dies or when it stays alive, etc. You can make up your own rules for experimentation. You can also expand upon the concept of a "live" cell: you could have different "species" of orgamisms which interact; each species might have slightly different transition rules.

The game of *LIFE* is an instance of "two dimensional cellular automata." Could there be one dimensional cellular automata? Sure. Instead of n lines of c cells each, you would deal with only 1 line of c cells, and the transition rules would have to deal with a cell's neighbors to the left and right. (A cell

would have no neighboring cells above it, because the line above would represent the previous generation; and it would have no neighboring cells below it, because that would be the next generation which would not yet exist.) In the March, 1984 issue of Scientific American, Brian Hayes discussed one dimensional cellular automata, —a sort of game of *LIFE* played on only one row at a time instead of a full screen. One interesting aspect of this is that each new generation can be displayed on each line of the screen, so that the original line is seen as "growing" from the top of the screen down (or, if you wish, from the bottom up.)

Implementation of a one dimension cellular automaton on the Compucolor or Intecolor computers is easy using plot graphics, where each plot block represents one cell ("alive" if the block is on, "dead" if not). Thus, a one dimensional cellular automaton could be 128 cells long, and you could display 128 lines or generations on your screen. (If you have a printer with dot addressable graphics, you can get even larger automata and print thousands of generations.)

Various transition rules for determining when a live cell dies, when a dead cell gives birth, etc., can be invented. Some transition rules will result in an uninteresting growth (or decay) of cells; others will generate unexpected, plant-like (or crystal-like) structures, often with surprising symmetries.

Listing 1 is a BASIC program which implements a one dimensional cellular automaton which begins at the top of the screen. Each new generation is displayed on the next line down. The transition rule used is simple: for each cell in the line, count the number of live cells two cells to the left and two cells to the right of it, and add 1 if the cell itself is alive. If the total is either 2 or 4, then the cell itself becomes (or remains) alive. Otherwise, it becomes (or remains) dead.

```
0 GOTO 63000 : REM  POKE IN NO-ECHO PATCH
10 REM  ************************************************
11 REM  *                                              *
12 REM  * CELL1D -- One dimensional cellular automata pro-  *
13 REM  *           gram.                              *
14 REM  *                                              *
15 REM  * D. B. Suits,  March, 16 a.l.                 *
16 REM  *                                              *
17 REM  * See Brian Hayes, "Computer Recreations" in    *
18 REM  * SCIENTIFIC AMERICAN, March, 1984              *
19 REM  *                                              *
20 REM  ************************************************
30 REM
40 RGT=127 : REM  Width of line (0--RGT).
50 T0P=127 : BOTT0M=0
60 KB=33278 : KF=33247 : REM  Addresses for keyboard reading.
80 ALIVE=-1 : DEAD= NOT (ALIVE)
90 COL0R=2 : REM  PLOT 6,COL0R for plotting live cells.
100 DIM LINE(RGT),TEMP(RGT)
110 REM
120 REM  ------------------------------------------------
130 REM  Define the transition function for each cell. In the
140 REM  present version, a cell becomes (or stays) alive if
150 REM  the total number of live cells 2 positions on either
160 REM  side of the present cell is either 2 or 4.
170 REM
180 REM  Count live cells in 2 spots left of cell X.
190 DEF FNL2(X)=-(LINE(X-1)=ALIVE)-(LINE(X-2)=ALIVE)
200 REM
210 REM  Count live cells in 2 spots right of cell X.
220 DEF FNR2(X)=-(LINE(X+1)=ALIVE)-(LINE(X+2)=ALIVE)
230 REM
240 GOTO 1000
250 REM
260 REM  This is the actual, called subroutine. It makes use
270 REM  of functions L2 and R2 above if possible. X is the
280 REM  present X index; NC is the Neighbor Count. Boundary
290 REM  checks must be included.
300 REM
310 IF X=0 THEN NC=FNR2(X) : GOTO 360
320 IF X=1 THEN NC=FNR2(X)-(LINE(0)=ALIVE) : GOTO 360
330 IF X=RGT-1 THEN NC=FNL2(X)-(LINE(RGT)=ALIVE) : GOTO 360
340 IF X=RGT THEN NC=FNL2(X) : GOTO 360
350 NC=FNL2(X)+FNR2(X)
360 REM
370 REM  The result is either a live cell or a dead cell,
380 REM  based upon the Neighbor Count.
390 REM
400 IF (NC=2) OR (NC=4) THEN TEMP(X)=ALIVE : GOTO 420
410 TEMP(X)=DEAD
420 RETURN
430 REM
440 REM  ------------------------------------------------
900 REM
910 REM  ================================================
920 REM
930 REM  Main program
940 REM
950 REM  ================================================
960 REM
1000 GOSUB 2000 : REM  Set up and instructions.
1010 GOSUB 3000 : REM  Get initial line.
1015 REM
1020 PLOT 3,64,0,6,COL0R : REM  Hide cursor; set color.
1030 PLOT 2 : REM  Plot mode.
1035 REM
1040 REM  Display initial line at top.
1050 FOR X=0 TO RGT
1055     IF LINE(X)=ALIVE THEN PLOT X,T0P
1060 NEXT
1063 REM
1065 REM  Main loop.
1070 FOR LINE=T0P-1 TO BOTT0M STEP -1
1080     GOSUB 4000 : REM  Calculate and display next line.
1090 NEXT
```

The program is rather slow, so have patience. You can insert some speed-up tricks, but translating the program into assembly language would represent a major speed improvement. (It might be interesting to try a version in FORTH.)

In the BASIC program, the initial line of live/dead cells is entered from the keyboard. The echo is turned off, and a two-high by one-wide plot block "cursor" is drawn. Pressing the space bar enters a "dead" cell and advances the cursor. Pressing any key other than the space bar and RETURN enters a "live" cell and advances the cursor. Pressing RETURN signals the end of input, the rest of the line is filled with "dead" cells, and the automaton begins its slow growth downward. □

COMP—U—WRITER

The popular word processor is still being sold. It was written by a company called INTERSELL for use on the Compucolor and Intecolor series of computers. Several readers have written to Colorcue with questions about the support for this software. We can only give you the names and address of the originators of the program and suggest that you contact them for further information.

INTERSELL
465 Fairchild Drive
Suite 214
Mountain View, CA 94043

Software: Thomas Crispin
Manual: Robert Moody

```
1095 REM
1100 PLOT 255 : REM  Exit plot mode.
1110 INPUT "";A$ : REM  Wait without destroying display.
1120 END
1130 REM
2000 REM  ================================================
2010 REM
2020 REM  Set up and instructions.
2030 REM
2040 PLOT 6,6,29,14,12
2050 PRINT ,,"1-D CELLULAR AUTOMATA"
2060 PRINT
2070 PLOT 15
2080 PRINT "Enter initial line. A space will represent a 'dead'"
2090 PRINT "cell, and any other key (except RETURN) will signal"
2100 PRINT "a 'live' cell. Press RETURN when done."
2110 RETURN
3000 REM
3010 REM  ------------------------------------------------
3020 REM
3030 REM  Get intial line.
3040 REM
3050 PLOT 3,64,0 : REM  Hide cursor.
3060 POKE KF,31 : REM  No-echo.
3070 CX=0 : CY=90 : REM  PLOT X,Y for inputting initial line.
3080 PLOT 30 : REM  Flag on so 2nd PLOT erases previous block.
3090 PLOT 6,COL0R,2
3100 GOSUB 3270 : REM  Display "cursor".
3110 POKE KB,0
3120 A=PEEK(KB) : IF A=0 THEN 3120 : REM  Wait for key.
3130 IF A=13 THEN 3180 : REM  "RETURN" for end of entry.
3140 GOSUB 3270 : REM  PLOT "cursor" a 2nd time to turn it off.
3150 IF A<>ASC(" ") THEN PLOT CX,CY : LINE(CX)=ALIVE : GOTO 3170
3160 LINE(X)=DEAD
3170 CX=CX+1 : IF CX<=127 THEN 3100
3180 POKE KF,12 : REM  Turn on echo.
3190 REM  Fill out initial line if necessary.
3200 IF CX<RGT THEN FOR I=CX TO RGT : LINE(I)=DEAD : NEXT
3210 PLOT 255,29 : REM  Exit plot mode; reset flag.
3220 PLOT 12
3230 RETURN
3240 REM
3250 REM  ------------------------------------------------
3260 REM  Plot the "cursor".
3270 PLOT CX,CY,CX,CY+1 : RETURN
3280 REM
4000 REM  ------------------------------------------------
4010 REM  Calculate next generation.
4020 REM
4030 FOR X=0 TO RGT
4040     GOSUB 310 : REM  Get new generation in TEMP().
4050     IF TEMP(X)=ALIVE THEN PLOT X,LINE
4060 NEXT
4070 REM
4080 REM  Transfer TEMP() to LINE().
4090 FOR X=LFT TO RGT : LINE(X)=TEMP(X) : NEXT
4100 RETURN
4110 REM
4120 REM  ------------------------------------------------
62000 REM
62010 REM  Ben Barlow's No-echo patch.
62020 REM
63000 RESTORE 63000 : DATA 245,175,50,255,129,241,201
63010 TM=256*PEEK(32941)+PEEK(32940)-7
63020 FOR X=1 TO 7 : READ D : POKE TM+X,D : NEXT
63030 BR=INT(TM/256) : POKE 33221,195 : POKE 33222,TM-BR*256+1
63040 POKE 33223,BR : POKE 32941,BR : POKE 32940,TM-BR*256
63050 CLEAR 50 : GOTO 10
```

KVHGRxRWRO KILTIZnNrmT

"...NLERMT RMGL GSY FKKVI IGMPH."

DQRIIB HIN DSROOB

I am aware of the patronizing attitude taken by some individuals, who learn something of value, perhaps print a few silly articles, and then retire with their wisdom forever. They reach some kind of summit from which they can no longer share, and decide all further questions are stupid. Have you, for example, ever seen an article that tells how to perform routine system debugging? Now you know why I have emerged from obscurity.

The idea of "bugs" in a computer system derives, so legend has it, from early experiences with vacuum tube computers at the Department of Defense, which were known to fail because of nests of moths and other little creatures. From this, we speak of our failures as being the responsibility of "bugs" in almost anything. So a "debugger" is meant to remove the human errors from software and hardware systems. We are discussing software debuggers.

There are five software debugging tools currently available for the CCII and 3651 computers; MLDP by ISC, DEBUG and NEWBUG by Com-tronics, Super Monitor Plus and IDA by Bill Greene. These are listed in increasing order of versatility, in my opinion, although each has its own special, useful features. MLDP differs from the others by its lack of printing facilities and is frustrating in this regard for serious programming. MLDP does, however, have a unique monitor which we shall examine next time. Most of the others are somewhat similar in their capabilities, but IDA stands out far above the rest. If you don't have IDA you are missing the companionship of a true friend, for it is the best there is, and an invaluable working and teaching aid.

These articles will tend to focus on IDA , but if you have one of the others and don't want to purchase IDA, I will include for you those commands which are somewhat comparable in MLDP and DEBUG, as far as they go. Super Monitor Plus is a "subset" of IDA, so many of IDA's commands will work with it as well. In addition to one or more of these debugging programs, you will need an assembled version of CYPHER (at 8200H) from the MAR/APR Colorcue, on a disk by itself, and another disk side with no valuable material on it. Any directory on this second disk will be obliterated in the course of this article. Your work will be facilitated by a printout of the assembly of CYPHER (see Colorcue Mar/Apr 84), showing addresses and mnemonic code.

A "debugging" software tool is a program that allows you to manipulate the contents of an FCS file, whether it be a PRG, BAS, SRC, LDA, or any other kind of file. The debugger is most often used with PRG files, to "run" them in selected portions and to examine what is taking place as they run. This kind of activity exercises the "monitoring" capability of a debugger. The debugger permits changing program code, in a limited fashion, without the necessity for reassembly, and it permits disassembling PRG files into their source code. A good debugger will permit you to correct faulty BASIC code lines that are making a program unexecutable. You may print a file in hex number form, or ASCII form. You may move code from one set of addresses to another. The list of possible uses is truly endless. I will introduce you to the power of debugging in a logical and disciplined way, using the program CYPHER.PRG.

A debugger is usually loaded into high memory, above the area used by most FCS files. This permits two programs to exist simultaneously in memory. In these articles the debugger is called the "instrument" and the program being worked on is called the "subject." If the subject code occupies some of the memory required by the instrument then debugging in the normal way is made impossible. If the subject only uses instrument memory to store data files, then debugging is still sometimes possible, even with this conflict. (IDA is available for loading at 4000H which keeps it out of the way, and therefore usable with any program that will fit into 32K CCII memory.)

To set up the instrument, the usual procedure is to load or run the subject program first from FCS. The LOAD instruction defaults to LDA, so if your program has a different file type, the load instruction must contain the file extension; example: LOAD CYPHER.PRG. With this command, CYPHER will be loaded into memory but will not "run." If the RUN command is used, the default type becomes PRG, so the command will read: RUN CYPHER. In this case, CYPHER will load and run both. To stop program execution, press [CPU/RESET] to exit, and [ESC] [D] to return to the file control system.

Debuggers will generally load at several of the addresses 4000H, 8200H, A000H, and E000H (look at the available loading addresses of your instrument on its disk directory, then pick a version that will load safely out of the way of your subject program). The last letter of the IDA name is the first letter of the loading address. (Some IDA versions are just named "8.PRG", "E.PRG", etc.) IDAE loads and runs at E000H. IDA8 loads and runs at 8200H, and so forth. If you have 16K memory and a subject at 8200H, then you must satisfy yourself with MLDP16 or DBUG16, or IDAA at A000H. If the subject program loads at E000H, then your debugger will have to be loaded below that, say at 8200H, so it does not conflict with the subject. To invoke the debugger, type from FCS a run command with the debugger name; example: RUN IDAE, or RUN DBUG32, or whatever.

Now it's time to get your feet wet. "LOAD" CYPHER.PRG, and when the FCS prompt reappears, RUN IDAE or some other instrument (into high memory). CYPHER begins at 8200H and ends at 847DH, so any of A000H, E000H or 4000H will do for the instrument. Since

we're loading into high memory, version ;02 of MLDP and DEBUG will be used, so a version need not be specified (unless you have ;03 for some reason):

```
[Note: Later versions of IDA use [ESC] for exits from all
IDA functions.  [ESC] will replace [DOWN/ARROW] where it
is specified in these articles.]
```

```
IDA (SUPER MONITOR):

FCS>RUN CYPHER
[CPU/RESET] [ESC] [D]
FCS>RUN IDAE
IDA>
```

```
DBUG32:

FCS>RUN CYPHER
[CPU/RESET] [ESC] [D]
FCS>RUN DBUG32
COMMAND>

MLDP:

FCS>RUN CYPHER
[CPU/RESET] [ESC] [D]
FCS>RUN MLDP32
DBG>
```

The last lines above illustrate the instruments' prompts, ready to receive a command. It is expected that when a command line is keyed in, you will follow with a [RET]. On DEBUG there are some commands that will result in an automatic RETURN. DEBUG also performs some automatic "space-overs" on command lines with multiple parameters. It will take a little "getting used to." We may now begin work. Let us look first at the elementary process of disassembly, in which the machine code is translated into the more readable mnemonics of the source code from which it was derived. The display will differ from original source code in that all labels will be replaced by their actual assembled addresses. The disassembler feature generally requires a disassembly command, a starting address, and a length of addresses to be disassembled. This length may take the form of an ending address or a number of addresses following the starting address. We will look at the first few complete code lines of CYPHER on the CRT.

```
IDA>D 8200 10+
[DOWN/ARROW] or [ESC]
```

```
COMMAND>S 8200 821F

DBG>DS 8200,20
```

What is seen with disassembly varies among the instruments. IDA will give the most spectacular readout (see LIST A), however, in this disassembly mode, most programs are very much alike. In List A, column 1 displays the memory addresses, column 2 shows the hexadecimal bytes located in those addresses (clustered in IDA, separated in other instruments), columns 3 and 4 show the disassembled mnemonic form of the code (columns 4 and 5 for MLDP), and column 5 prints a translation of column 2 which interprets memory as an array of characters and not mnemonic code. (In MLDP, this display appears as column 3, which "strips" lower case to upper case. The only way to tell if you're looking at a lower case character is to examine the hex code.) Look at address 820CH. The first byte represents the op-code for "CALL" (CD). The next two bytes are the low and high bytes of the address of the CALL (2A, 18 for v8.79 and v9.80, F4, 33 for v6.78). This line is calling OSTR in ROM. If you have IDA, in column 5 you will see "OSTR" printed. IDA translates all frequently used ROM addresses into their source code name (very helpful if you are modifying a PRG file for another version of system software). To find what OSTR is about to print, look at address 8209H and you will find the HL registers loaded with address 82D7H, which just happens to begin the printable string—6,2,3,0,10,'NORMAL TEXT'—etc. All the lines in Listing A are instructional lines of code, including the three NOPs at the beginning which give the instruction to do

Now let's move into the "high weeds" (as Myron used to say). Repeat the last disassembly but start at 8204H. You should get something like List B, and it looks nothing like List A at all until it reaches address 8206H, even though it has all but one byte of List A (not counting NOPs) in its disassembly. How come? The disassembler began to decode the memory contents at address 8204H, where it saw 25H, which happens to be the op-code for"DCR H." The disassembler will try to interpret the first byte it sees as an op-code. That isn't what we intend the 25H to mean. We mean it to be the low byte of the address of string CLR (at

```
IDA>D 8200 15+          List A

8200   00       NOP              ; @
8201   00       NOP              ; @
8202   00       NOP              ; @
8203   212583   LXI    H,8325H   ; !%C
8206   CD2A18   CALL   182AH     ; OSTR
8209   21D782   LXI    H,82D7H   ; !WB
820C   CD2A18   CALL   182AH     ; OSTR
820F   212083   LXI    H,8320H   ; ! C
8212   3600     MVI    M,00H     ; 6@     0
8214   23       INX    H         ; #
8215   360C     MVI    M,0CH     ; 6L    12
8217   0E00     MVI    C,00H     ; N@     0
8219   212383   LXI    H,8323H   ; !#C
821C   3602     MVI    M,02H     ; 6B     2
821E   3EC3     MVI    A,C3H     ; >C   195
8220   32C581   STA    81C5H     ; IPCRT
8223   21A782   LXI    H,82A7H   ; !'B
8226   22C681   SHLD   81C6H     ; ICRT1
8229   3E1F     MVI    A,1FH     ; >_    31
822B   32DF81   STA    81DFH     ; KBDFL
822E   C35E82   JMP    825EH     ; C^B
```

8325H). If the computer is going to see it that way, we **must** begin disassembly in such a way that the computer will recognize 25H as a low byte address of a preceeding op-code, specifically here, 21H, the op-code for "LXI, H", which is at address 8203. To satisfy this requirement, we must always begin disassembly at some single byte instruction or single byte of independent data, such as a DB byte. Otherwise, we'll be "in the high weeds" and working against the wind. If you're in a program unknown to you, then you must "feel" your way from the starting address, writing down legitimate starting places as you disassemble your way through the program. If this isn't clear, try disassembling at each address in turn, from 8200H to 820BH and see when the disassembly is congruent to List A and when it isn't.

The procedure for controlling the number of disassembly lines varies among instruments. In our examples, IDA is given the number of lines to decode (in hex), DEBUG is given the last line to decode, and MLDP is given the number of bytes to decode (hex). If the end specification falls at the beginning or in the midst of a two or three-byte instruction, the entire instruction will be decoded.

Now, using the same technique used to generate List A, disassemble the code beginning at 8340H. IDA will accept a number of different disassembly specifications, but I like specifying the number of lines the best. DEBUG wants the start and end address. MLDP wants the number of bytes to decode:

```
IDA>D 8340 10+      COMMAND>S 8340 8351      DBG>DS 8340,12
```

This will look very much the same for all the instruments, and is shown in List C. What you see in the mnemonic columns is interesting but only garbage. The "real" meaning

```
IDA>D 8204 15+                List B

8204  25       DCR    H          ; %
8205  83       ADD    E          ; C
8206  CD2A18   CALL   182AH       ; OSTR
8209  21D782   LXI    H,82D7H    ; !WB
820C  CD2A18   CALL   182AH       ; OSTR
820F  212083   LXI    H,8320H    ; ! C
8212  3600     MVI    M,00H      ; 6@     0
8214  23       INX    H          ; #
8215  360C     MVI    M,0CH      ; 6L    12
8217  0E00     MVI    C,00H      ; N@     0
8219  212383   LXI    H,8323H    ; !#C
821C  3602     MVI    M,02H      ; 6B     2
821E  3EC3     MVI    A,C3H      ; )C   195
8220  32C581   STA    81C5H      ; IPCRT
8223  21A782   LXI    H,82A7H    ; !'B
8226  22C681   SHLD   81C6H      ; ICRT1
8229  3E1F     MVI    A,1FH      ; )_    31
822B  32DF81   STA    81DFH      ; KBDFL
822E  C35E82   JMP    825EH      ; C^B
```

of these memory contents is in column 5 (column 3 for MLDP)—a translation of a string of ASCII characters. How do we know? For one thing, the disassembled mnemonics don't make sense. For another, the ASCII translation does. Furthermore, using the hard copy of the assembled version of CYPHER, we can trace this location back to address 8203, where the LXI H,CLR (8325H) instruction appears. OSTR will begin printing at 8325H and continue until it reaches a "239". (We are viewing a mid-section of this long string.) The first subsequent "239" occurs at address 8429H (EF). This won't just come to you, you have to study the printout of CYPHER to see it.

The subsequent area of memory contains the misspelled words, and we are going to correct them without reassembling CYPHER. Instruments allow you to change memory contents. Disassemble from 8352H and observe the word "prmgram". We will first replace the bad "m" in "prmgram" with a good "o". (If you don't have lower case, just use capitals.)

```
IDA>D 8352 10+
Press [UP/ARROW]
 (Bar cursor appears)
Hold [UP/ARROW]
 until it is on "m".
Press [P] (peek/poke)
 ASCII representation
 will appear on bottom
 of screen. Cursor is
 under the "m."
Now press [INSERT/CHAR]
 and lower case "o".
 Code will appear in
 hex, but ASCII will
 not change.
Press [RET] to exit the
 Peek/Poke mode.
```

```
COMMAND>S 8352 8362
COMMAND>E 835B
835B: 6D-
(Enter 6F)
[RET]
835C: 67-
[DOWN/ARROW]
COMMAND>
```

```
DBG>DS 8352,12
DBG>?835B
835B 6D 'M' MOV L,L
MEM>=6F
835C 67 'G' MOV H,A
MEM>/
DBG>
```

To see the wonderous results of your work, disassemble again from 8352H. We can get more practice at this by correcting misspelling at addresses 83B0 and 83DD. So do that now on your own.

Reference: "i" (69H), "I" (49H), "a" (61H), "A" (41H)

Instruments allow you to "run" a program in memory. Now "run" the program and observe the corrected spelling.

```
IDA>G 8200          COMMAND>G 8200          DBG>R 8200
```

After admiring your new spelling, note that the first "a" of "according" is at the end of a line, with the rest of the word on the following line. We can correct that easily with IDA.

```
IDA>D 8340 15+                List C

8340  54       MOV    D,H        ; T
8341  6F       MOV    L,A        ; o
8342  20       - -               ;
8343  45       MOV    B,L        ; E
8344  6E       MOV    L,M        ; n
8345  63       MOV    H,E        ; c
8346  6F       MOV    L,A        ; o
8347  64       MOV    H,H        ; d
8348  65       MOV    H,L        ; e
8349  20       - -               ;
834A  54       MOV    D,H        ; T
834B  65       MOV    H,L        ; e
834C  78       MOV    A,B        ; x
834D  74       MOV    M,H        ; t
834E  2E03     MVI    L,03H      ; .C     3
8350  00       NOP               ; @
8351  02       STAX   B          ; B
8352  0602     MVI    B,02H      ; FB     2
8354  54       MOV    D,H        ; T
8355  68       MOV    L,B        ; h
8356  69       MOV    L,C        ; i
```

Press [CPU/RESET] to exit CYPHER. Reenter any of the instruments by pressing [ESC] [user]. Now disassemble beginning at 8390 (List D). If we could place a carriage return and line feed after "it", then "according" would be altogether on the next line. The space (20H) between "it" and "according" can be used for the carriage return (we don't need the space at the end of the line anymore), but we need another memory location for the line feed. If we read the entire string given to OSTR, beginning at address 82D7, we find that there is unused memory following the "239" at address 842AH. Therefore, if we moved all the code, extending from 8393 to 8429, down one address, we would free a memory location at 8393 for the line feed.

```
IDA>D 8390 10+  List D (Before changes)

8390   69      MOV     L,C       ; i
8391   74      MOV     M,H       ; t
8392   20      - -               ;
8393   61      MOV     H,C       ; a
8394   63      MOV     H,E       ; c
8395   63      MOV     H,E       ; c
8396   6F      MOV     L,A       ; o
8397   72      MOV     M,D       ; r
8398   64      MOV     H,H       ; d
8399   69      MOV     L,C       ; i
839A   6E      MOV     L,M       ; n
839B   67      MOV     H,A       ; g
839C   20      - -               ;
839D   74      MOV     M,H       ; t
839E   6F      MOV     L,A       ; o
839F   20      - -               ;
```

Instruments allow you to move blocks of memory. The MOVE instruction has a starting and ending address of the block to be moved, and the new starting address of the block after the move. The "move" is really a "copy", since the original contents will not be altered unless it is written over. IDA will move from anywhere to anywhere, but DEBUG and MLDP will not. For these last two, you must move the memory area completely outside the working area in which it currently resides, and then move it back in again with a one-byte displacement. If you have DEBUG or MLDP, fill the area from A000 to B000 with 00 (to unclutter it and make examination easier). Move the string into that area, then move it back where we want it ..and think about saving money to purchase IDA. (My MLDP manual has an error for MOVE instructions. It is shown correctly here):

```
IDA>M 8393 8429 8394      COMMAND>Z A000 B000 00
                          COMMAND>M 8393 842A A000
                          COMMAND>M A000 A097 8394

                                DBG>F A000:B000:00
                                DBG>M 8393:842A TO A000
                                DBG>M A000:A097 TO 8394
```

(Move the range 8292-8429 to the range beginning at 8394.)

Disassemble from 8390 and notice two "a"s at 8393 and 8394. The first "a" can be replaced by a line feed.

```
IDA>D 8390 10+          COMMAND>E 8392          DBG>E8392
[UP/ARROW] to 8392      8392: 20-               8392 20 ' '
Press [P]               (Enter 0D)              MEM>=0D
Enter 0D (at 8392)      8393: 61-               8393 61 'A'
Enter 0A (at 8393)      (Enter 0A)              MEM>=0A
[RET] to exit           [DOWN/ARROW]            8394 61 'A'
                                                MEM>/
```

Now check it out. (See 'after' List E.) Run the program with 'G 8200' (or 'R 8200' for MLDP.)

Isn't that better! IDA is unique in that it can move bytes around in a very tight space. Try this with DEBUG or MLDP and you'll have an interesting experience.

We don't want to have to do these corrections twice, so let's save a new CYPHER on disk. Note we haven't increased the length of CYPHER, only changed its contents. You may use FCS commands to save a "memory image" (a picture of memory) on disk. This memory image just happens to be a working PRG program. Most instruments will make this SAVE from FCS, but if you have IDA, you can return to the instrument and do it from there.

```
IDA>XSAVE CYPHER.PRG 8200-847D

          MLDP AND DBUG: FCS>SAVE CYPHER.PRG 8200-847D
                         FCS>[ESC] [^]
```

We will now erase all the memory used by CYPHER plus a little more. Instruments allow you to fill a range of memory with any byte you may choose. The FILL instruction usually has a starting address, an ending address, and the byte with which you wish to fill:

```
IDA>F 8200 8500 00          COMMAND>Z 8200 8500 00
                                    DBG>F8200:8500:00
```

(Fill the memory range 8200-8500 with byte "00.") Verify the fill by disassembling from 8200H. All NOPs.

```
IDA>D 8390 10+  List E (After changes)

8390   69      MOV     L,C       ; i
8391   74      MOV     M,H       ; t
8392   0D      DCR     C         ; M  < CTRL M = cr
8393   0A      LDAX    B         ; J  < CTRL J = lf
8394   61      MOV     H,C       ; a
8395   63      MOV     H,E       ; c
8396   63      MOV     H,E       ; c
8397   6F      MOV     L,A       ; o
8398   72      MOV     M,D       ; r
8399   64      MOV     H,H       ; d
839A   69      MOV     L,C       ; i
839B   6E      MOV     L,M       ; n
839C   67      MOV     H,A       ; g
839D   20      - -               ;
839E   74      MOV     M,H       ; t
839F   6F      MOV     L,A       ; o
```

There is another way to load CYPHER into memory for further work. We know it isn't there now. Call up your disk directory and write down the starting block (SBLK) of the newest CYPHER (the corrected version.) My starting block is 207. (I have an 8" drive. See List H.) You will use your own appropriate block number where I have written "207" in the instructions:

```
IDA>XDIR

                DEBUG AND MLDP: [CPU/RESET] [ESC] [D]
                                FCS>DIR
```

We will use the FCS "REAd" command to get CYPHER this time. Most instruments will require that you do this from FCS, but IDA can do it from the IDA prompt. The REAd command contains the starting disk block number, the start address in memory and the end address in memory where the code will be placed. We can get the start and end addresses from the assembler printout of CYPHER:

```
IDA>XREA 207 8200-847D

                DEBUG AND MLDP:  FCS>REA 207 8200-847D
                                 FCS>[ESC] [^]
```

(Read from disk beginning at block 207, into memory beginning at 8200, until memory is filled to 847D.)

This loads CYPHER directly. Notice how fast this load is. Speed is increased because the operating system doesn't have to work with the directory. Disassemble from 8200H to verify this load. Now we are going to write a new subroutine into memory and use it. Notice the instruction, at 8206H, CALL 182A (CALL OSTR). (Your address will be 33F4 for v6.78.) This is the famous output string subroutine in ROM. Let's not use it from ROM, but write our own version of OSTR inside CYPHER. When OSTR is called, the HL registers hold the address of the first byte to be printed. The last byte is followed by "239." This routine (which we shall call OMSG) will do:

```
Label Mnemonics Operation..................... Address  Hex Code

OMSG: MOV A,M ; Move memory byte to accummulator  842D      7E
      CPI 239 ; Is it the end? (239=EFH)          842E      FE EF
      RZ      ; Yes, return to caller             8430      C8
      CALL LO ; No, print this byte               8431*     CD C8 17
      INX H   ; Point to next memory location     8434      23
      JMP OMSG;  and get next byte                8435      C3 2D 84

                    * "CD F4 33" for v6.78.
```

There is room for this routine at 842D (leaving a few bytes free beyond the ASCII string). We cannot use it as shown above. for we must replace all the labels with actual addresses. Instruments allow you to insert op codes and their accompanying addresses in memory in the form of hex numbers. IDA will let you enter the mnemonics directly, the rest of you will have to poke in the hex numbers, one at a time from the table above:

```
IDA>D 842A 15+          COMMAND>E 842D        DBG>@842D
[UP/ARROW] to 842D      842D: FF-             842D FF '_' RST 7
Press [0] to set        (Enter  7E            MEM>=7E
 origin for IDA's               FE            MEM>=FE
 assembler.                     EF            MEM>=EF
ASM>MOV A,M                     C8            MEM>=C8
ASM>CPI 239                     CD            MEM>=CD
ASM>RZ                          C8 (*)        MEM>=C8 (*)
ASM>CALL 17C8 (*)               17 (*)        MEM>=17 (*)
ASM>INX H                       23            MEM>=23
ASM>JMP 842D                    C3            MEM>=C3
[RET]                           2D            MEM>=2D
                                84            MEM>=84
                        [DOWN/ARROW]          MEM>/

* Use 33F4 for v6.78 (F4, 33)
```

Disassemble from 842A to verify that the subroutine has been entered (List F). Now we have to call 842D every time we formerly called 182A. IDA makes it easy to find all the references to OSTR in our program. We will conduct a "simple search" for the code string "CD 2A 18" (or CD F4 33), placing the low byte first for the OSTR address. While these address bytes "read right" in the disassembly, you know by now that they are reversed in memory:

```
IDA>SS 8200 8500 CD 2A 18
```

```
COMMAND>F 8200 847D CD 28 1A  (Sorry. MLDP can't
                               do this.)
```

IDA and DEBUG locate this code at four places: 8206, 820C, 827D, 8290. With MLDP, you would have to disassemble the entire file, and watch for OSTR's address, making note of its occurrances. IDA will also replace the code at these addresses if we ask it to. We perform a "search and replace." When using this function IDA asks for your "replace" string (RPL):

```
IDA>SR 8200 8500 CD 2A 18
RPL>CD 2D 84

While they're doing their
thing (to the right),
let's check it out by
disassembling at least
one address, using
IDA>G 8200, to watch our
new OMSG at work! By that
time maybe the others will
be ready to go on!
[CPU/RESET] [ESC] [^] to
return to IDA).
```

```
COMMAND>E 8206
8206: CD-
[RET]
8207: 2A-
(Enter 2D
8208: 18-
(Enter 84
[DOWN/ARROW]
COMMAND>E 827D
827D: CD-
[RET]
827E: 2A-
(Enter 2D
(etc.)
```

```
DBG>@8206
8206 CD 2A 18
MEM>=CD
MEM>=2D
MEM>=84
MEM>=/
DBG>@827D
MEM>=CD
MEM>=2D
MEM>=84
MEM>/
(etc.)
```

(Be sure you change all four addresses.)

We can now save this new version of CYPHER, but rather than create still another new disk file, we will write new version 2 over new version 1. We can do this easily with the FCS WRIte instruction. Using the same SBLK as before we will transfer new version 2 to disk. Substitute your own disk block number for my "207" again:

```
IDA>XWRI 207 8200-847D
                    DEBUG AND MLDP: [CPU/RESET] [ESC] [D]
                                    FCS>WRI 207 8200-847D
```

Now let's write another copy of CYPHER to a clean disk. Remove the first disk and put a clean disk in the drive. We don't really need a directory, you know, so let's write CYPHER beginning at block 00. Do not use a disk for this procedure that contains anything you want to keep. Any directory on this disk will be destroyed:

```
IDA>XWRI 00 8200-847D
                    DEBUG AND MLDP: FCS>WRI 207 8200-847D
```

Now look (or try to look) at this disk directory. You will get an ENVE error because there is no directory. But CYPHER is there, my friends. Fill 8200-8500 with 00 again and verify that this memory is cleared. Now read CYPHER back into memory from the same no-longer-clean disk:

```
IDA>XREA 00 8200-847D
                    DEBUG AND MLDP: FCS>REA 00 8200-847D
                                    FCS>[ESC] [^]
```

Disassemble from 8200H. There it is! Isn't this great fun! You can "run" the program with "G 8200" if you are skeptical.

For our last act this time, we are going to change the encoding used by CYPHER. We will be original and let "A" = "A", "B" = "B", and so on. This is an easy change to make. The encoding table begins at 8448. The first letter ("A") is the "normal" letter of your keyboard input. It is followed by the letter to which the first letter will be encoded ("Z"). This correspondence between pairs of letters continues to 847E. The last two entries, at 847C and 847D "convert" spaces to spaces, to allow them as legal entries into the code. (Any character not represented in the table is not recognized. Does that give you any ideas for the space beginning at 847E?) List G is IDA's printout of this memory range in ASCII. The other instruments will make a hex dump of this table to use for reference as you poke in the new data. Be careful you get the right address for data entry:

```
IDA>D 8448 10+
[UP/ARROW] to 8448.
```

```
COMMAND>H 8448 847E
```

```
Press [P] for Peek/Poke
option.
[RIGHT/ARROW] to 8449
Press [INSERT/CHAR] [A]
[RIGHT/ARROW] to 844B
Press [INSERT/CHAR] [B]
Continue until all
 letters are changed.
At the end of the first
 line of ASCII display,
 the next line will be
 displayed automatically.
Press [RET] to return
 to IDA).
```

```
COMMAND>E 8449
8449: 5A-
(Enter 41
[RET] [RET]
(Enter 42
[RET] [RET]
(Enter 43
(etc.)
.........
(Enter 5A
[RET]
[DOWN/ARROW]

DBG>D 8448:847E
(Error in MLDP manual
for this command. It
is shown correctly here.
DBG>@8449
8449 5A 'Z'
MEM>=41
[RET] [RET]
MEM>=42
[RET] [RET]
MEM>=43
(etc.)
.........
MEM>=5A
847C: 20
MEM>/
```

Verify the new encoding table with a dump from 8448 to 847D. You should see pairs of hex numbers from 41 to 5A. You can also try the program by running it again.

We are going to save this new encoding table as a separate disk file. We will put it on the disk with no directory. In order to do this, we need to know how much disk space CYPHER occupies, so we won't overwrite CYPHER accidently. CYPHER begins at 8200 and ends at 847D. Let's calculate the number of bytes. (There is a mistake in my MLDP manual regarding calculations. The "/" symbol is not to be used at all. Numbers entered directly will be assumed as hexadecimal numbers. Numbers preceeded by a "#" will be taken as decimal.)

```
IDA>847D-8200=
027D  637  00000010 01111101
```

```
DBG>=847D-8200
=027D #637
```

Calculations say the program is 027DH or 637 decimal bytes long, but we must also add the end byte, which makes 638. Wait a minute! With IDA we don't have to calculate this. If we enter a "V" at the IDA prompt, IDA prints the usual directory but adds the file size in hex and decimal for us (List H). Well, anyway, how many blocks is this (at 128 bytes/block)?

```
IDA>027D/%128=
```

```
DBG>027D/#128
```

The calculators say 4 blocks, but they are performing integer arithmetic. It's actually 4.98 blocks long. We'll say 5 blocks because we must write with integer values of block numbers. FCS begins numbering its blocks with 00, so the sixth block, where we might plan to put our new table, is actually referred to as block "5". For some reasons that might become apparent later on, let's store it beginning at block 0A. The code table begins at 8448 and ends at 847B:

```
IDA>XWRI 0A 8448-847D
                    DEBUG AND MLDP:  [CPU/RESET] [ESC] [D]
                                     FCS>WRI 0A 8448-847D
```

Now we can use either code table. If we REAd CYPHER to memory from block 00 we will have the original table:

```
IDA>XREA 00 8200-847D

                DEBUG AND MLDP:  FCS>REA 00 8200-847D
```

If we "overlay" the code table from block 0A we will have the amended code table:

```
IDA>XREA 0A 8448-847D
```

There is no directory to give tell-tale clues about the disk contents. We have a rather well-protected encryption procedure as long as we don't forget which block is which.

Well, let's try one more act. Load CYPHER.PRG back into memory from block 00. Now initialize the "clean" disk we have just been using, but only allow "five" directory blocks:

```
IDA>XINI CD0:FREE 5

                DEBUG AND MLDP:  FCS>INI CD0:FREE 5
```

Call up the directory and find the first available unused block (SBLK for "Free Space.") WRIte CYPHER to that start block (my SBLK is 05):

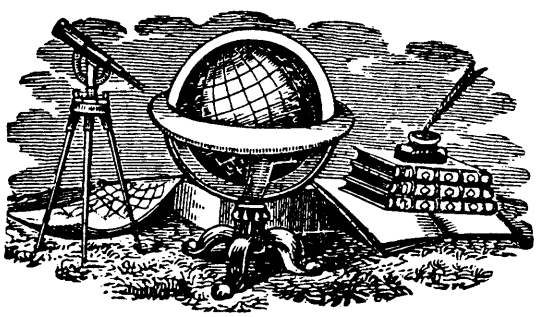
```
IDA>XWRI 05 8200-847D

                DEBUG AND MLDP:  FCS>WRI 05 8200-847D
```

If you look at the directory again, you'll notice CYPHER isn't listed even though we just "wrote" it there. Now go into BASIC by pressing in sequence [CPU/RESET] [ESC] [E]. We are going to create a directory entry for CYPHER.PRG from BASIC! Since CYPHER takes nearly five blocks, we can reserve 5 * 128 bytes/block for it (=640 bytes total). Type the following in "immediate" mode:

```
FILE "N", CYPHER.PRG, 1,640,1
```

Call up the directory from FCS. We have a record of CYPHER's presence. Type "RUN CYPHER." Honestly now, did you really think it was going to work? Well, it can if we put in the correct loading and starting addresses..... next time. Meanwhile, you can WRIte it into memory and "run" it from your instrument; no one else is going to RUN your secret encoder! If you wrote CYPHER to disk beginning at block 05, then the "overlay" is still intact at block 0A. You can load it just as we did before. For homework, why not create a directory entry for the overlay? More fun to come!□ *W. S. Whilly*

Note: All listings and their comments were printed directly from IDA's screen. A "CRT mode" facility permits writing comments anywhere on the screen for inclusion in printouts. Virtually any IDA screen display may be sent to the printer during the course of work.



```
IDA>D 842A 15+          List F  (DMSG)

842A   EF       RST     5           ; o
842B   FF       RST     7           ;
842C   FF       RST     7           ;
842D   7E       MOV     A,M         ; ~         (
842E   FEEF     CPI     EFH         ; ~o   239  (
8430   C8       RZ                  ; H         (
8431   CDC817   CALL    17C8H       ; LO        (
8434   23       INX     H           ; #         (
8435   C32D84   JMP     842DH       ; C-D       (

8438   FF       RST     7           ;
8439   FF       RST     7           ;
843A   FF       RST     7           ;
843B   FF       RST     7           ;
843C   FF       RST     7           ;
```

```
IDA>P 8448    List G      (The encoding table)

8448 49 4A 4B 4C 4D 4E 4F  50 51 52 53 54 55 56 57
  A  Z  B  Y  C  X  D  W   E  V  F  U  G  T  H  S
 41 5A 42 59 43 58 44 57  45 56 46 55 47 54 48 53

8458 59 5A 5B 5C 5D 5E 5F  60 61 62 63 64 65 66 67
  I  R  J  Q  K  P  L  O   M  N  N  M  O  L  P  K
 49 52 4A 51 4B 50 4C 4F  4D 4E 4E 4D 4F 4C 50 4B

8468 69 6A 6B 6C 6D 6E 6F  70 71 72 73 74 75 76 77
  0  J  R  I  S  H  T  G   U  F  V  E  W  D  X  C
 51 4A 52 49 53 48 54 47  55 46 56 45 57 44 58 43

8478 79 7A 7B 7C 7D 7E 7F  80 81 82 83 84 85 86 87
  Y  B  Z  A        1  3   @  \  D  H  @  V  B  (
 59 42 5A 41 20 20 31 33  00 9C 84 C8 00 96 C2 28
```

```
DIRECTORY DF0: CYPHER     05     List H      BYTE  COUNT

ATR  NAME TYPE VR SBLK SIZE LBC LADR SADR   HEX     DEC

03 CTE    .PRG;01 0005 0022 80  8200 8200  1100    4352
03 CTA    .PRG;01 0027 002E 43  82A0 9000  16C3    5827
03 IDA8   .PRG;01 0055 0040 80  8200 8200  2000    8192
03 IDAE   .PRG;01 0095 0040 80  E000 E000  2000    8192
03 MLDP   .PRG;01 00D5 003F 80  8200 8200  1F80    8064
03 MLDP   .PRG;02 0114 003F 80  E000 E000  1F80    8064
03 DBUG32.PRG;01 0153 0021 60  EB00 EB00  1060    4192
03 SMP798.PRG;01 0174 0026 80  8200 8200  1300    4864
03 SMP79A.PRG;01 019A 0026 80  AD00 AD00  1300    4864
03 SMP79E.PRG;01 01C0 0026 80  ED00 ED00  1300    4864
03 CYPHER.SRC;01 01E6 001C 56  0000 0000  0DD6    3542
03 CYPHER.PRG;01 0202 0005 7E  8200 8200  027E     638
03 CYPHER.PRG;02 0207 0005 7E  8200 8200  027E     638  <<<<
01 <FREE SPACE>  020C 1000                0000       0
```

SOFTWARE CATALOG

The following information is reproduced by courtesy of Intelligent Computer Systems, 12117 Comanche Trail, Huntsville, AL 35803. (205-881-3800). It is extracted from their software catalog for CCII, 3650, and 9650 computers. Prices are current. To our knowledge, ICS is the only 'full line' East Coast distributor for CCII software. Personal experience has proved them to be conscientious, reputable, and knowledgable dealers. Colorcue recommends that you consider purchasing any of these software selections soon, because their availability in the future is uncertain.

The software catalog number appears first, followed by an evaluation code of six characters with the following significance:

PART NUMBER

SEPARATOR (' + ' indicates ROM version available.)

9 9 0 0 3 2 / Q D T F E C

User Experience Required
- A: Beginner
- B: Intermediate
- C: Advanced
- D: Expert
- X: Not rated

Available Disk Formats
- 1: CCII/3621 v6.78
- 2: CCII/3621 v8.79
- 3: 3651/9651 v9.80
- 4: 3651/9651 v9.80-3
- 5: Both 1 and 2
- 6: Both 2 and 3
- 7: Both 3 and 4
- 8: All types

Quality of Software
- A: Good
- B: Average
- C: Needs Improvement
- D: Inadequate
- X: Not rated.

Status of Documentation
- A: Good
- B: Average
- C: Needs Improvement
- D: Not available, not needed
- X: Not rated

Available Disk Types
- 1: 5" MD for 3651/9651
- 2: 5" CD for CCII
- 3: 8" FD for 3651/9651
- 4: 5" MD or 8"FD for 3651/9651
- 5: All disk types

Category of Software
- A: Business
- B: Finance
- C: Engineering
- D: Educational
- E: System Utilities
- F: Special Applications
- G: Games and Entertainment

BUSINESS

JH102A/AA58BA ASSEMBLY DATA BASE/$85. Assembly language program to write data fields into formatted SRC files; includes editing, sorting, and printing facilities. Change, delete, edit, review, sort, and search. Print entire record or up to five selected fields. Screen editor may edit files. Convert existing RND files to SRC. Fast disk access. Options available for extended performance.

JW129/AA58AA CARAMOR/$24.58. Car expense accounting with calculation of cost/mile, cost/day, etc. Can determine cost effectiveness of lease/rental/own vrs public transportation.

991545/AB58BA COMP-U-WRITER/$190. Full-feature word processor with single key commands, full screen editing, move, copy, delete, search and replace, boldface, underlining, selection of print parameters, fast disk access, mailmerge facility. Professional performance. (Colorcue is written on Comp-U-Writer. Ed.)

BG115 + AB25BA COMPUCALC/$120. "Spreadsheet" program in assembly with unique file recovery system, stores data on ARY files. Contains usual "Visicalc"-like features. Efficient, trouble-free software.

EM122/BB58AA DECISION MAKER/$10. Program examines input parameters for best buy or optimum decision. Optional printer readout of input and output data.

EM111/AA58AA INVOICE & ORDER/$30. Write personallized invoices to printer, with inputs for discounts, taxes, shipping, credit card numbers; input error correction provided. Totals and customer ID can be stored on disk. Orders may also be written, with similar features, and from a data base for frequently-used vendors.

EM109/BD58AA LABEL/$15. Use any line printer to print address labels. Special print features of Epson and IDS printer families supported. Print individually or from disk file.

JW104/BA58AA PERT/$39.50. Create 7 PERT plans with 200 activities each on one disk. Create, modify and analyze plan to determine critical paths, with data accounting.

JH114 + AB58AA PLANNING CALENDAR/$17.50. Assembly language program to log 7 events/day to total of 400 events. Specify daily, weekly, monthly, quarterly, bi-annual, and annual repetition. Requires screen editor or assembly data base to set up files.

EM103/BD58AA SCHEDULE/$49.50. Generate, update, and save bar graph screen displays of scheduling. May be printed on Epson and IDS printer families using special character functions. Designed for multi-national aerospace projects and is readily applicable to many other complex situations.

EM120/BD58AA VU-GRAPH/$10. Quick generation of VU-Graphs for printout, with descriptor fields, centered text, highlighting.

991005/BA58CB BONDS/$20. See effects of bond price and yield by variables, use realtime dates or time-to-maturity, up to 5 call dates for callable bonds. Print amortization schedule with a single keypress.

991007/AA58BA EQUITY/$20. Depreciation by straight line, double-declining balance, constant percentage, sum of digits, and sink-fund methods. Capitallized cost of periodic changes of up to three assets simultaneously to determine best option. Solve for any variable in the capitalization equation.

EM202/BD58AB FINANCIAL PROGRAMS SERIES/$18.50. Twenty programs for daily use in business management, including future investment value, calculation of effective interest rate, earned interest table generation, depreciation rate, depreciation amortization, salvage value, principal loan, payment on loan, last payment on loan, term of loan, mortgage amortization, and more.

EM205/BD58AB GENERAL LEDGER/$48.50. Capacity is 99 credit and 99 debit accounts with 200+ records per month. Printouts include accounts summary report by month and quarter, quarter and year, and complete account report by item.

EM201/AD58AB GRAPHICS CHECKBOOK/$38.50. Accommodates 15 expense and 6 deposit categories. Display or print (with Epson or IDS) tables sorted by date, totals of one or all categories. monthly cash flow, totals for category, for month, for year.

JH206 + AA58AB LEDGER PAD/$75. For business and home financial data. 80 rows by 32 columns, with titles, data editing, arithmetic computation, report generation (to screen and printer). Supports numbers to 99,999,999,99 and computes to 2 to the power 40 with $ 0.01 accuracy.

DP208/AA58AB PERSONAL BUDGET/$29.95. Enter and edit monthly budget items by category, show CRT summaries in tabular and plot form of income, expenses, and "net worth"(tabular form also on printer.)

991001/BB58BB PERSONAL FINANCE/$20. Calculate annuities, interest rates, and mortgage payments by several methods.

EM204/AD58AB REAL ESTATE INVESTMENT/$18.50. Calculate best mortgage, monthly payments, investment return, positive or negative cash flow.

DP207/AA58AB STOCK FUND SWITCH STRATEGY/$39.95. For telephone switch mutual funds (or similar) to determine maximum earnings. Favors switch to common stock during market rise, and to cash or money market on market fall.

ENGINEERING

AUG401/BD58AC ELECTRICAL ENGINEERING I/$10. Color graphics supported; Ohm's law for AC, DC circuits. Transformerless power supply design.

AUG402/BD58AC ELECTRICAL ENGINEERING II/$10. Color graphics supported; attenuator design for T, H, PI, and O pad configurations. Multivibrator timing design for astable and one-shot.

SP302/AB25AC GASMIL/$19.50. From mileage and gas purchase, computes last miles/gallon, average miles/gallon for last three fill-ups, and average miles/gallon to date. Data, by table or chart, to Epson MX-80 printer.

993004/CD58BC STATISTICS 1/$25. 1) FILES: generates, maintains and displays files for use by other programs; 2) REGRES: linear, log, exponential or reciprocal regressions with confidence limits and graph; 3) PLOT: plot one to three graphs on rectangular coordinates from disk file or equation; 4) STAT: compute measures of central tendency, dispersions, skews, movement about the mean, from grouped or ungrouped data; 5) GRAPH: display histograms or polygonial graphs from grouped or ungrouped data.

993006/CD58BC STATISTICS 2/$30. 1) FILES; 2) MLTREG: multiple linear regressions to 6 variables, with or without transformations; 3) POLREG: polynomial regressions to 5th degree; 4) DISREG: fits binomial, norm, or Poisson distribution to input data, plus CHI-square check for goodness of fit; 5) VARINZ: test several sets of data for variance, estimate evaluation mean.

993008/CD58BC STATISTICS 3/$30. 1) FILES; 2) TIMSER: smooth time series by trend regression and other cyclical procedures; 3) INDEX: computes eight types of index data for several sets of data; 4) CMPTIM: computes variation within or between sets of data; 5) RANK: rank analysis on data oairs using Mann-Whitney test, computes rank correlations.

EDUCATIONAL

BM310/AA25BD ASSEMBLY LANGUAGE TUTORIAL/$50. 18 lessons in assembly language programming.

992516/AA58BD BASIC LANGUAGE I/$25. Two disks, beginning tutorial.

992519/AA58AD BASIC LANGUAGE II/$20. Program algorithms, more advanced commands.

992512/AB58AG HANGMAN/$20. Word game. Disk also includes MATH TUTOR for simple arithmetic and logic, and TWO TO TEN, a simple card game for practice in sums.

992514/BB58AD MATH TUTOR/$20. Simple arithmetic. Disk also includes CHECKBOOK, for checkbook balance and income tax data; RECIPE for storage and retrieval of favorites; MATH DICE for practice of elementary arithmetic, and the BIORHYTHMS program from the Sampler.

AUG301/BD58AD PRIMES & PRIME FACT/$10. Prime numbers and factoring to primes.

SYSTEM UTILITIES

SYSTEM UTILITIES [Comments in this section by COLORCUE. Starred programs are highly recommended as basic software packages for both the CCII and 3650 series computers. Evaluations are made on the basis of completeness, reliability, versatility, and usefulness with all software versions. In some cases other software may perform as well for your needs. Only one recommendation is made in each category.]

Definitions: Screen editor: used to enter assembly language source code and any other text that can be stored in a SRC file (containing only ASCII characters). May function as a mini-word-processor. Printing capability to serial printer. Easy editing of text in full view on CRT. Will load and edit any SRC or TXT file, as well as any ASCII file with optional file extension type. Monitor: a program to examine and modify the 8080 registers in the course of program execution, sometimes single-step through a program, examine and modify memory. Assembler: a program to convert ASCII source code (mnemonics) into machine code, producing a memory image file, ultimately "runnable" as a finished PRG program. A good assembler finds coding errors and flags them, and produces hardcopy of assembly showing source code, machine code, label chart and address assignments. Debugger: an all-purpose tool for working with any kind of disk file, principally used to work with PRG files. A complete debugger will act as an interpreter, monitor, mini-assembler, and disassembler. It facilitates examination and repair of disk file damage, disassembly of ROM, error correction in BAS files.

JH855 + AA25CE ASSEMBLER/$20. Jim Helm's assembler with printer driver.

JH811/AB25BE BASE2/$15. Set printing options on BASE 2 printer.

** **QS816 + AB58BE 'The' BASIC EDITOR/$49.95.** Quality Software's tour-de-force.

CT851 + AA25AE BASMERGE/$19.95. Merge Basic programs.

JH856 + AA25BE BASSRC/$19.95. Basic code to SRC file conversion.

** **CT830 + XX25XE CLIST/$19.95.** List Basic programs to printer; converts control characters to printable symbols for increased accuracy of printed program. Works with more printers than most.

CT828/XX25CE COMTRX/$19.95. Terminal software, for network communications.

JH857 + AB25CE CROSS REFERENCE GENERATOR/$10. Generates cross reference of JMPs and CALLs from assembly code. SRC code for program available.

** **CT852 + AA25CE CTA-ASSEMBLER/$34.95.** Probably the best all-around assembler for ISC computers using FCS. Good printer driver, good error displays.

** **CT833 + AB58CE CTE-SCREEN EDITOR/$54.95.** Text move, copy, delete, control character inserion. Will accommodate upper-case-only if necessary; all keyboards. Printer driver not as good as CTA, but adequate. Designed for source code entry, may be used as limited word processor.

CT841/AA22BE DEBUG/$29.95. Editor, monitor, disassembler, mini- assembler. Fine debugging tool, supports serial printer.

CT825 + AA58AE DFM/CRC/$49.95. Disk duplicating; complete disk on 1 or 2 drives, single file to either drive. Scan disk by sector, protect files, change directory, write unique format.

JH838 + AA25BE DIRECTORY/$15. Creates directory SRC file compatible with Jim Helm's assembly database program.

CT822/AA25BE DIRMOV/$24.95. A directory handling system for up to 600 disk directories, with machine language sort by name or number; search, delete, add, save, or print results.

BG836/BD25BE DIRECTORY ORGANIZER/$19.50. Saves directory informaion in SRC file, for editing by screen editor.

JH837 + AB25AE DISK EDITOR/$20. Move, copy, rename disk files.

JH809 + AA58BE DRIVER/$10. Printer driver for SRC files, set printing parameters. Supports special control codes for Epson and Base2 printers.

JH815/AB25CE EXPANDED ASSEMBLER/$20. Printer driver, line overflow function, program count, auto-pause on error.

JH814/AA58BE EXPANDED SCREEN EDITOR/$20. Complete screen text editing with move, copy, delete block functions. Good programming. Not for 3651.

CT850 + AA58AE FILEMRGE/$24.95. Merge SRC files.

991527/AB47AE FORMATTER/$25. For 3651 5" or 8" disks.

SP845/AD25AE FORMATTER/$24.95. For CCII, includes disk drive speed monitor and MAZE game.

991532/AA58CE FORTRAN/$75. Fortran compiler, linker and library. Experienced users report satisfaction with this version of FORTRAN.

JH813 + AX58AE HEXDEC/$10. Hexadecimal/decimal memory dump of system or selected sections.

JH839 + AA25CE HEXDIS/$15. Assembly language disassembler, displays hex, object code, mnemonics to screen and printer. Creates SRC disk file is requested.

** **BG851/AA25CE IDA-Interpreter, Disassembler, Assembler/$49.50.**
This gets an entire page of stars. Software debugging tool. It is not a SRC code assembler in the usual sense, but does everything else to perfection. Fascinating graphics, unbelievable printer facilities, excellant monitor, absolutely written to ease debugging tasks. If you are a serious programmer, this one will do it all, better than you ever thought possible. Available for loading at 4000, 8200, A000, E000. [See W.S. Whilly's article, and review this issue.]

CT853/AA25CE LDA FILE/$19.95. Convert LDA files to PRG the easy way.

BM810/XA25CE LDIS/$29. Two-pass labelling disassembler.

JH806 + AB58AE LISTER/$10. List BAS files to printer. Screens out control codes.

CT829 + AB25AE LLIST/$19.95. List BAS files to printer, set printer parameters.

JH812/XB25BE LOAD76/$10. For BASE2 printer; loads and prints user-defined character fonts.

991539/AB58DE MACROASSEMBLER/$50. Similar to assemblers used by "professional" computers. Includes linking loader and cross-reference generator. Some impossible bugs but generally usable for most things. Difficult instructions.

SP842/AD25AE MX80 BLOCK GRAPHIC DUMP/$14.50. MX80 graphic dump of CCII screen.

CT854 + AA25CE NUBUG/$39.95. Software debugging tool. Improved version of DEBUG.

CT831 + AA58BE PRINT II/$19.95. Printer driver portion of Com-Tronics software. Fine driver; set parameters, pause.

CT847 + AA58AE RENUM/$19.95. BAS file renumbering.

JH805/AB25DE REPACK/$10. Removes REM statements from BAS files.

JH840 + AA58BE SCREEN EDITOR/$20. Requires deluxe keyboard, complete screen editing. Not for 3651.

CT844 + AA58AE SRCPRT/$24.95. Machine language dump of CCII screen to printer (especially IDS printers). Rests in high memory and 'on call' with key press. Requires very careful handling. Not satisfactory on 3651.

JH805/AB58BE SMERGE/$10. Merge SRC disk files.

CT831 + AA58BE SORT/$19.95. Binary-weighted sort program, callable from BASIC. Sorts up to 1000 variable length array elements. Rests in high memory or at 4000H. Reliable.

JH808/AB25CE SORT PROGRAMS/$20. Series of programs for SRC files; sort, set fields to equal length, add fields; add, remove, substitute, rearrange order of data. Versatile programming for SRC.

JH802/AB25CE SOURCE/$30. SRC file from PRG file. Select dump to printer and to disk file. Set printer parameters.

CT832 + AB25BE SRCBAS/$34.95. The reverse of BASSRC. Convert SRC files into executable BAS files. Good for modem users.

JH803/AB25CE STRIP/$10. Remove sections of SRC files to disk. Set up SRC code subroutine library.

CT846/AA25AE SUPER MENU/$19.95. Select programs from directory display by bar cursor.

BG801/AB58CE SUPER MONITOR/$30. Subset of IDA (see above) and really obsolete because of it. Good monitor and debugger, but 'how ya goin' ta keep 'em down 'n the farm...?

BG835 + AX58CE SUPER MONITOR PLUS/$49.50. Ditto.

** **CT821 + AA58BE TERM II/$69.95.** Terminal software. The early version seems more bug-free, the second is easier to be with visually. Fancy provisions for uploading and downloading, setting transmission parameters. Good writing, but can't someone get rid of the bugs? Ask ISC for both versions.

CT831/BB25AE VLIST/$19.95. List BAS file variables to screen or printer.

SJ843/AA25BE XEDIT/$25.50. 'A totally different screen editor' with programmable function keys. New program, but we've had no experience with it. Can someone write a review? It sounds interesting.

GAMES

990046/AB58AG AIRRAID/$20. Airraid, Car race, Rover robot and Tiles.

SR962/AD37AG ALIENS/$25. Good graphics, fast.

AUG930/BC58BC B747 FLIGHT SIMULATION/$15. New York to Hartford. We need a review!

JH908/AA58AG BIORHYTHM/$12.50. Assembly language with printout.

990040/AB58AG BLACKJACK/$20. Blackjack, Roulette, Drag race, Horse race, Slot machine.

990052/AA58AG BOUNCE/$20. Bounce, Battleship, Slither.

AUG934/AB58AG BOUNCE BALL/$10. Bounce ball, Crossword, Create crossword puzzle, and 2 person chess.

990036/BB58AG CHESS/$20. Chess, Acey-ducey, Line five, Biorhythms.

SB950/AA58AG CHOMP/$29.95. Assembly masterpiece by David Suits.

JH901/AA25AG CRIBBAGE/$20. A peg board on a screen.

990042/AB58AG CUBIC TIC-TAC-TOE/$20. Tic-tac-toe, Greed, Galaxy, Space lander.

RT960/AD25AG FINAL FRONTIER/$25. Rick Taubold's space battle with the Klingons.

AUG933/AD58AG FOOTBALL/$10. Football, Space colony, Drop the marble,15 piece puzzle, Mastermind.

JH937/AB25AG FOOTBALL/$20. Assembly language, with real time clock.

QS910/AD26AG INVADERS/$34.95. The 'arcade' version with sound, Tic-tac-toe, Battleship.

JH906/AD25AG KALEIDOSCOPE/$10. Assembly language graphics thriller.

JH905/BB25AG LIFE/$10. Assembly language life and death struggle.

990056/AB58AG LUNAR LANDER/$20. Lunar lander, Coalition, Linko.

PS952/AD58AG MAZE/LUNAR LANDER/$14. Help a mouse find the cheese, and David Suits land on the moon.

990060/AB58AG MAZE MASTER/$20. Maze, Crossword puzzle, Create crossword.

BG936/BD25AG MILLEBORNES/$15. Complete version of the French card game.

AG935/AD58AG MONOPOLY/$10. Monopoly, Maze, Puzzle, and Hyper-space war game.

990034/AB58AG OTHELLO/$20. Othello, Math dice, Concentration.

WL949/AB25BG PROJECT APOLLO/$14.50. Dock Apollo on Skylab. Good graphics.

JH904/AB25AG PYRAMID SOLITAIRE/$15. The card game for the recluse.

SR961/AD37AG ROBOT WARS/$25. Fast action space game.

AUG932/BD58AG ROULET/$10. Roulet, Reverse number game, Dog chase cat, Ask Eliza, Clock, Robot.

JH907/AD58AG SCROLL/$10. Scrolling demo with source code.

990044/AB58AG SHARKS/$20. Sharks, Towers, Kalah, Mill.

990054/AB58AG SHOOT/$20. Shoot, 15 piece puzzle, Hyper-space war game, Seawar.

GH947/AD25AG SNAKES & LADDERS/$25. Board game from Australia with sound. Lemonade stand, Time signature.

JH903/BB25AG SOLITAIRE/$15. Standard solitaire.

JP940/AD25AG SPACE ARCADE I/$29.50. Three fast-action games: Galaxy attack, Invaders, Galactic patrol.

JP941/AD58AG SPACE ARCADE II/$29.50. Meteor mission, Orbits, Star battle.

AUG931/CD25AG SPACE WAR/$10. Good graphics.

GH946/AD25AG STAR FIGHTER/$20. Games with sound, plus word puzzle.

BJ920/BD25AG STAR TEC/$19.95. Star trek, Mastermind, Bounce, and x-y graph plot demo.

9048/AB58AG STAR TRADER/$20. Star trader, Color hunt, Decision maker, Calendar, Concentration.

990050/AB58AG SWARMS/$20. Swarms, Human reaction time, Roulette, Reverse numbers, Captain Alien.

SOFTWARE REVIEW Joseph Norris

The primary source of software for the Compucolor/Intecolor line of computers is Intelligen Computer Systems, in Huntsville, Alabama, whose catalog is printed in this issue. In addition, there are several other sources of programs, source code, and programs in ROM, some sources with only a few offerings. This summary will treat these additions to the software library. It is important, too, to remember the disk libraries of the several user groups. Catalogs of these holding are available to members.

FASBAS. $25.00. Peter Hiner, 12 Pennycroft, Harpenden, Herts AL5 2PD, England. This is the one and only Basic Compiler for the CCII and 3651 (all versions.) At this time, all Basic commands are compilable. Peter is contantly making improvements, which are available to previous buyers free of charge if they send back the disk (or send $5 for a new disk.) A new compiler, called ZIP, is in preparation which Peter says makes Basic color graphics nearly as fast as assembler graphics. We'll give more details when ZIP is ready. The price of FASBAS is mine, not Peter's. It is approaching the obscene to pay him less for all his incredible effort. Don't you agree?

COLOR GRAPHICS DISKETTE. $20.00. Joseph Charles, PO Box 750, Hilton, NY 14468. This diskette contains almost all the demonstration programs listed in David Suit's book on color graphics for the CCII/3651 computers. It will save many hours of typing. When ordering, please specify the software version desired.

COLORWORD, word processor, screen editor. $50.00. (Shipment by airmail included.) Program Package Installers, PO Box 37, Darlington, Western Australia 6070. For CCII (all versions), and 3651 computers. Assembly program with 20K buffer for any keyboard. Will process existing SRC files (or any file extension representing an ASCII text file), with conventional word processing facilities: word wrap; block copy, move and delete; string search with optional replace; upper or lower case character set accommodated; HELP facility; printer parameter setup; automatic repeat on keystrokes; imbedding control characters for printer parameters; screen preview of printout; compact FCS file storage; all FCS commands available.

COM-TRONICS. Com-Tronics software is available from Inteligent Computer Systems. There is no current activity for CCII from Com-Tronics, and they have not answered my mail. Fortunately, little software support would be required. Com-Tronics programs, by and large, work very well on all software versions. CTE and CTA are still my standards.

GARY DINSMORE. Selected software available. See advertisement in last Colorcue. Gary is still interested in establishing an independent software marketing arm for CCII. Write to him for details.

BILL GREENE. Bill has current releases of Compucalc, IDA, some games, and a (free) version of FORTH for the CCII. His software is available through Intelligent Computer Systems. FORTH may be obtained by writing to Bill (or to Colorcue for a version of FORTH for 3651. Colorcue will also assist in providing 3651 versions of any other Greene software.)

JIM HELMS. Jim has announced that he will not support his software for the CCII/3651 beyond October of 1984. The source code for his software is therefore for sale at a cost equal to 1.5 times the selling price of the assembled version. Source code may be ordered from Jim Helms, and purchasers must agree not to distribute such purchased code. User groups may purchase source code for their libraries, with distribution regulated by the same rules that apply to disk library holdings. Software compatibility to 3651 computers has been a problem for Helm software. If versions dated 1983 or later are used, they may be assumed compatible with 3651. Otherwise, Jim cannot guarantee results.

Helms software is available from Jim Helms or from any of these authorized dealers: Intelligent Computer Systems, 12117 Comanche Trail, Huntsville, Alabama 35803; Howard Rosen, PO Box 434, Huntingdon Valley, PA 19006; Program Package Installers, PO Box 37, Darlington 6070, Australia. Some price reductions have been made recently. Note that many programs are available in ROM for use at 4000H.

Jim Helms. 1121 Warbler, Kerrville, TX 78028. (512-895-5136)

METIER. This software firm specializes in programs for educational use in schools. They write software to specification in Basic for a variety of computers, including the CCII and 3651. Most software may be altered to suit specific needs. Purchaser of programs must agree to the sale contract prohibiting copying or redistributing the materials. Some currently-available programs are listed here:

Database - Names, Addresses, Phone numbers (home use) $27.95

Programs for Trigonometry, $29.95

Names, Addresses, Student Data (for teachers), $27.95. (Note: the above programs may be purchased by Colorcue subscribers for $15.00 for a limited time.)

Budgets, Requisitions and Cash Flow (useable with Title IV grants), $700.00.

Please write to METIER for a complete description of this software and other services.

METIER. PO Box 51204, San Jose, CA 95151.

PROGRAM PACKAGE INSTALLERS. This Australian firm is offering selected software on disk or in ROM (for 4000H):

Colorword, $55; General Ledger, $70; Disk Editor/Formatter, $20; Directory (for use with Database II), $20; WISE-

II (emulator), $25; Lister (Basic programs), $15; Printer Driver (for SRC files), $15; Screen Editor and Assembler, $30; Cross Reference Assembler Utility, $15; BASSCR, HEX/DIS, SOLITAIRE, and CRIBBAGE, $15 each; 'THE' Basic Editor (available soon.)

PPI will copy your program to ROM for $10.00.

Program Package Installers. PO BOX 37, Darlington 6070, Australia.

Compucolor/Intecolor User Groups

The active user groups still serving Compucolor and Intecolor owners are listed here. As usership declines, there tends to be a collaborative spirit among these groups, sharing library holdings as well as articles for publication. Should you join more than one? If you are hungry for every bit of available information, I think so, particularly CHIP, CompUKolour, and CUVIC. These publications will be particularly helpful in providing the little bit of hardware source material that might be available. There are borrowings from material in Colorcue, and you may expect this to continue. However, local activity is still in evidence, and this activity is sometimes only available in the local publication. We have done our best to gather complete information. Please write to the group of your interest for more details.

AUSTRALIA

CUVIC, Victorian Compucolor Intecolor User Group. Monthly newsletter, about six pages, with meeting notes, reviews, software and hardware articles. Recent article titles have been on Comp-U-Writer techniques, reading and writing FCS files to Comp-U-Writer (including SRC and TXT files!), a Computerized Star Map, and conversion of disk drives for use with the CCII. I find CUVIC a good source of 'hard-to-find' material. Meeting attendance varies between 15 and 20. Total membership unknown. Subscription rate unknown, but probably about $30 for US readers. CUVIC maintains a disk library, $5 disk cost for copies of their holdings (plus postage for U.S.)

President: Ken Winder, 8 Brindy Crescent, East Doncaster, 3030, Victoria, Australia.
Secretary/Treasurer: Ted Stuckey, Box 420, Camberwell, 3124, Victoria, Australia.
Editor: Barry Holt, 19 Woodhouse Grove, Box Hill North, 3129, Victoria, Australia.
Address library requests to the Secretary.

CUWEST, Western Australia Compucolor Intecolor User Group, quarterly newletter, about 8 pages. Membership about 30. Articles in last issue included a discussion of Australia's VIDEOTEX system, and a reprint of Gary Dinsmore's article from Colorcue.
Secretary: John Newman, PO Box 37, Darlington, 6070, Western Australia. (Program Package Installers)

UPDATE, the publication of the Compucolor User Group of New South Wales. I have not seen this newsletter, but I suspect it serves a small membership.
Editor: Tony Lee. 52 Cowane Road, St. Ives 2075, New South Wales, Australia.

(Continued on back cover)

GREAT BRITAIN

CompUKolour, the magazine of the Compucolour User Group (UK), published quarterly, or whenever, about 20 pages. Heavy article content and valuable material. Recents topics have included 'Improving your power supply', 'Transistor replacements', 'Compiling Basic' (Peter Hiner), and 'CREF, a cross-reference generator.' Issue 5 was dated December 1983, Issue 6 was dated March 1984. Membership fee is fifteen pounds (UK) yearly. Number of members unknown. This publication carries some resemblance to FORUM, and seems like a good source of materials. The user group maintains a disk library, and is currently sharing holdings with CHIP, in Rochester.

Secretary: Peter Hiner, 11 Pennycroft, Harpenden, Herts AL5 2PD, England.

Librarian: Bill Donkin, 19 Harwood Avenue, Bromley, Kent BR1 3DX, England.

Treasurer/Editor: John Booth, 27 Romany Lane, Tilehurst, Reading RG3 6AP, England.

UNITED STATES

CHIP, publication of the Rochester Compucolor Intecolor User Group. This venerable body, the granddaddy of us all, continues after six years as the leader in contributionship to periodicals, disk libraries, and general sustaining activity. Membership is $10 per year, which includes a subscription to CHIP (quarterly if lucky), and access to their very large disk library. I don't know their current membership, but monthly meetings draw from ten to twenty local members on a regular basis. Membership is international, however, and extensive. Newsletters can be 20 pages long. While content has been primarily a catalog of the disk library recently, CHIP, historically, is a source of material on all subjects relating to the CCII, hardware as well as software. An investment in back issues is well made (forgive me, Ben!). When the CCII is about to give it's final sigh, the Rochester User Group will most likely be there to ease it's last days. A remarkable collection of talent and dedication to whom we all owe a great deal. The list of names is a Who's Who for all former and current CCII materials. CHIP is currently working to assist 8000 users and investigate conversion of their library to 8000 format. CHIP maintains a library for the 3650 computer series as well. Write to them for more details.

Membership: Gene Bailey, 28 Dogwood Glen, Rochester, NY 14625

Librarian: Doug Van Putte, 18 Cross Bow Drive, Rochester, NY 14624

Editor: Ben Barlow, 161 Brookside Drive, Rochester, NY 14618

STAN PRO, West Coast user group and newsletter. I wish I could be more specific about this organization but they are, frankly, ephemeral at best. I have seen several newsletters, and they appear to include useful, and often 'rare' materials, with articles by names familiar and unfamiliar (Rick Manizar, Carl Hennig). Stan Pro has been an Intecolor dealer for some time, selling and servicing CCII, 3650 and 8000 series computers. Communications between Colorcue and Stan Pro have been less than satisfactory. The best information I have is this: the newsletter is still published quarterly(?), membership is about $35 per year, back issues are available (more than $100 per set). Service facilities are still available. Mr. Pro wrote to us recently giving permission to Colorcue to reprint articles from past issues of his newsletter. We will be purchasing the back issues and reviewing them. Thre is a large disk library associated with this group, perhaps 1000 or so programs. I know nothing about their content or availability. Perhaps you will want to communicate with Stan and see what is available.

S. P. Electronic Systems, 5250 Van Nuys Blvd, Van Nuys, CA 91401.

8000 SERIES COMPUTER USER GROUP

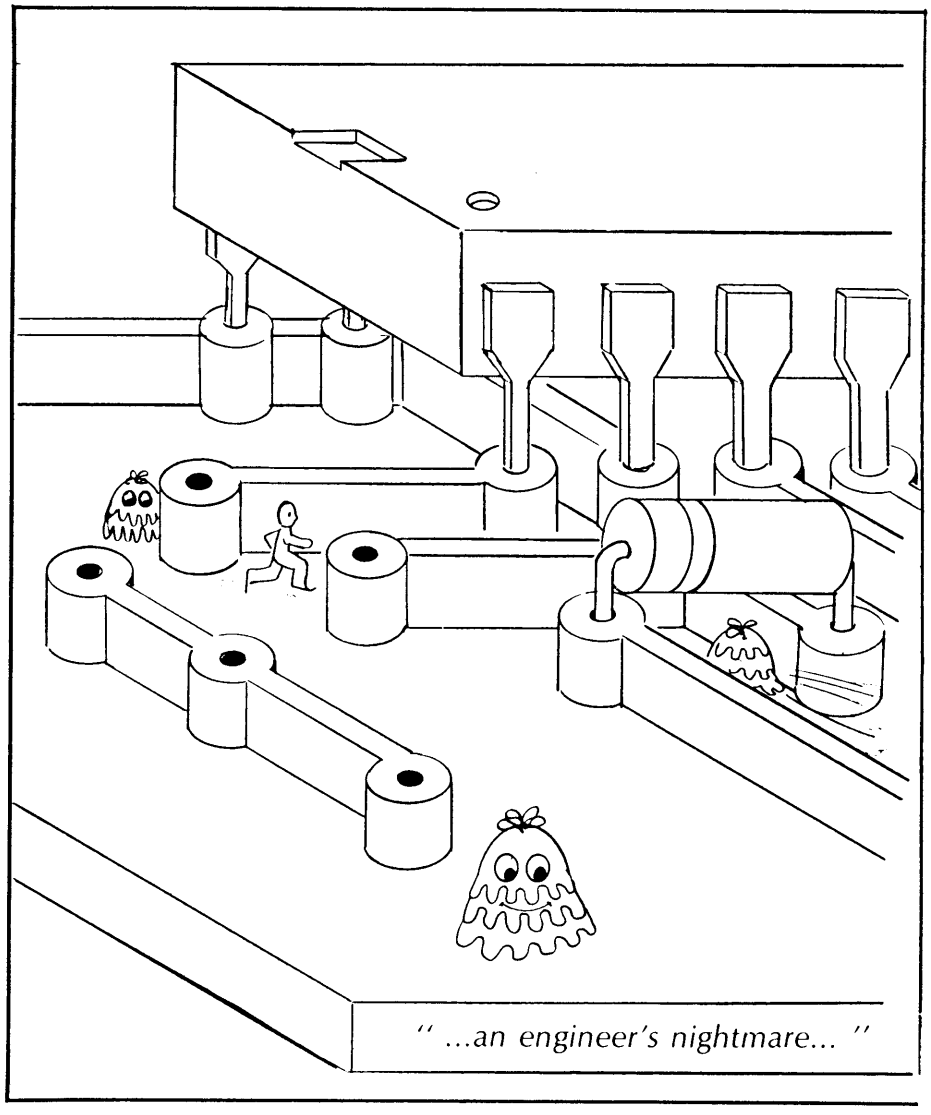
A relatively new user group for 8000 users is in the process of organization. I have had correspondence with several members and the outlook for interfacing with v6.78, v8.79 and v9.80-3 software looks promising. You may write to the 'founder', Glen Gallaway, at 1637 Forestdale Avenue, Beavercreek, OH 45432. Mr. Gallaway is in the process of moving and has a new address by this time, but perhaps his mail will be forwarded for awhile. Colorcue is interested in providing an outlet for articles, programs and source materials for the 8000. Please write if we can be of help. □

NEXT ISSUE: Last installment of Peter Hiner's series on Compiling Basic; W. S. Whilly masters the disk directory; Compucolor parts and service information; A switching box you can build; An 8000 article by Bob Mendelson....and more.

COLORCUE

A BI-MONTHLY PUBLICATION BY AND FOR INTECOLOR AND COMPUCOLOR USERS

VOLUME VI
NUMBER 4



W. S. Whilly on the rampage again....

ANIMATION

Compiling Basic

First 8000 Article!!!

Tiny-PASCAL

Calling Assembly Routines from BASIC

JULY/AUGUST 1984

Colorcue

VOLUME VI, NUMBER 4 JULY/AUGUST 1984

CONTENTS

COVER: "Nightmare" design by Jane Devlin.

BACK: W. S. Whilly's Directory Chart.

EDITOR: JOSEPH NORRIS COMPUSERVE: 71106, 1302

COLORCUE is published bi-monthly. Subscription rates are US$18/year in the U.S., Canada, and Mexico (via First Class mail), and US$30 elsewhere (via Air Mail). All editorial and subscription correspondence should be addressed to COLORCUE, 19 West Second Street, Moorestown, NJ 08057, USA. (609-234-8117) Every article in COLORCUE is checked for accuracy to the best of our ability but is not guaranteed to be error free.

"8000 SPOKEN HERE!!!!!"

This issue contains the first article to appear in Colorcue for the Intecolor 8000 Series Computers. It duplicates, to some extent, the material in W. S. Whilly's series on IDA, but is of a slightly different bent. I hope readers will forgive the similarity as we push to include 8000 users in our community. Many thanks to Bob Mendelson for taking the plunge! Colorcue has received ROM listings for the 8000 and will publish the more pertinent addresses in the next issue. The 8000 is very like the Compucolor and the 3651. The primary differences are in the 24K user RAM, an 80 by 48 screen display, and a totally different memory mapping. In general, there is no reason why CCII programs can not be converted for use on the 8000. The 8K RAM not present in the 8000 is occupied by Command files, normally external in the 3651 and CCII, and by printer and light pen routines. It is exciting to have this extension to our readership and I hope more articles will be forthcoming.

Several subscription renewals have been received at the office, well ahead of schedule. As you probably know, Colorcue has been accepting only full calendar year memberships, both to help us with planning on a six-issue basis, and because plans for next year have been uncertain. Please do not send in subscription renewals for next year until they are called for in Colorcue, that is, unless you owe more money to complete the 1984 edition. We will be making plans for next year during the remainder of the summer and announce them in the Sept/Oct issue.

As the Sourcebook materials continue, you will notice a dearth of material on hardware. PPI in Australia and Tom Devlin, in Michigan, are virtually the only vendors selling a selection of hardware materials for the CCII. Tom Devlin continues to offer his RAM card and Analog Protector circuit. Ben Barlow is offering a lower case character ROM (See previous issue). I do not know if Frepost Computers still exists.

Communications with them have been unanswered. I would like to repeat that any materials, software or hardware, that you want to add to your system, should be purchased very soon. It is not likely that they will be available much longer. The Rochester User Group remains the strongest source of software and overall inspiration. Join them to assure their continuation, and to make available to yourself a rich deposit of software in the free library. Colorcue is interested in having reviews of interesting programs in the CHIP library. This is a good time for you to write one on a CHIP program that has been interesting to you.

Since the publication of the ROM tables in Colorcue, we will no longer specify multiple ROM locations in articles, both to save editing time and to encourage you to use the tables. It is appropriate that our articles lean more heavily on Assembly Language programming than ever before. Assembly programming is the highest point in the Compucolor experience. It offers the greatest power. It is very enjoyable to work on, and it brings one as close as possible to the wonders of the Compucolor computer. Although 'getting started' articles have appeared in FORUM and Colorcue, there are readers who write that they are still puzzled and 'all at sea' with Assembly Language procedures. As I have noted previously, one almost has to wait for the 'light' to turn on. It will not turn on without your help, however. If you are among those still in the dark, we invite you to write to us for a personal tutorial in Assembly programming, tailored to your own special needs. Please state in your letter what you have already done, your analysis of your present problems, and what you think might be helpful. We will try to respond accordingly, even if you don't know where to start. It isn't difficult, truly, and the rewards are tremendous. You owe it to yourself to accept the challenge. □

John.

COMPILING BASIC

Part IV.

Peter Hiner
11 Penny Croft
Harpenden
Herts, AL5 2PD
ENGLAND

For this article, the last in this series, I am left with a long list of Basic commands not yet described and some other miscellaneous items. However, the majority of the remaining Basic commands warrant, at most, a brief comment of the "Did you know that...?" variety.

Did you know that once you have defined a function (as in DEF FNA(X)=X+Y-Z), you can subsequently use that function in defining further functions (as in DEF FNB(M)=FNA(1)+N)? This is rather like nesting subroutines, and you can add further levels of complication by using FNB in the definition of yet another function. Personally, I find even a single level of FN functions too complicated, but I came across nested functions while testing out my compiler on Startrek.

Did you know that numbers in a DATA statement can be read as strings? For example, DATA 1,2,3 could be followed by READ A,B$,C. If you subsequently PRINT A; B$; C, you will get a space in front of 1 and 3 but not in front of 2. I expect you knew that Basic puts the space in front of positive numerical values to make them line up neatly in columns with negative values (which have a minus sign instead of a space.) But did you know that Basic carefully avoids splitting a long number (such as 123456) between two lines on the screen? Before printing a numerical value, the interpreter creates a string of ASCII characters (in string manipulation space) and counts the number of characters. If the cursor is too near the end of a line, the interpreter inserts a carriage return before it starts printing out the number (which it now handles in the same way as a string.) However, the statement A$="123456" : PRINT A$, would not cause the interpreter to check the cursor position, and this string could be split between two lines.

Did you know that in a statement such as PRINT "DEAR ME";N$, you can omit the semicolon? I learned this the hard way when someone reported a bug in an early version of FASBAS.

Did you know that the statement PRINT SPC(X) does not cause a carriage return and line feed, even though it is not followed by a semicolon? The same applies to PRINT TAB(X). Unfortunately, I have only just had this pointed out to me, so there is still a bug lurking in my compiler, which can be avoided by putting a semicolon after these statements.

While on this subject, I should advise you of the other known bug. If you have a PRINT statement which contains a "cursor down" character, it will compile without problem, but, during the subsequent assembly operation, the "cursor down" character will look like an end-of-line marker, causing chaos and an error message. This problem can be overcome by replacing the "cursor down" character with

CHR$(10), although that will be marginally slower. To maximize speed, you could be brave and edit the intermediate version of a compiled program before assembly, but I doubt that you would notice the difference in speed.

To avoid having this article become a jumble of miscellaneous items, I will move on to the subject of memory space, beginning with a map to compare the allocation of blocks of memory for Basic and compiled programs. (See Fig 1.)

The manual supplied wih *FASBAS* describes the difficulty in defining (for either Basic or compiled programs) the limits of the "spare space" block, squeezed between blocks allocated forward from 829AH and blocks allocated backward from the end of memory. The "spare space" block will vary in size, both from program to program and during the course of running a program, so it is a dangerous practice to use it for machine code routines. The safe place for these is in space reserved at the end of memory, and this space can be reserved by starting with a line like this:

```
0 POKE 32940,a : POKE 32941,b : CLEAR 50
```

FIG 1. Relative memory allocations

ADDR	BASIC	COMPILED
8000H	--- FCS Parameters ---	
8200H	--- Basic parameters ---	
829AH	Basic program including DATA	Run-time library Data, Compiled program Variables & constants One dimensional numerical arrays
????	Variables and string pointers	String pointers
????	All types of array and File buffers	String arrays File buffers Multi-dimensional numerical arrays
????	--- Spare space ---	
????	--- Space for stack ---	
????	--- String space ---	
????	Space can be reserved here for machine code routines	
END OF RAM		

The values for a and b can be found from Fig 2. The POKE statements set a limit on the size of memory available to the interpreter, and the CLEAR statement defines the size of string manipulation space to be allocated (counting backward from the new limit of available memory.) The stack space is automatically moved further down in memory at the same time.

If you have a program which runs out of memory space when using FILE routines, you will have to consider reducing the size or the number of file buffers. The Compucolor Basic Manual gives all the information you need, but requires very careful reading to appreciate the implications of the decisions you may make concerning blocking factor and number of file buffers allocated.

The fast and easy way is to allocate enough buffers to enable the whole file to be stored in RAM at once, but I will assume, now, that you have run out of memory space. If you had allocated more than one buffer in the FILE "R" statement, you can reduce the number of buffers without any problem, except that the number of file accesses may be increased. If you had only allocated one buffer, you can use the option allowed by the FILE "R" statement to override the blocking factor. This can be done without changing the file itself, provided that everything (including your new blocking factor) fits exactly into a pattern of multiples of 128 bytes, or else you will be in trouble. If this suggestion leaves you confused, then do not attempt to implement it.[1]

If you are starting a new file, then you can get things right in the first place. The number of bytes per record multiplied by the blocking factor (number of records per block) determines the size of one file buffer (we can ignore the additional bytes used for housekeeping.) If this resulting number is not an exact multiple fof 128, then you will waste space both on the disk and in RAM.

Let us take as an example a file consisting of 256 records, each containing 32 bytes, and then consider the effects of varying the blocking factor and the number of file buffers allocated. To read the entire file at once from disk would require 8K of memory space, and it would make no difference whether we chose a blocking factor of 64 (and therefore allocated one file buffer of 8K bytes) or chose a blocking factor of 4 (and therefore allocated 64 file buffers of 128 bytes each). Other combinations between these extremes would also give the same result.

If, however, we could only afford 512 bytes of memory space for the buffers, then the maximum value of blocking factor we could use would be 16, and we would then allocate 1 buffer. This would cause 16 records to be read every time the disk is accessed, and would be the best arrangement (within this memory limitation) for sequential file access, or for most forms of random access.

To see why it is normally best to make the blocking factor as large as possible and to allocate only one file buffer, let us consider what would happen if, in the above example, we chose a blocking factor of 4 and allocated 4 file buffers of 128 bytes each. The first file access would cause 16 records to be read from disk (just as previously) and these

would fill all 4 file buffers. But any subsequent access to a part of the file not already in memory would cause only four more records to be read from disk. These four records would be put in the "least recently used" file buffer (overwriting the previous contents.) You can see that this is likely to result in nearly 4 times as many disk read operations, and will therefore be much slower. The only case in which you would benefit from multiple file buffers is when you want to retain part of the file (such as an index) in memory all of the time, and to achieve this you might have to insert dummy GET statements to assure that those buffers you wish to retain do not become the "least recently used."

FIG 2. Values for Reserved Memory Space.

Number of bytes to reserve	16K Memory (a)	16K Memory (b)	32K Memory (a)	32K Memory (b)
128	127	191	127	255
256	255	190	255	254
512	255	189	255	253

In one way or another, FILE statements gave me quite a lot of trouble while writing FASBAS. For a start, I had never been a heavy user of FILE statements in Basic programs, so I had to learn how they were meant to work in Basic before I could even comtemplate compiling them. I could see that FILE routines might spend a lot of time accessing the disk, which would not offer any chance for speed improvement. So I did not apply much effort to them initially, and I only included the minimum facilities in *FASBAS* v12.20. I still managed to get some bits wrong, and while correcting these bugs for v12.21, I decided to try to provide a complete implementation of all the FILE statements, including the obscure FILE "A" command. (Does anybody ever use it?)

The most difficult FILE statement was FILE "T", which provides error trapping. The theory is that if you include in your Basic program a statement like FILE "T",1000, the interpreter will jump to line 1000 instead of giving an error message, if at any time it finds an error while executing a FILE command. The trap facility is turned off again by declaring FILE "T", without a line number. The problem confronting me was that the error trap routine would try to find and interpret line 1000, and I could not make the interpreter give control back to the compiled program. The file routines are long and complex, containing many check points which might cause the program to jump into the error trap routine. So I could not put an alternative error trap routine in the run-time library unless I was prepared to rewrite all the file routines as well.

I came to the conclusion that I would have to include a bit of Basic program to satisfy the interpreter when an error was trapped. This bit of Basic program would contain an escape mechanism to give control back to the compiled program. Since the interpreter would start from address 829AH in its search for the required line, the compiled program would have to start with one or more lines of Basic before the machine code. The solution I eventually chose was related to the solution to another problem (chaining compiled programs), but I will try to keep them separate for the moment.

First of all, I made life a bit easier by determining that the error trap routine would always redirect the interpreter to the same line number (I chose line 2 for reasons which will be apparent later.) The compiled program would start with three lines (0, 1 and 2) of Basic, and line 2 would look like this:

```
2 POKE 33216,242 : POKE 33217,130 : PLOT 27,94
```

The POKE statements put the value 82F2H into memory at address 81C0H (which is the location of the jump vector for [ESC] [USER]. PLOT 27,94 is equivalent to keying in [ESC] [USER], and this will result in the program being vectored to 82F2H, which is the start of the routine for error trapping. So we have achieved the first part of our objective, by escaping from the interpreter routines to our own machine code routine.

The error trap routine contains an instruction JMPwxyz, where 'wxyz' represents an address which has previously been inserted during the execution of a FILE "T" statement (in our example the address inserted would be the location of the compiled version of line 1000.) So by a devious route, the program will eventually reach the right place.

Now we can look at the first two lines of Basic program, which are included to provide a mechanism to chaining compiled programs.

```
0 REM (followed by what appears as garbage)
1 POKE 33215,195 : POKE 33216,154 :
  POKE 33217,130 : PLOT 27,94
```

The POKE statements put the assembly language instruction JMP 829AH into the user escape location, and the PLOT 27,94 causes user escape to be activated. So if the Basic interpreter were to start reading what looks like the beginning of a nor-

Fig 3. Beginning compiled code lines.

Address	Object Code	Assembly	Meaning in Basic
829AH	A3H	ANA E	Address of next
829BH	82H	ADD D	line of Basic
829CH	00H	NOP	Basic line
829DH	00H	NOP	number 0
829EH	8EH	ADC M	REM
829FH	C3H,FDH,82H	JMP 82FDH	Rubbish
82A2H	00H	NOP	Basic end-of-line marker

mal Basic program, it would start at line 0, find a REM statement, and ignore the rubbish in the rest of the line. Then it would execute line 1 and the user escape function ould cause the processor to jump out of the interpreter routine and go back to address 829AH.

This time, the program would no longer be under the control of the interpreter and therefore the code starting at address 829AH would take on an entirely different meaning. To explain this, I have listed the address, object code and assembly language version of the first few bytes in Fig 3.

The code in 829AH to 829EH (which is really the Basic linking address, line number, and REM token) does nothing useful when taken to be machine code instructions, but it does no harm either. The JMP instruction then directs the program to the correct address after the rest of the Basic lines and the error trap routine.

Fig 4. Lines for pseudo-Basic program.

```
        ORG  829AH

        DB   0A3H,82H,0,0,8EH
        JMP  BEGIN
        DB   0,0CFH,82H,1,0
        DB   95H,'33215,195:'
        DB   95H,'33216,154:'
        DB   95H,'33217,130:'
        DB   92H,'27,94',0,0,0

BEGIN:  ; The rest of your program here.
```

Now we have a compiled program which can be run as a PRG type program under FCS control, or can be accessed through the Basic interpreter. So if we change the file type from PRG to BAS in the directory, we can load and run it just like a normal Basic program. We can freely chain together any combination of Basic and pseudo-Basic (compiled) programs using LOAD ....: RUN statements.

If you want to make your own assembly language programs look like Basic programs, you can include something like the Basic lines 0 and 1 at the beginning. (They must load to 829AH.) You can change the file type in the directory from PRG to BAS using the FCS RENAME instruction and then treat the program as if it were in Basic. One possible application of this would be to name a program as MENU, so that it can be run from the AUTO key.

For your convenience I give a listing in assembly language of the instructions you would need at the start of a pseudo-Basic program in Fig 4. This concludes my series of articles on compiling Basic. I hope you have found this ramble around the subject interesting and, in places, useful.

[FASBAS is available from the author for $25US. Ed]

1. See also COLORCUE, VOL VI, No 2, pps 26-28. Ed.

Assembly Language Programming

Part XV. Joseph Norris

Why are we so fascinated by animation? I wouldn't want it told that in spite of a very proper upbringing and a lifetime of rather sophisticated intellectual pursuits, I can be hooked for hours at the terminal trying to outwit a dumb figure that I know, perfectly well, is programmed to do me in, everytime. So, welcome to the human race!

This article has been requested more often than any other, yet you already have a good feeling for the construction of animation in assembly language. The fact that much of what we do here will seem obvious and elementary will be evidence of that. Since our purpose is to explore possibilities, rather than create a finished program, we will look at some techniques, carry them somewhat into a meaningful area, then cruelly leave you on your own, with your imagination and an inspirational screen display to use as a "springboard" for further play.

Animation on the CCII/3650/8000 computer can be achieved through the family of PLOT functions enabled in the system ROM, and, more satisfactorily, through the direct use of screen memory. We will begin by examining a brief example of animation using the PLOT functions.

In BASIC, we can construct and plot a rectangular "animaton" with a single line:

```
105 PLOT 3,30,15,2,110,111,3,30,16,2,100,109,255
```

This line sets the cursor at x = 30, y = 15; enters the character plot mode ("2") and prints the 1st and 4th quadrant "corner figures where they belong.

To add dignity to the rectangle, we can enter the blind cursor mode by preceding the x,y coordinates with a number higher than the largest valid x coordinate (greater than "63"). I have chosen "82" because it satisfies a "bug" in early software versions. (See the Instruction Manual for a description of the blind cursor mode.)

```
105 PLOT 3,82,30,15,2,110,111,3,82,30,16,2,100,109,255
```

If we change the values of x and y in line 105 then we can make the animaton "move" through the screen area. This is done easily in Basic by placing variables X and Y in the string, and replotting the string with changing values of X and Y:

```
105 PLOT 3,82,X,Y,2,110,111,3,82,X,Y+1,2,100,109,255
```

The illusion of motion of an animaton requires that we erase the previous position of the animaton before plotting its new location. To erase the rectangle, we can construct another line that is identical to Line 105, except that it will print "spaces" ("32") over the graphics characters, hence "erasing" them:

```
115 PLOT 3,82,X,Y,2,32,32,3,82,X,Y+1,2,32,32,255
```

animation

These two lines, executed in repeating succession, produce a "blinking" character plot on the screen. We may plant these same lines, virtually untouched, into an assembly routine as labelled DB strings. The only difference is the addition of "239" at the end of each string, so OSTR can be used to print them:

```
MAIN:  LXI   H,GRAPH  ;Point to string
       CALL  OSTR     ; and print it
       LXI   H,CLEAR  ;Point to erase string
       CALL  OSTR     ; and print it
       JMP   MAIN     ;Do it all again!

GRAPH: DB 3,82,30,15,2,110,111,3,83,30,16,2,100,109,239

CLEAR: DB 3,82,30,15,2,32,32,3,82,30,16,2,32,32,239
```

To change the plotting position of the rectangle, we need only alter that portion of the memory contents of these two strings which determine the x and y plotting values. For GRAPH:, these are the third and tenth bytes for the x values (GRAPH+2,GRAPH+9), and the fourth and eleventh bytes for the y values (GRAPH+3, GRAPH+10); similarly for CLEAR:.

Before we can experiment with this animaton, we need a skeletal program to perform the plotting and get some interfacing from the "joystick" or keypad. I call, again, on David Suits's keyboard input routine, from the June/July 1982 Colorcue. We need only those portions that will "get" a character press and place it in the accumulator for further processing. (Please refer to that article for explanation of this portion of the source code.) Specifically we will use the routines labelled "TEST", "GTCHA", and "CHRINT."

Our need for a skeletal program is filled by GRAPH.SRC (see Listing I). The comments are rather thorough, but I offer the following additional explanations:

Keyboard Assignments. I have chosen to use the numeric keypad for operator interfacing with the program, following the convention of "CHOMP", with "4" and "6" meaning "left" and "right", and "8" and "2" meaning "up" and "down." My joystick is connected to these keys. If you have a joystick assigned to the "arrow" keys, then make the appropriate conversions in GRAPH.SRC to accommodate them, by changing the CPI values in MAIN. If you have no joystick, then assign any keys that are comfortable for you. One such arrangement that works well is to use "N" and "M" for left and right, and "D" and "C" for up and down—using two hands for control.

The ten lines of code at MAIN control the proceedings. Their purpose is to intercept a keyboard input and branch to the appropriate subroutine for action. Key in GRAPH.SRC and assemble it to an appropriate origin. What you see when you RUN the program doesn't seem very exciting. A stilted rectangle moves inelegantly in four directions within the screen boundaries we have imposed. We will take some steps, however, to transform this into a rather interesting animaton.

Suppose we bypass the GTCHA routine, and cause the program to operate at full speed, maintaining its previous branching action until a new keypress is stored in KBCHAR. Change the source code at MAIN:

Replace MAIN: CALL GTCHA with MAIN: LDA KBCHAR.

With this change, the last keypress will remain in effect until a different keypress is stored in KBCHAR ("typematic" action, so to speak.) If you assemble this code you will now have a more challenging animaton on your hands, zipping from side to side and up to down , "out of control." But you have some important information in this demonstration. You now know how fast the PLOT structure can move things on the screen!

Let's slow things down a bit. Replace the single line at MAIN with these two lines:

```
MAIN:  CALL  WAIT     ;Pause a bit
       LDA   KBCHAR   ;Get last keypress
```

Add the WAIT subroutine in Listing 2. following the PRINT subroutine. This handy timer has a very large range of delays, and you will enjoy placing different numbers in the B register (by changing the source code and reassembling). You now have reasonable control of the rectangle. Notice that as the speed increases (as the value of B goes down), the lowly rectangle takes on a more interesting aspect. In a single additional step, we can create a "useful" game. We will paint the background blue, and permit the rectangle to trace a path through the screen as it moves. Change the DB string CLR to read as follows (adding a second line):

```
CLR:   DB    6,36,12,27,24,15,30,6,2,3,0,30,11
       DB    3,0,31,11,3,0,31,'SCORE: ',239
```

Plot 6,36 gives us a blue background (other codes will do that as well, of course) and we have "erased" the blue on lines 30 and 31 for future use as a scoring area. In mid-string, we have changed to green on black, so our rectangle will plot in those colors through the blue background we just layed out.

Now assemble and run GRAPH.PRG. Now see how "animated" our rectangle has become, "eating" its way around the screen under joystick or keypad control. Try to make a continuous path around the screen without intersecting any previous path; then test your skill at retracing it without going off the path (see FIG 1). (It isn't easy if B = 32 or less.) If you get tired, pressing the "fire" button (or any non-defined key) will stop the rectangle in its tracks. (We are touching on "DIG-DUG" territory.)

Making a useful "game" of this skeleton program isn't difficult. Your ideas will be better than mine, but here are some inspirational thoughts. Suppose we began the game with a highest possible score of 9999 and B = 16. A subroutine called at the beginning of MAIN decrements the score by one. Your goal is to gobble up all the accessible blue area before the score reaches 0. After displaying the final score of the first game, the program recycles with a still lower value in the B register, getting faster and faster each time it's played.

Another interesting set of refinements comes from a subroutine that can tell if the animaton is about to travel into a previously "erased" area or not. This permits scoring in a game that wants you to retrace a previously etched pathway. Such a subroutine can be derived by testing the CCI character in the plot blocks to be written to next.

With the ability to test plot blocks for their CCI content, we can also "plant" barriers to the animaton in a previously etched pathway, requiring a change of motion, tracing an alternate pathway, all with additional score reductions.

We will need a counting routine for the score as well. These are some things you can work on until next time, and we will then examine the use of direct access to screen memory as a technique for animation. □

Listing 1.

```
;GRAPH: AN ASSEMBLY ANIMATION PRIMER

;       JULY 25, 1984 JHN
;       Turn lower case off, caps lock on

;       Group Equates together here .....

;       Use your ROM tables to get v6.78
;       addresses. ISC 8000 users see end
;       of listing for instructions.

        OSTR    EQU     182AH   ;v8.79 &
        KBCHAR  EQU     81FEH   ; v9.80

;       Set origin for ESC T .............

        ORG     8200H

;SETUP..Set initial conditions ..........

BEGIN:  LXI     H,0     ;Clear HL
        DAD     SP      ;Add SP to HL
        SHLD    FCSSP   ;Store old SP
        LXI     SP,STACK ;Add new SP

        MVI     A,0C3H  ;Setup for CHRINT
        STA     81C5H   ; See David Suits
        LXI     H,CHRINT ; input routine
        SHLD    81C6H   ;  for details
        MVI     A,1FH   ;
        STA     81DFH   ;

        LXI     H,CLR   ;Clear screen etc
        CALL    OSTR

        JMP     PRINT   ;Draw initial box

;MAIN PROGRAM.....................

MAIN:   CALL    GTCHA   ;Get key press
        CPI     '6'     ;Move right?
        JZ      XINR    ;Jmp right routine
        CPI     '4'     ;Move Left?
        JZ      XDCR    ;Jmp left routine
        CPI     '8'     ;Move Up?
        JZ      YDCR    ;Jmp up routine
        CPI     '2'     ;Move Down?
        JZ      YINR    ;Jmp down routine
        JMP     MAIN    ;Invalid input

;SUBROUTINES.....................

CHRINT: PUSH    PSW     ;Suits input routine
        XRA     A       ; See his article
        STA     81FFH   ;
        POP     PSW     ;
        RET

GTCHA:  XRA     A       ;Get keyboard char.
        STA     KBCHAR  ; See David Suits
GTCH1:  LDA     KBCHAR  ;
        ORA     A       ;
        JZ      GTCH1   ;
        RET             ;

XINR:   LDA     GRAPH+2 ;Move right routine
        CPI     60      ;Right limit?
        JZ      MAIN    ;Yes. Don't move
        CALL    PREP    ;Erase old box
        LDA     GRAPH+2 ;get old x value
        INR     A       ; & increase by 1
        JMP     DOX     ;Store new x value
```

```
XDCR:   LDA     GRAPH+2 ;Move left routine
        CPI     2       ;Left limit?
        JZ      MAIN    ;Yes. Don't move
        CALL    PREP    ;Erase old box
        LDA     GRAPH+2 ;Get old x value
        DCR     A       ; & decrease by 1
        JMP     DOX     ;Store new x value

YINR:   LDA     GRAPH+3 ;Move down routine
        CPI     27      ;Lower limit?
        JZ      MAIN    ;Yes. Don't move
        CALL    PREP    ;Erase old box
        LDA     GRAPH+3 ;Get old y value
        INR     A       ;Increase by 1
        STA     GRAPH+3 ;Store new value
        STA     CLEAR+3 ; in two places
        LDA     GRAPH+10;Get old y1 value
        INR     A       ;raise by 1
        STA     GRAPH+10;Store it also
        STA     CLEAR+10; in two places
        JMP     PRINT   ;Draw new box

YDCR:   LDA     GRAPH+3 ;Move up routine
        CPI     1       ;Top limit?
        JZ      MAIN    ;Yes. Don't move
        CALL    PREP    ;Erase old box
        LDA     GRAPH+3 ;Get old y value
        DCR     A       ;Reduce it by 1
        STA     GRAPH+3 ;Store it in
        STA     CLEAR+3 ; two places
        LDA     GRAPH+10;Get old y1 value
        DCR     A       ;Decrease by 1
        STA     GRAPH+10;Store also in
        STA     CLEAR+10; two places then
        JMP     PRINT   ;Draw new box

PREP:   LXI     H,CLEAR ;Erase current box
        CALL    OSTR
        RET

DOX:    STA     GRAPH+2 ;Store new x
        STA     GRAPH+9 ; value in
        STA     CLEAR+2 ; these four
        STA     CLEAR+9 ; memory slots
        JMP     PRINT   ;Print new box

PRINT:  LXI     H,GRAPH ;Print box at
        CALL    OSTR    ; new x,y &
        JMP     MAIN    ; go back.


;STRINGS.....................................

;Setup string:  BG=BK, FG=GR, Erase page
                ;Page mode, A70ff, Flag On
```

```
CLR:    DB      6,2,12,27,24,15,30,239

;String to print animaton.....................

        ;       BC  x  y   Chars..

GRAPH:  DB      3,82,30,15,2,110,111

        ;       BC  x  y   Chars..

        DB      3,82,30,16,2,108,109,239

;String to erase animaton.....................

        ;       BC  x  y   Space

CLEAR:  DB      3,82,30,15,2,32,32

        ;       BC  x  y   Space

        DB      3,82,30,16,2,32,32,239

        ;       Storage for Stack Pointer


;Numerical Storage ..........................

FCSSP:  DW      1       ;FCS Stack Pointer

SCORE:  DS      2       ;Score storage

;Stack Allocation. This EQU entry means:
;       'Let the address STACK be 80H
;       bytes further on than this address'
;
;       We do this because the stack works
;       backwards toward 'SCORE' and we
;       want to allow enough room so it
;       won't write into 'SCORE's area.

        STACK   EQU     $+80H

        END     BEGIN

        ;Don't forget CR after 'BEGIN'

;INSTRUCTIONS FOR 8000 USERS:

;OSTR   EQU   0901H
;KBCHAR EQU   9FFEH

;       ORG   A000H
```

Listing 2. ;MAIN PROGRAM.....

```
MAIN:   CALL    WAIT    ;Delay subroutine
        LDA     KBCHAR  ;Use last key
        CPI     '6'     ;Move right?
        JZ      XINR    ;Jmp right routine
        CPI     '4'     ;Move Left?
        JZ      XDCR    ;Jmp left routine
        CPI     '8'     ;Move Up?
        JZ      YDCR    ;Jmp up routine
        CPI     '2'     ;Move Down?
        JZ      YINR    ;Jmp down routine
        JMP     MAIN    ;Invalid input
```

```
PRINT:  LXI     H,GRAPH ;Print box at
        CALL    OSTR    ; new x,y &
        JMP     MAIN    ; go back.

WAIT:   ;Delay counter subroutine - a
        ;counter within a counter. The
        ;number placed in B register
        ;determines how many times the
        ;C register will countdown. If
        ;you're really good, try '1' in
        ;B. If you're only mortal, start
        ;with 255 in B and work down
        ;until you're pleased with the
        ;results. This routine determines
        ;how long MAIN will wait before
        ;plotting a new animaton position.

        MVI     B,32    ;Do COUNT 255
WAIT1:  DCR     B       ; times.
        RZ              ;When B=0
        CALL    COUNT
        JMP     WAIT1

COUNT:  MVI     C,255   ;Count from 255
COUNT1: DCR     C       ; down to -
        RZ              ; 0 & return or
        JMP     COUNT1  ; keep counting
```

```
;STRINGS.....

;Setup string:  BG=BL, FG=BL, Erase page
                ;Page mode, A70ff, Flag On
                ;CLR erases page in blue

CLR:    DB      6,36,12,27,24,15,30

                ;BG=BK, FG=GR, Erase score
                ;lines & print title. Set
                ;color for box and Score
                ;as green on black (6,2)

        DB      6,2,3,0,30,11,3,0,31,11
        DB      3,0,31,'SCORE: ',239
```

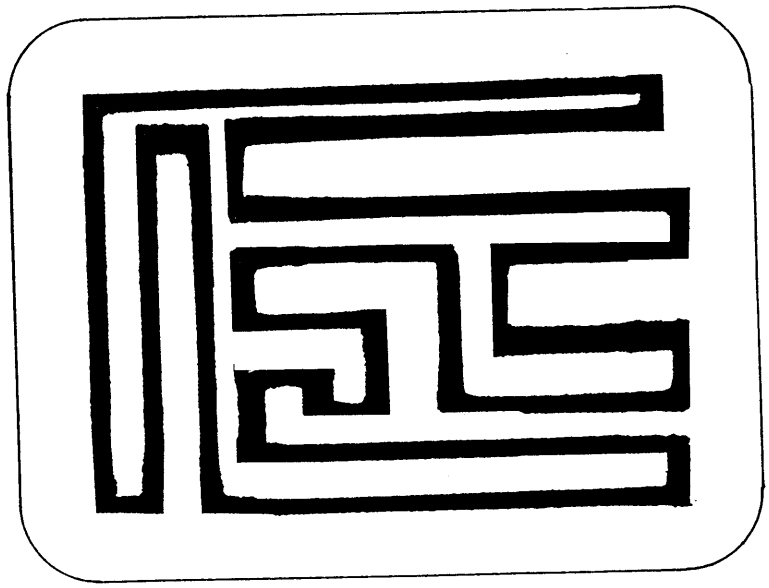


FIG. 1. Testing your skill with GRAPH

BOOKS FOR THE COMPUCOLOR II

Charles, Joseph. *Basic training for Compucolor computers.* Hilton, NY: Joseph J. Charles, 1980. 192 pps. (130 Sherwood Drive, PO Box 750 , Hilton, NY 14468. $14.95.

Dewey, Dale. *Advanced programmers manual.* Dale Dewey, $15. (Write Colorcue for details.)

Fernandez, Judi N. and Ashley, Ruth. *Introduction to 8080/8085 Assembly Language programming.* NY: John Wiley and Sons, Inc, 1981. 303 pps. $10.95.

Hogan, Brian. *Computing in color.* (Programmer's manual supplement. (Write Colorcue for details.)

Suits, David. *Color graphics for Intecolor 3651 & Compucolor II computers.* Hilton, NY: Joseph J. Charles, 1981. 152 pps. (130 Sherwood Drive, PO Box 750, Hilton, NY 14468.) $15.

Watson and Newman. *Programming colour graphics for Compucolor-Intecolor computers.* Program Package Installers, PO Box 37, Darlington, Western Australia 6070.

Compu-Free Colorware

Take the risk out of buying software. Compu-Free Colorware is user supported software. Send your disk in a mailer, and include return postage (the same amount it takes to send the disk), and I will return the disk with the program of your choice copied onto both sides. Alternately I can copy two separate programs, one on each side. Each program includes a manual on the disk in a .SRC file along with a simple minded Basic program to print it out. Use the program for several days. Try your data in it, see if it will do anything useful for you. If not erase the disk and all you are out is a couple bucks for the postage. If you like what you see, then we ask you to send a donation directly to the author of the program. The request will be spelled out in the Copyright message, along with the authors name and address. You are also encouraged to copy and share the program with your user group, and Compucolor friends. They would be under the same honor code to try the program and pay if they like it, or erase it if they don't.

Here is our current library:

Program Name:	Author:	Description:
Wordy (NEW!)	Dinsmore	Word Processor with dictionary. (Send two disks)
Book	Dinsmore	Programmed textbook teaching aid.
Intro. to Assembly	Dinsmore	Text for Book
Assebly Language	Dinsmore	Text for Book
Assembly Applicaton	Dinsmore	Text for Book
Pottery Making	Dinsmore	Text for Book
Survey	Dinsmore	Conduct your own opinion survey.
Stock Market	Dinsmore	Tracks price of stocks from newspaper
Budget	Dinsmore	Will handle a home budget. (32K)

Programmers: Let me register and distribute your programs through the Compu-Free Colorware system. You receive payment for your programs directly from the users. I handle duplicating and advertising. Send a SASE for complete instructions for submitting your work, and the standards to which your program and documentation should measure up to.

ORDER FORM: COMPU-FREE COLORWARE

Gary Dinsmore's
Creative Software
32695 Daisy Lane
Warren Or. 97053

Program name: _____ (one program copied both sides.
Second name: _____ (backside of disk)

Name: _____ [ ](place me on mailing list)
Address: _____
City: _____, State _____ Zip _____
Send a disk for each request. Send the return postage with request.

Notes On The CRT Controller Chip

Tom Devlin
3809 Airport Road
Waterford, MI 48095

Failure of the CRT controller chip in the CCII is a distressing problem. These parts are becoming very difficult to find. The history of this part in the Compucolor computer is a little complicated because three different parts have been used over the years.

The part first used was the SMC Microsystems CRT-5027. This has been, perhaps, the most widely-used CRT controller chip in the industry. It currently lists for $19.00 in the JDR Microsystems catalog and in the most recent Byte magazines. It is programmable for a variety of screen formats, and so it must be re-programmed each time the computer is powered-up. In the v6.78 systems, the data is stored in the PROM, UA1, to be read (as I/O of all things!) and written into the 5027 by the FCS routine from 3774H to 3794H.

At some time early in the v6.78 production, ISC had SMC mask a version of the 5027 for a 64*32 format specifically for the CCII. This version was given the part number CRT-5027-003, and it eliminated the need for PROM UA1, although socket space for UA1 was left on the logic board for quite some time after the change was made.

It is my understanding that the CRT-5027-003 device is now out of production. On v6.78 systems it is possible to replace it with a standard CRT-5027 and a copy of the original PROM. This PROM is an 82S123 (32*8) programmed as follows:

```
ADDRESS: 00 01 02 03 04 05 06 07-1F
   DATA: 35 97 D3 F9 60 30 F1  FF
```

(I can supply these PROMs at a cost of $20.00 each.)

When ICS went to v8.79, they moved the set-up parameters into the FCS ROM proper, and thus eliminated the need for UA1. Since they used the masked part, this data was never used, but its presence does make it easy to replace the masked (-003) part with one of the standard programmable versions.

The very last Compucolors built used a CRT-5048-3, instituted about the same time as the REV 4 logic board. It is probably not directly replaceable with the 5027. The 5048 is used on the 3651 so availability should not be a problem.

QTZ

```
100 REM  ** COMPUCOLOR II CHARACTER DISPLAY **
105 REM              by J. Ramsey
110 REM      ....Press <RETURN> to escape....
120       CLEAR : L=1 : C=2 : PLOT 12,3,64,0
130          N=28670 : FOR XX=0 TO 127
140          XX$=RIGHT$("  "+STR$(XX),4)
150             FOR I=1 TO 4 : N=N+2
160       POKE N,ASC(MID$(XX$,1,1)) : NEXT
170     N=N+4 : POKE N,XX : POKE N+1,C : C=C+1
180             IF C>7 THEN C=2
190     N=N+4 : POKE N,XX+128 : POKE N+129,C
200      POKE N+128, XX+128 : POKE N+129,C
210          C=C+1 : IF C>7 THEN C=2
220       L=L+1 : IF L>8 THEN L=1 : N=N+128
230             NEXT : POKE 33278,0
240          IF PEEK(33278)=0 THEN 240
250                    END
```

ADDENDUM

It's time for some special thanks: to Peter Hiner for four extraordinary articles on Basic, representing countless hours of work exploring, programming, writing and giving generously of himself; to Jane and Tom Devlin for performances far above the ordinary, and steady support of this magazine and its staff; to the faithful writers of this Volume - Doug Van Putte, W. S. Whilly, and Rick Taubold - who keep us in good company; and to all of you who continue to fill our office with such good materials.

We are still only hearing from a very few. There is room in our pages for much, much more. It's time for your article now! Contributions on animation are especially needed.

So far there have been no entries in the Colorcue Contest announced in the Mar/Apr issue. Does that mean I get to keep the prize money? You might try an entry, you know. If only one person enters..he wins!

We are sorry to lose Tom Andries as a user. He has endured much with a failing CCII. If you haven't tried Tom's Hour Glass graphic from Vol V, Jun/Jul, you're missing a remarkable bit of programming. Try compiling it with FASBAS for some extra pleasure. It is a simple and elegant use of CCII graphics capabilities.

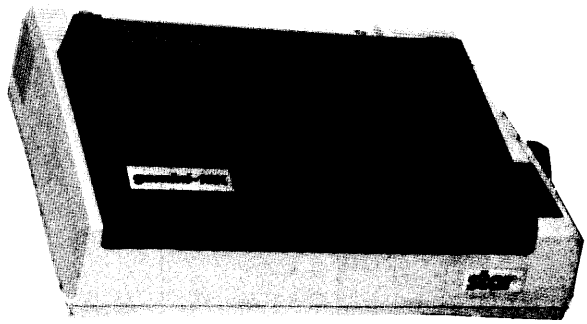
CUSTOM KEY CAPS FROM ARKAY

Arkay Engravers sells custom key caps and keyboard switches for the CCII and 3651. Colors and cap styles are an exact match, in both glossy and matte finishes. Front and side face engraving are available with up to two-color fill. Prices must be quoted to your specifications. This is a good way to expand to the full keyboard, and to customize caps for your favorite programs.

Arkay Engravers, Inc. 2073 Newbridge Road, PO Box 916, Bellmore, NY 11710. (516) 781-9343. Write for catalog.

Product Review -

The Gemini 10X Printer



David R. Ricketts
108 Joyce Avenue
Red Bank, TN 37415

It's Thursday of my one week of vacation, it's raining, you want articles, my CCII does not have lower case, my word processor is "COMP-U-WRITER 3.3, I have never written for a magazine before and I can't stand rejection. In spite of all these handicaps, here goes.

Spend $300 for a printer, me! the original tightwad, when this model 35 works great?? Oh! I gotta have a serial board too, even more money, a buffered serial board would be nice, even more money. Oh well, a friend of mine is selling the STAR, GEMINI line. "Take one home, try it out, pay me if you like it and etc.". I did, I did and I did.

I bought the Gemini-10X with the 4K buffered serial board and I have been using it for several months now and, if I can do so without sounding like a commercial, I'll try to share my experience.

DOCUMENTATION: The "USERS MANUAL" (packed in the box) although preliminary, is thorough, even to the point of illustrating the Removal of the Upper Case, Replacement of the Fuse and Replacement of Print Head. Also included are such things as: Parallel Interface Specifications, Connector Signals and Functional Description for Parallel Interface, Block Diagram, Code Chart & etc.

Although a "snow-job" at first, the instructions for set-up became clearer as I began to use them. My "preliminary" manual usage was short lived as my friend (the salesman) provided a much more thorough "USERS MANUAL" which provides sample programs for most of the popular computers (but not the CCII - that's okay 'cause we Compucolor users are accustomed to such). This manual utilizes illustrations liberally, is in an easy to read format, and it's 282 pages are a wealth of information, right down to the Glossary (pages 266 & 267) and the QUICK REFERENCE CHART on the inside back cover. In short, in a field where documentation is so scarce, many manufacturers (and software suppliers) could take a lesson from this book.

Did I mention that I did get the 4010X buffered Serial Interface? Well, it has it's very own USERS MANUAL, although very much like the "preliminary" it too is thorough and includes a schematic diagram, Interface instructions for several computers (not CCII). You must look at the specifications page to get general information on the EIA connection. Also included are complete step by step installation instructions for the serial board.

I also purchased a TECHNICAL MANUAL, but have had very little use for it as no problems have developed in the printer. It does appear to be another excellent publication, with plenty of illustrations, a fairly good schematic, complete parts list, lubrication instructions and much more. The cost of this manual was suprisingly low, compared to what we are accustomed to paying for maintenance manuals of any type.

INTERFACE: Thanks, in part, to Ben Barlow's article "The Serial Port" (COLORCUE, AUG/SEP 1981), The interfacing was not a big problem. I had to add the Handshake Modification to the CCII and determine which pin # to use going into the printer for handshaking, wire up a db-25 connector and plug it in. I must admit that this is the one area I used the printer's Technical Manual as it has a good explanation of the serial interface.

PRINTER FEATURES: Font Styles include Standard, Italic and eight international character sets. Font Pitches are Pica, Elite and Condensed (136 columns per line), double-width (5, 6 and 8.5 CPI). SOME MORE FEATURES: Double-strike, Emphasized, Underline, Superscript, Subscript, Unidirectional, pre-set linefeed to almost any value, Form Feed, Variable form length (# of lines or Inches), Variable Header location, Vertical tab, Horizontal Tab, Back Space, Graphics - (Normal, Double, Quadruple-density), Macro instruction, Downloadable characters (make your own), and more.

Since I was told that it is EPSON compatible, I gambled several hours of programming to assemble Martin P. Rex's Screen Dump program (FORUM, SEP/OCT 1982) and try out the graphics. Mr. Rex's program and the Gemini will reproduce any and every character you can put on the CCII screen (in black & white, of course - unless you use some other color ribbon).

Speaking of ribbons, there is nothing special or expensive about the ribbon used in the Gemini. Even though discouraged by the salesman, I have been using up my stock of Teletype ribbons. I am careful to look for deposits on them before I put one on the printer. Technically speaking, it is a Standard Underwood spool-type, 13x50mm.

The Gemini 10X handles tractor feed paper (fanfold) 3-10 inches, Roll paper 8.5-10 inches (5 inch Dia.) and single sheets 8-10 inches.

The Gemini does all the Ads say it will and does it very well. It is of course, Dot Matrix but even that is hard to notice with a good ribbon. I am well satisfied with it's performance to date. The only question remaining is that of reliability. I have been through several ribbons and probably 10000 sheets of paper without a problem. The only failure I have heard of was almost immediate and the printer was replaced when it was returned.

Telling the Gemini what to do is as easy as typeing "plot 27,52" (print in italics) or "plot 27,69" (print in emphasized mode). Gemini also reconizes some control codes i.e. 14(A7 on) causes the printer to print in enlarged mode for one- line only, 18 (green)- pica, 19(yellow) - takes printer "off-line, 17(red)- puts printer back "on-line" and etc.

Since I have had no other quality printer I cannot compare the Gemini, but In case you haven't noticed, I am as pleased with the Gemini as I am with the CCII. All I need now is a Word Processor that will take advantage of all the Gemini features.

A Deluxe Keyboard Aid

Steve Perrigo
16925 Inglewood Road NE
B-306
Bothell, WA 98011

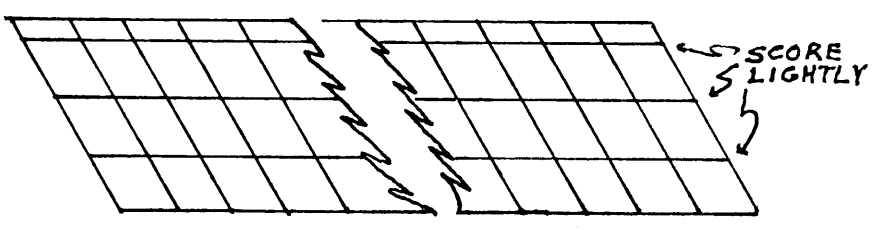
For those of you with the "deluxe" keyboard, the one with the 16 Function keys, this project is a 'must.' You have all probably wanted to use the Function keys for a program but haven't implemented them because of the difficulties of labelling them appropriately. If you have a word processor and screen editor that uses these keys the top of your keyboard can be cluttered with a lot of labels identifying the key functions. Here is a way to get rid of that clutter, and to give yourself some incentive to use the Function keys as they were intended to be used.

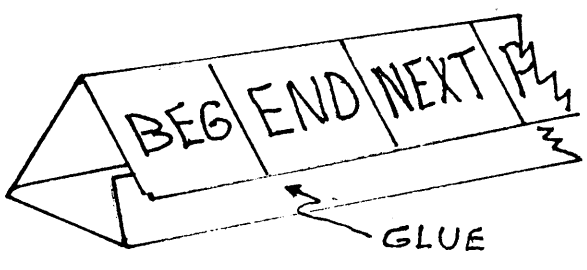
The idea is to fabricate a 'prism' that lies on the keyboard cover just above the row of Function keys, with a function description written on the prism just above each key. Each face of the prism holds the key codes for a different program, three programs for each prism. This method permits key labels large enough to read comfortably.

Such a prism may easily be made in several ways. Plastic supply houses often sell triangular solid stock. Have them cut a few 12-3/4" pieces for you. If that appears too expensive, you may want to search local drug stores for cheap plastic engineer's scales, triangular scales. These usually have finger grips cut horizontally which may be covered by stiff card stock to make a smooth surface. For any of the above, you can write or type the labels on stick-on label stock and fix them so they lay over the appropriate key cap. You may want to mark a vertical dividing line between each key cap position. Color coded label stock may be used for added emphasis. Some users color code their programs, white for the word processer side, green for the screen editor side, etc.

The prism may also be constructed from cardboard stock entirely. Using a sharp X-Acto type knife, cut the cardboard to 12-3/4" by 2-1/4". With a drawing pen, ink 16 vertical lines every 3/4" to make the divider between the seventeen keys on the Function key row (this includes the UP ARROW key.) Measuring from one long edge of the cardboard, ink three lines horizontally every 5/8" to mark the sides of the 'prism.' A 3/8" strip will be left at the other edge. This will become a glueing surface for fastening the cardboard into an enclosed triangle.

Use the knife to score the horizontal lines, being careful not to cut all the way through the cardboard. A light score will do. Write the appropriate key functions between the vertical lines (or fasten stick-on label stock previously prepared). Put a fine layer of glue on the 3/8" area, fold the cardboard into a triangular shape and slip the glued 3/8" surface under the open side of the triangle to close it. A narrow piece of wood and the table surface can be your clamps to hold the triangle closed until the glue sets. Instead of glue, you may be able to use double sided masking tape, if your cardboard is not too stiff. To seal the open ends of the 'prism' cut some triangular pieces from 3/16" balsa wood (available at any hobby store) and glue them in place. □





"ERLOZGRMT GSV
RMMVI HZMXGFN"

PeSTiCidAl

Our cliff-hanger in the last article was the prospect of forcing a directory entry for CYPHER.PRG, composed on IDA and written to the disk. Those of you who suspected this was pure braggadocio are in for a pleasant surprise. Let us first summarize where we were when we stopped.

Using an uninitialize disk, we used the WRIte command to write the code of CYPHER to the disk, beginning at block 0005. We then initialized the disk with five directory blocks. Next, through Basic, we created the file CYPHER.PRG, allowing enough space to include all the file code (this was done with a FILE "N" statement). When we tried to RUN CYPHER we received an error message because some critical parameters had not yet been entered into the directory record. Furthermore, we had an overlay table written to disk, beginning at block 000A, with no directory entry at all. (But you have created one as homework, by now.)

The directory doesn't much care who writes the information in it, whether it be operating system routines or you and me. It has a simple format, and as long as all the right data is in the right place, the directory will be effective. We need to study the directory a bit before we try to create one. I'm a great believer in creating "clean" baselines for this kind of work. To disect the mysteries of the directory, we can begin by taking a clean disk and formating it. Do this, and RUN IDAE or some other debugger. Reading the first 80H bytes into IDA at 8200H, a disassembly shows a pattern of "e's", or E5H. This is the deposit of my formatter. (Yours may be different.) The debugger instruction is:

```
IDA>XREA 00 8200-8280, or from FCS>REA 00 8200-8280
```

[I won't always cite procedures for the other instruments each time. Refer to the last issue or the instruction manuals for a reminder.]

In order to get a "cleaner" base line, let's write some 00H's to the first block of the formatted, uninitialized disk.

```
IDA>F 8200 8500 00
```

This will clear computer memory from 8200 to 8500.

```
IDA>XWRI 00 8200-8500, or from FCS>WRI 00 8200-8500
```

This will write the 00H's (NOPs) to the disk. Now initialize the newly-formatted disk:

```
IDA>XINI CD0:TESTDISK 05, or from FCS>INI CD0:TESTDISK 05
```

This will initialize the disk with five directory blocks.

Now we can copy CYPHER.PRG;02, from our very first work disk, to the newly initialized disk. If you have a single drive, then use the COPY.PRG software to make the transfer.

Activating IDAE again, we can now inspect the directory:

```
IDA>F 8200 8500 00; clear lots of computer memory to 00H.
IDA>XREA 00 8200-8280; read in the first directory block.
```

Within reasonable limits, you should see the contents of the directory as shown in Figure 1. The differences will be in the Free Space entry, because I am using an 8" double-sided disk which has considerably more space than the CD disk.

The first column of Fig. 1 shows the first 23 bytes of the directory. Most of them are not used for anything at all. The first byte is always 00H (NOP) for the first directory block. This byte states which directory block we are looking at, Block 0 being the first block, Block 1 being the second, and so on up to 04 for the fifth directory block. The second byte, labelled "Marker" in Fig 1, is always one less than the number of directory blocks specified in the INI command. Fig 1 shows I have five directory blocks on my disk (04 + 1). The third byte is always 41H, the ASCII letter "A", and is the attribute byte for the volume name. The next ten bytes, from 8203 to 820C contain the volume name. (You can put all kinds of things in there with IDA; colors, crazy characters, etc.) The remainder of the bytes, through 8216 are not used. They will be all 00H in our case because we wrote 00H to the disk. Otherwise they will either be E5H, on a newly formatted disk, or garbage left over from previous disk contents on a used disk.

Now begins a succession of file entries, the first three entries shown in the chart, each occupying 21 bytes.. The attribute byte for an unprotected file is 03H, always. Data involving two bytes, such as SBLK, SIZE, LADR, etc, are written low byte first. The contents, you will notice, follows the sequence that appears on the CRT directory listing. The "spare" byte, first appearing at 822B isn't used for anything, although numbers will sometimes be written there by the system software. I puzzled over this byte for a long time, failing to establish any correlation between its value and what was occuring with the disk file. I expected it to be a check sum for testing write integrity but I can't demonstrate that. (If you know something I don't know, please drop me a line.) A value of 00H is just fine for the "spare" byte, and has never failed to work for me with any kind of file.

Column three is the FREE SPACE entry for the directory.[1] Its attribute byte is 01H, always. There is no other meaningful data until SBLK, SIZE and LBC. LBC is always 80H for the FREE SPACE entry, so it's no problem determining that. But SBLK and SIZE should be accurate. Notice that SBLK is 000A, the first free disk byte following CYPHER, which confirms what we wrote to our uninitialized disk last time. (This was the beginning of the overlay for CYPHER's alternate encoding table.) The SIZE data in your directory will be a function of your drive type, CD drives having less free space than MD or FD drives.

PROGRAMMING!!!

Part Two

W. S. Whilly
(Wherever)

The last valid directory entry is always the free space entry, and it counts as one of the files in the directory. My directory blocks hold five files, which means I can store four "real" files and one FREE SPACE "file." You can always identify the FREE SPACE entry by the presence of the attribute byte (01H).

Column four is unused, and contains all 00H. Should I want to add another "real" file to the directory, I would add it beginning at the byte at 822CH, and move my FREE SPACE entry, now amended as to attribute, SBLK and SIZE, to the byte beginning at 8241H in Column 4 of Fig 1.

So let's go to work, and fix our last disk from the previous article. The directory should currently look like this (unless you really did create a file entry for the overlay):

FIG 1. MAP OF DISK DIRECTORY BLOCK

-----DISK VOLUME SPACE-----				-------1ST FILE ENTRY-------				-------2ND FILE ENTRY-------				-------3RD FILE ENTRY-------			
ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION
8200	0	?	NOP	8217	03	CTL C	Attribute	822C	01	CTL A	Attribute	8241	00	?	Attribute
01	04	CTL D	Marker	18	43	C		2D	00	?	(Free Space	42	00	?	
02	41	A	Attribute	19	59	Y		2E	00	?	Entry)	43	00	?	
03	54	T		1A	50	P	File	2F	00	?	File	44	00	?	File
04	45	E		1B	48	H	Name	8230	00	?	Name	45	00	?	Name
05	53	S		1C	45	E		31	00	?		46	00	?	
06	54	T	Volume	1D	52	R		32	00	?		47	00	?	
07	44	D	Name	1E	50	P	File	33	00	?	File	48	00	?	File
08	49	I		1F	52	R	Type	34	00	?	Type	49	00	?	Type
09	53	S		8220	47	G		35	00	?		4A	00	?	
0A	4B	K		21	01	1	Version	36	00	?	Version	4B	00	?	Version
0B	20	space		22	05		SBLK	37	0A		SBLK	4C	00	?	SBLK
0C	20	space		23	00		0005	38	00		000A	4D	00	?	
0D	00	?		24	05		SIZE	39	02		SIZE	4E	00	?	SIZE
0E	00	?		25	00		0005	3A	12		1202	4F	00	?	
0F	00	?		26	80		LBC	3B	80		LBC	8250	00	?	LBC
8210	00	?	Not	27	00		LADR	3C	00		LADR	51	00	?	LADR
11	00	?	Used	28	82		8200	3D	00			52	00	?	
12	00	?		29	00		SADR	3E	00		SADR	53	00	?	SADR
13	00	?		2A	82		8200	3F	00			54	00	?	
14	00	?		2B	00		Spare	8240	00		Spare	55	00	?	Spare
15	00	?													
16	00	?													

```
DIRECTORY DF0: TESTDISK 05

03 CYPHER.PRG;01 0005 0005 80 0001 0280
01 FREE SPACE)   000A 1202
```

Let's reinitialize this disk (no, it won't hurt the files we have on it!) and start from "scratch," making the directory conform to our needs. Go ahead! Have faith!

```
IDA)XINI CD0:TESTDISK 05
```

Now print the new directory on the screen and write down the SIZE of the FREE SPACE entry for future use:

```
IDA)XDIR ; (my FREE SPACE is 1207H blocks.)
```

Let's clear some computer memory,

```
IDA)F 8200 8500 00
```

and read in the first block of the new directory:

```
IDA)XREA 00 8200-8280
```

You may now do a disassembly or hex dump from 8200 to 8280, and fill in the values you find there in the first column of the chart in Fig 2 (from addresses 8200 to 8216 only).

Using the chart of Fig 2 as a worksheet, and working lightly in pencil at first, let's prepare to construct a directory for our disk, making entries for CYPHER.PRG and the overlay. Working now in column 2, the first directory entry, we enter the value 03H at address 8217, because the attribute for a "real" file is always 03H.

From 8218 we enter the file name, CYPHER, in hex, as 43H, 59H, 50H, 48H, 55H, 52H. Beginning at address 821E, we enter the file type, PRG, as 50H, 52H, 47H. At address 8221 we enter 01H for the version. (Are you doing this as we go?!)

At address 8222, we enter the SBLK (start block), low byte first, 05H, 00H (=0005H), and the SIZE of CYPHER (5 blocks, remember?) at address 8224; 05H, 00H (= 0005H).

We settled on a (L)ast (B)lock (C)ount of 80 for CYPHER, last time, even though that's not entirely correct; but we can use it. At address 8226, enter 80H. Now the fun begins. At address 8227, we enter the desired LADR (loading address) and it will be 8200, but low byte first: 00H, 82H. We can get fancy with the starting address. Remember that CYPHER begins with three NOPs that don't do anything. At address 8229, enter "8203", low byte first: 03H, 82H. Put a 00H in the "spare" byte slot (=00H) and we've done it!

But wait! We must still make a FREE SPACE entry. So move over to the next file column, and put the correct attribute byte at address 822C. What did you put there? A 01H, of course. 01H is the attribute byte for FREE SPACE. Now add the SBLK at address 8237, low byte first. 05H (for the directory) + 05H (for CYPHER) = 0A as the next block. (5 + 5 is A, in hex.)

The SIZE will be the free space you noted from the freshly initialized directory (1207 for me) minus the blocks we just assigned to CYPHER (=05H). So the FREE SPACE SIZE is, for me, 1207-0005 = 1202. I will put 02H, and 12H into my chart beginning at address 8239. Put 80H in the LBC slot at address 823B.

Let's put the values from the chart into memory. With IDA it's very easy:

```
IDA)P 8200
```

Move the cursor right or left with the arrow keys. As you enter hex numbers, the cursor will automatically move to the next slot on the right. When you reach the end of the line, the next set of addresses are automatically displayed. Begin at 8200 and keep on entering data, checking addresses and contents as you go, until you reach 823B, the LBC data for FREE SPACE.

Check it out with a hex dump from 8200 to 8240.

If it all checks out, write the directory you just made to disk:

```
IDA)XWRI 00 8200-8280
```

Display the directory:

```
IDA)XDIR
```

Run CYPHER from your new directory:

```
IDA)XRUN CYPHER
```

WOW! Now return to IDA and clear 8200-8500 with 00H again. Read the directory back in:

```
IDA)XREA 00 8200-8280
```

Using the Chart of Fig 2, erase the numbers in column three, the FREE SPACE column. We will now create a directory entry there for the overlay.

At address 822C, we must change the attribute from 01H to 03H. We can now enter the file name, TABLE1.OVR;01 in succeeding bytes. (That's this succession of hex numbers, my friends: 54, 41, 42, 4C, 45, 31, 4F, 56, 52, 01.)

SBLK is the same is it was, 000AH, low byte first. Enter at 8237H. Enter the SIZE as one block, 0001H. The LBC must be calculated somehow. LBC tells how many of the 128 disk block bytes are being used. It does not mean how many are left over! An LBC of 80H means all 128 bytes in the block are used. You will have 34H bytes or so in your overlay, depending on whether or not the last two space bytes were saved with the table. Put LBC at 823B.

The loading address is 8448H. That's where the table begins, and that's the point we saved it from (see last Colorcue). But what about the start address?

SADR is used by the system ROM to get a LDA or PRG program by placing this number in the program counter. ROM will not be looking for SADR in our file type OVR, which we just made up. We can make SADR 000H, then. We prevent SADR from causing trouble by LOADing the OVR, which simply puts the bytes in memory, instead of RUNning it, which would cause ROM to look for a starting address.

We need to put 00H into the "spare byte" slot at address 8240. Now it's your time to make the necessary FREE SPACE entry in column four, the third file entry. After you've done it, refer to [3] for my answers.

Write the new directory to disk, right over the old one:

```
IDA)XWRI 00 8200-8280
IDA)XDIR
IDA)LOAD CYPHER)PRG;01
IDA)LOAD TABLE1.OVR
IDA)G 8200
```

There it is, folks! You have broken into the inner sanctum!

But what happens when we look beyond the fifth directory entry? We will find two bytes preceeding the sixth entry. The first of these will be 01H, indicating that we are in the second directory block (of five.) The second byte will be 04H, the total number of directory blocks minus one. The following byte will be the attribute byte for the sixth directory entry.

It should be clear from this exercise that you now have a ready tool for reconstructing a clobbered disk directory....IF... you have a printout of the directory to work from. I periodically make a directory printout of all my important disks and store it in the disk sleeve. Not only is this a convenient way to view the disk contents, but it is a good way to have the data on hand for a reconstruction, ... and who hasn't needed a reconstruction at one time or another.

Even without a directory printout, IDA can search a clobbered disk, one block at a time, locate programs and create

FIG 2. DIRECTORY WORKSHEET

-----DISK VOLUME SPACE-----				-------1ST FILE ENTRY-------				-------2ND FILE ENTRY-------				-------3RD FILE ENTRY-------			
ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION
8200	0	2	NOP	8217			Attribute	822C			Attribute	8241			Attribute
01	04	CTL D	Marker	18				2D				42			
02	41	A	Attribute	19				2E				43			
03	54	T		1A			File	2F			File	44			File
04	45	E		1B			Name	8230			Name	45			Name
05	53	S		1C				31				46			
06	54	T	Volume	1D				32				47			
07	44	D	Name	1E			File	33			File	48			File
08	49	I		1F			Type	34			Type	49			Type
09	53	S		8220				35				4A			
0A	4B	K		21			Version	36			Version	4B			Version
0B	20	space		22			SBLK	37			SBLK	4C			SBLK
0C	20	space		23			0005	38			000A	4D			
0D	00	2		24			SIZE	39			SIZE	4E			SIZE
0E	00	2		25			0005	3A			1202	4F			
0F	00	2		26			LBC	3B			LBC	8250			LBC
8210	00	2	Not	27			LADR	3C			LADR	51			LADR
11	00	2	Used	28			8200	3D				52			
12	00	2		29			SADR	3E			SADR	53			SADR
13	00	2		2A			8200	3F				54			
14	00	2		2B			Spare	8240			Spare	55			Spare
15	00	2													
16	00	2													

new directory listings for them. If you know anything at all about your programs, the construction of their source code, their text, or whatever, IDA will assist in retrieving them from a destroyed disk. It would be helpful to make a "dry run" with CYPHER.PRG, by reinitiallizing the disk we have just created, and by means of a block to block search, get CYPHER back into computer memory and SAVE it.

A review of FCS SAVE command will also be useful.[2] It is often used to save memory contents as a PRG file. You will have noticed that when we use the FCS REAd or WRIte commands they take a similar format: for example:

```
IDA>XREA 00 8200-847D; Note dash between last two numbers.
```

This means "read beginning at block 00 into memory starting at address 8200, and continue reading until memory is filled to address 847D." The dash between 8200 and 847D indicates that both numbers are memory addresses. If the dash is omitted, the last number, 847D, indicates how many bytes are to be read—in this case far too many for the purpose. FCS allows you to specify either the last memory address to be filled (by using the dash) or the number of bytes to be read (without the dash.) This same convention applies to the SAVE command as well. The SAVE command also permits you to specify LADR and SADR, and this is useful for converting LDA files to PRG from IDA. The format is a little tricky. Here are some possibilities for a mythical LDA file, CYPHER.LDA:

```
Example 1 - SAVE CYPHER.PRG;01 8200-847D 8203
```

This tells FCS to save the code beginning at 8200 and ending at 847D to a disk file. LADR will be 8200 and SADR will be 8203.

```
Example 2 - SAVE CYPHER.PRG 8200-847D
```

We omitted the version number this time, so FCS will supply one for us. Since we omitted a specific SADR, FCS will make SADR the same as LADR, in this case 8200.

```
Example 3 - SAVE CYPHER.PRG 8200 0280 8203 9000
```

This is a most sophisticated instruction. We have changed the memory specification to show, not the end address in memory (847D in the first examples), but the number of bytes (0280H) to be saved. We have indicated SADR is to be 8203H. But the last number is telling FCS that the code we want to save isn't currently located in memory at 8200 at all. It is really in memory beginning at 9000, but we want it to read from disk to memory, henceforth, with LADR = 8200. FCS will write 0280H bytes, beginning at 9000 onto the disk, label it CYPHER.PRG and set LADR = 8200, and SADR = 8203. What use is this? Not much, in fact, and a "neater" way would be to use IDA to relocate the code where we actually wanted it to be in memory before a SAVE to disk as a PRG file.

Use Example 2 if LADR and SADR are to be the same. Use Example 1 if you want to specify an SADR not the same as LADR.

We have not exhausted the potential of IDA by a long shot, and we'll continue next time with the monitor discussion I promised for this time. (The editor won't give me any more room.) But one of IDA's most useful features is the reports it can generate to the printer. Any screen display, from the top of the screen to the current cursor position may be dumped to the printer by simply pressing CMD/PRINT. If you do not have the extended keyboard, this may be simulated by holding down at the same time the following three keys: SHIFT/CONTROL/V (cyan color key).

To set the port Baud rate, type from the IDA prompt some form of Bn(2), where n = 1 to 7, and the optional (2) adds two stop bits, example;

```
IDA>B7 ; for a Baud of 9600 and 1 stop bit.
```

IDA will also permit you to write on the CRT, in a simulated CRT mode, to make notes on the screen before you dump it! RUN IDAE, and XLOAD CYPHER.PRG. Disassemble from 8200 15+ to get a screen display. Now enter the simulated CRT mode by pressing the BREAK key, followed by CMD/CRT (COMMAND key and SHIFT/CRT all at the same time.) You may now use the cursor control keys to position the cursor anywhere on the screen, type your messages, then press ESC to return to the IDA prompt. CMD/PRINT will now dump the edited screen to the printer. Several of the printouts in my first article were constructed in this way. If you haven't ordered IDA yet, there's still time. Much more fun to come! W. S. Whilly. □

[1] If there were formerly a "real" file in this entry column on your disk, the file name and type may still be visable. The DELete command does not erase these parameters, but the attribute byte will be 01H, telling FCS that this is, indeed, the FREE SPACE entry.

[2] This information has been published previously by Jim Minor in *DATA CHIP*, -29, Dec/Jan 1982. Jim has a thorough presentation here of the REA, WRI, SAVE and LOAD commands that has not been published elsewhere. I highly recommend this article as a clear and thoughtful presentation of this material.

[3] SBLK = 000B; SIZE = 1202-1 = 1201; LBC = 80.[172]

FIG 3. Hex Dump of Directory with CYPHER.PRG as the only file. Note FREE SPACE entry.

```
IDA>H 8200 8280

8200  00 04 41 54 45 53 54 44  49 53 4B 20 20 00 00 00
8210  00 00 00 00 00 00 00 03  43 59 50 48 45 52 50 52
8220  47 01 05 00 05 00 80 01  00 80 02 01 01 00 00 00
8230  00 00 00 00 00 00 00 0A  00 02 12 80 00 00 00 00
8240  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
8250  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
8260  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
8270  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
8280  00
```

ITEM	PRICE

COMPUTERS:

Intelligent Systems Corporation:

Model 3651, 32K RAM, 117 key keyboard, lower case ..... ask for price

Morrow Micro Decision:

MD2, CP/M computer system, 2 diskdrives, single sided, 190 K, with Wordstar wordprocessor ..... $ 1,490

MD3, CP/M computer system with 2 diskdrives each 386K, Liberty monitor, with complete set of business software ..... $ 1,845

MD11, complete 11 mbytes hard disk computer system, high resolution graphics monitor, complete set of business software, ... $ 2,645

NEC APC

True 16bit CP/M86 (or MS.DOS) computer system, with high resolution graphics monitor, 2 8" diskdrives each 1 mbyte, ... $ 2,795

USED CCII 32K incl. software ..... $ 500
USED KAYPRO II incl. software ..... $ 1,300

PERIPHERALS AND OPTIONS:

Bell kit and simple soundware kit for 3651	.....	$ 25
CCII RS232 CTS kit "handshake"	.....	2
Lower case character kit, switchable	.....	38
Joysticks with instruction manual	.....	33
Bank board 56 K EPROM, software selectable	.....	286
Disk drive 5 1/4" for 3651 cable included	.....	350
Disk drive 5 1/4" for CCII V6.78 cable incl.	.....	250
Disk drive 5 1/4" for CCII V8.79 cable incl.	.....	250
Keyboard upgrade kit for CCII, 72 keys to 117 keys	.....	150
Keyborad upgrade kit for 3651, 72 keys to 117 keys	.....	250
Wordprocessor keycaps	.....	31

PRINTERS:

Gemini 10X dot matrix printer	.....	359
Gemini 15X dot matrix printer	.....	495
RS232 Serial interface board	.....	55
Brother HR-15 daisy wheel printer with sheetfeeder	.....	852
Cable for printer to computer	.....	25

**** all items subject to availability ****

VISA, MASTER CHARGE AND AMERICAN EXPRESS ACCEPTED

ICS INTELLIGENT COMPUTER SYSTEMS INC.

Disk Salvage

Bob Mendelson
27 Somerset Place
Murray Hill, NJ 07974

[Because of differing ROM calls and memory mapping, this program is not suitable for the CCII. Refer to W.S. Whilly's article, this issue, for a suitable equivalent procedure for the CCII. ed.]

The unexpected happened. I intended to initialize a new disk in drive #1 but I forgot to type in the '1', and lo and behold I had reinitialized a utility disk with 30 programs. 'DIR' only printed out an empty disk. I knew that INI does not wipe out the disk in the same way that formatting does, but I had only a vague idea of how the Directory is constructed. To explore this, the first 9 sectors of the directory were loaded into memory at A000H by use of the REA command. From here on it was easy.

The first 16 bytes are used for the ID of Sector 0, the name of the disk, followed by 10 'don't care' bytes. It was also apparent that only the first 80H bytes of Sector 0 are cleared to 00H. Everything else was unchanged; that is, the rest of the directory entries, the final line that shows the sectors used, the number left, and the delimiter, 80H (LBC), and all the program code.

My first try was to type in the data for the first 5 programs by use of the DIR printout that had been made for my library reference. This was done with the CPU monitor and an ASCII table for letters. It was primitive but not difficult. Each line of the directory uses 15H bytes, the last one of which is a 'don't care' spare byte. Therefore, new lines start at addresses A017, A02C, A041, A056, A06B... ending at A07F. A080 has a two-byte ID for Sector 1, which is followed by another set of 15H data blocks. Following the last line of active directory entries, there is an 01H, followed by 10 bytes, each 00H, and then the calculated number of sectors used, the number remaining, and an 80H delimiter.

The program in Listing 1. was written to allow simple reentry of the first five lines of the directory. Following reentry, the program will then write the data onto the disk and call for a directory printout. Should the directory have four or less lines, the last line will have been wiped out. Therefore, after typing the last data line, an 01H at the start of the following line will automatically calculate the used and free sectors and write them to the disk.

To save publishing space for the Listing, I have omitted the program instructions from the SRC code. They are printed here instead:

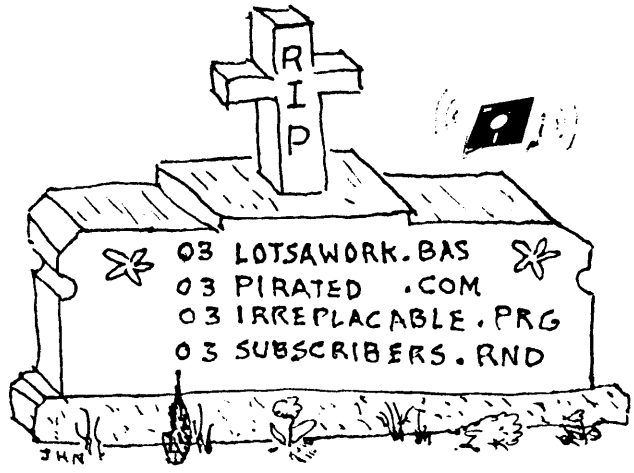
INSTRUCTIONS FOR UNLOCK.PRG -

THIS PROGRAM WILL RECOVER A DISK THAT WAS INITIALIZED BY MISTAKE IF A PREVIOUS PRINTOUT OF THE DISK DIRECTORY IS AVAILABLE CONTAINING THE ORIGINAL DIRECTORY INFORMATION.

TO OPERATE THIS PROGRAM, ENTER THE COMPLETE DATA FROM EACH LINE AS IT APPEARS ON THE PRINTOUT. DO NOT PRESS 'RETURN' UNTIL THE ENTIRE LINE IS ENTERED. IF NO PRINTOUT IS AVAILABLE, USE THE 'REA' COMMAND AND ESC P TO DETERMINE THE START OF EACH PROGRAM BLOCK AND THE PROGRAM SIZE.

THE LAST BLOCK COUNT (LBC) MAY BE 80H IF IN DOUBT. IF LADR (LOAD ADDRESS) AND SADR (START ADDRESS) ARE NOT KNOWN, CHOOSE ONE AND LATER CHANGE TO THE CORRECT ADDRESS. ONLY ALPHABETICAL CHARACTERS MAY BE EDITED DURING DATA ENTRY UNDER CURSOR CONTROL. DO NOT ATTEMPT TO CHANGE BEYOND THE ';' FOLLOWING THE FILE TYPE. NUMBERS ARE IN HEX, AND HANDLED IN THE SAME WAY THAT THE MONITOR HANDLES THEM; THAT IS, ONLY THE LAST FOUR HEX DIGITS WILL BE ACCEPTED BY THE PROGRAM. ENTER A SPACE TO SEPARATE NUMERICAL ENTRIES ON EACH LINE.

IF THE TOTAL NUMBER OF LINES IS FOUR OR LESS, TYPE '01' FOR 'ATR' TO END THE INPUT AND HAVE THE PROGRAM AUTOMATICALLY FILL IN THE 'FREE SPACE' DATA. IF FIVE LINES ARE TYPED IN, THE 'FREE SPACE' DATA WILL BE SUPPLIED WITHOUT MANUAL HELP. □



LISTING 1 Disk Salvage: UNLOCK
By R. Mendelson, V6-84

```
;A program to recover contents of a disk that was
; initialized in error. Only the first 80H bytes
; need to be restored to recover the directory.
; Remaining directory blocks & programs are intact.

;Equates.....................................

A017  BUFF    EQU     0A017H  ;Store DIR string
0103  CI      EQU     0103H   ;Console in
0109  CO      EQU     0109H   ;Console out
0100  CPUOS   EQU     0100H   ;CPU Monitor
000D  CR      EQU     13      ;Carriage return
0E74  CRLF    EQU     0E74H   ;Next line + GRN FG
00EF  EOS     EQU     239     ;'End of string'
0133  EXPR    EQU     0133H   ;Convert ASCII -> HEX

0012  GR      EQU     18      ;GREEN
000A  LF      EQU     10      ;Line feed
010F  LO      EQU     010FH   ;Character to CRT
9FFF  KEYBF1  EQU     9FFFH   ;KEYBOARD READY flag
012A  OSTR    EQU     012AH   ;Print string
```

```
0011  RED     EQU     17      ;RED
0020  SPACE   EQU     20      ;SPACE
0013  YEL     EQU     19      ;YELLOW
0F27  VECT    EQU     0F27H   ;Restart vector

;.............................................

;Note: Leading zeros may be omitted from hex inputs.

0000                  ORG     0B000H

B000 AF       START:  XRA     A
B001 32FF9F           STA     KEYBF1
B004 CD0301           CALL    CI
B007 2149B1           LXI     H,MSG1A ;Disk warning
B00A CD2A01           CALL    OSTR    ;Print it.

;Note: If 01 is hit before adding any line to the
; directory, Free Space SBLK will be set at 0009
; & SIZE at 0000.

B00D 3E09     PROTEK: MVI     A,9
B00F 3235B2           STA     SBLKX   ;Set SBLK=0009
B012 AF               XRA     A       ;Clear Acc
B013 3236B2           STA     SBLKX+1 ;Set SIZE to
B016 3237B2           STA     SBLK+2  ; 0000H and
B019 3238B2           STA     SBLK+3  ; store it.

B01C 2117A0   START1: LXI     H,BUFF  ;Load ptr
addr
B01F 2232B2           SHLD    ADDR1   ;Save it
B022 32FF9F           STA     KEYBF1  ;(A=0)
B025 CD0301           CALL    CI

B028 21C9B1           LXI     H,MSG2  ;Column headings
B02B CD2A01           CALL    OSTR    ;Print them

B02E 2117A0           LXI     H,BUFF
B031 AF               XRA     A
B032 0668             MVI     B,7FH-17H

;Set counter just past directory name and fill unused
; bytes with 00H....

B034 77       FILL:   MOV     M,A     ;Byte to memory
B035 23               INX     H       ;Index pointer
B036 05               DCR     B       ;Decr counter
B037 C234B0           JNZ     FILL    ;Fill next memory

B03A 21F9B1   READ1:  LXI     H,MSG3  ;Point to MSG
B03D CD2A01           CALL    OSTR    ;Print it
B040 3E05             MVI     A,5     ;Number of lines
B042 3234B2           STA     COUNT   ; to be typed in.

B045 3E20     SCREEN: MVI     A,SPACE ;Set CRT position
B047 CD0F01           CALL    LO
B04A 3E30             MVI     A,30H   ;Fake zero
B04C CD0F01           CALL    LO

B04F CDB1B0   ATR:    CALL    Y1      ;Input ATR
B052 FE31             CPI     31H     ;Is it 01H?
B054 CADEB0           JZ      FREE    ;Yes, set Free
Space
B057 D630             SUI     30H     ;No, ASCII to HEX
B059 2B               DCX     H       ;Back to byte #1
B05A 77               MOV     M,A     ;Insert hex #
B05B 23               INX     H       ;..for next char
B05C 3E20             MVI     A,SPACE ;Insert CRT space
B05E CD0F01           CALL    LO      ;Print it.

B061 CDBAB0   NAME:   CALL    Y2      ;Input file name
B064 3E07             MVI     A,7     ;Check if all
B066 B8               CMP     B       ; chars are in.
B067 C261B0           JNZ     NAME    ;No, go back.
B06A 3E2E             MVI     A,'.'   ;Add TYP delimit
B06C CD0F01           CALL    LO

B06F CDBAB0   TYPE:   CALL    Y2      ;Get file TYP
B072 3E0A             MVI     A,10    ;3 more bytes in?
B074 B8               CMP     B
B075 C26FB0           JNZ     TYPE    ;No, go back
B078 3E3B             MVI     A,';'   ;Add TYP delimit
B07A CD0F01           CALL    LO

B07D CD2EB1   VERS:   CALL    HEXNU1  ;Get version in.
B080 2B               DCX     H       ;Overlay MSB w/LSB

B081 CD2EB1   SBLK:   CALL    HEXNU1  ;Get SBLK
B084 EB               XCHG            ;From DE to HL
B085 2235B2           SHLD    SBLKX   ;Save it.
B088 EB               XCHG            ;Back to DE

B089 CD2EB1   SIZE:   CALL    HEXNU1  ;Get file soze
B08C EB               XCHG
B08D 2237B2           SHLD    SIZEX
B090 EB               XCHG

B091 CD2EB1   LBC:    CALL    HEXNU1  ;Get LBC
B094 2B               DCX     H       ;Overlay MSB of LBC

B095 CD2EB1   LADR:   CALL    HEXNU1  ;Get loading addr

B098 CD2EB1   SADR:   CALL    HEXNU1  ;Get SADR from
keybd
B09B 23               INX     H       ;Pass spare byte
B09C 2232B2           SHLD    ADDR1   ;Save ptr addr
B09F 3A34B2           LDA     COUNT   ;Get prev line cnt
B0A2 3D               DCR     A
B0A3 FE00             CPI     0       ;See note below
B0A5 CA1FB1           JZ      FREE2
```

```
;Note: If 5 lines are being typed, stop before adding a
; 5th line 'Free Space' line, since the rest of the
; directory is still intact.

B0A8 3234B2          STA     COUNT
B0AB CD740E          CALL    CRLF    ;For next dir line.
B0AE C345B0          JMP     SCREEN  ;Start next line.

;Input handler.....................................

B0B1 2A32B2  Y1:     LHLD    ADDR1   ;Current buff addr
B0B4 0600            MVI     B,0     ;Reset char counter
B0B6 AF              XRA     A       ;A=0
B0B7 32FF9F          STA     KEYBF1  ;Clear keybd flag
B0BA CD0301  Y2:     CALL    CI      ;Input character
B0BD FE1A            CPI     1AH     ;Back space?
B0BF CAC9B0          JZ      Y4      ;Yes, jump

;Input to buffer...................................

B0C2 77              MOV     M,A     ;Char into buffer
B0C3 23      Y3:     INX     H       ;Incr buffer addr
B0C4 2232B2          SHLD    ADDR1   ;Update buff ptr
B0C7 04              INR     B       ;Incr counter
B0C8 C9              RET             ;..for next char.


;Backspace routine.................................

B0C9 2B      Y4:     DCX     H       ;Back up buff ptr
B0CA 05              DCR     B       ; & start of line.
B0CB C2C9B0          JNZ     Y4      ;Do again!
B0CE 2232B2          SHLD    ADDR1   ;Update buff ptr
B0D1 3E0B            MVI     A,0BH   ;Erase line.
B0D3 CD0F01          CALL    LO
B0D6 3E0D            MVI     A,CR    ;Carriage return
B0D8 CD0F01          CALL    LO
B0DB C345B0          JMP     SCREEN  ;Start line over.


;Insert FREE SPACE entry if dir < 5 lines

B0DE D630    FREE:   SUI     30H     ;ASCII to HEX
B0E0 2B              DCX     H       ;Back 1 buff addr
B0E1 77              MOV     M,A     ;Number into buff
B0E2 23              INX     H
B0E3 AF              XRA     A       ;A=0
B0E4 060A            MVI     B,10    ;Set 10 blanks

B0E6 77      FREE1:  MOV     M,A     ;Insert zero
B0E7 23              INX     H
B0E8 05              DCR     B       ;Reduce counter
B0E9 C2E6B0          JNZ     FREE1   ;If B<>0

;Calculate next free block (SADR).....

B0EC 2232B2          SHLD    ADDR1   ;Save buffer addr
B0EF 2A35B2          LHLD    SBLKX   ;Get last SBLK
B0F2 EB              XCHG            ;Save it in DE
B0F3 2A37B2          LHLD    SIZEX   ;Get SIZE
B0F6 19              DAD     D       ;Add for SBLK
B0F7 EB              XCHG            ; and move to DE
B0F8 2A32B2          LHLD    ADDR1   ;Current buffer
addr
B0FB 73              MOV     M,E     ;Place LSB in
buffer
B0FC 23              INX     H
B0FD 72              MOV     M,D     ;Place MSB in
buffer
B0FE 23              INX     H

;Calculate blocks still free (SIZE)...
; DE still has SBLK -

B0FF 2232B2          SHLD    ADDR1   ;Save buffer addr
B102 7B              MOV     A,E     ;Make 1's
complement
B103 2F              CMA             ; of E register
B104 5F              MOV     E,A     ;Save it.
B105 7A              MOV     A,D     ;1's complement of
B106 2F              CMA             ; D register
B107 57              MOV     D,A     ; and save it too.
B108 EB              XCHG            ;Complement into HL
B109 1E01            MVI     E,1     ;Set DE=0001
B10B 1600            MVI     D,0
B10D 19              DAD     D       ;Convert to 2's comp
B10E EB              XCHG            ;Move it to DE
B10F 2602            MVI     H,02H   ;Total sector count
B111 2E76            MVI     L,76H   ; is 276H
B113 19              DAD     D       ;Difference in HL
B114 EB              XCHG            ;Move it to DE
B115 2A32B2          LHLD    ADDR1   ;Current buffer
addr
B118 73              MOV     M,E     ;Insert LSB
B119 23              INX     H
B11A 72              MOV     M,D     ; and MSB.
B11B 23              INX     H

;Terminate directory ..............

B11C 3E80            MVI     A,80H   ;Delimiter (LSB)
B11E 77              MOV     M,A     ; into buffer

B11F 210FB2  FREE2:  LXI     H,MSG4  ;Put dir on disk
B122 CD2A01          CALL    OSTR
B125 2125B2          LXI     H,MSG5  ;Print to CRT
B128 CD2A01          CALL    OSTR
B12B C3270F          JMP     VECT    ;Return to system
```

```
;VECT is restart vector RST1, and the technical end of
; the program.

;Subroutines ....................................

B12E 2232B2   HEXNU1: SHLD    ADDR1    ;Save buffer
addr
B131 CD42B1           CALL    HEXIN
B134 EB               XCHG             ;Move HL to DE
B135 2A32B2           LHLD    ADDR1
B138 73               MOV     M,E      ;Insert LSB
B139 23               INX     H
B13A 72               MOV     M,D      ; and MSB
B13B 23               INX     H

;Note: DE retains the latest value for
; use in SBLK & SIZE -

B13C 3E20     HEXNU2: MVI     A,20H    ;'space'
B13E CD0F01           CALL    LO       ;CRT display only
B141 C9               RET

B142 0E01     HEXIN:  MVI     C,1      ;4-byte ASCII to 2.
B144 CD3301           CALL    EXPR     ;Hex # in HL
B147 E1               POP     H
B148 C9               RET

;String storage.................................

B149 0C135052 MSG1A:  DB      12,YEL,'PROGRAM TO REPAIR'
B14D 4F475241
B151 4D20544F
B155 20524550
B159 414952
B15C 204C4F53         DB      ' LOST DIRECTORY-',GR
B160 54204449
B164 52454354
B168 4F52592D
B16C 12
B16D 03000242         DB      3,0,2,'BY R. MENDELSON',YEL
B171 5920522E
B175 204D454E
B179 44454C53
B17D 4F4E13
B180 03000450         DB      3,0,4,'PLACE ',RED,'DISK TO '
B184 4C414345
B188 20114449
B18C 534B2054
B190 4F20
B192 42452052         DB      'BE REPAIRED ',YEL,'IN DRIVE-'
B196 45504149
B19A 52454420
B19E 13494E20
B1A2 44524956
B1A6 452D

B1A8 03200412         DB      3,32,4,GR,'(HIT ANY KEY TO '
B1AC 20484954
B1B0 20414E59
B1B4 204B4559
B1B8 20544F20
B1BC 434F4E54         DB      'CONTINUE)',CR,LF,LF,EOS
B1C0 494E5545
B1C4 290D0A0A
B1C8 EF

B1C9 41545220 MSG2:   DB      'ATR NAME TYPE VR  SBLK  SIZE'
B1CD 4E414D45
B1D1 20545950
B1D5 45205652
B1D9 20205342
B1DD 4C4B2020
B1E1 53495A45
B1E5 20204C42         DB        LBC LADR  SADR',CR,LF,LF
B1E9 43204C41
B1F1 53414452
B1F5 0D0A0A
B1F8 EF               DB      EOS

B1F9 1B045245 MSG3:   DB      27,4,'REA0:0 A000-A47F',13
B1FD 41303A30
B201 20413030
B205 302D4134
B209 37460D
B20C 1B1BEF           DB      27,27,EOS

B20F 1B045752 MSG4:   DB      27,4,'WRI0:0 A000-A47F',13
B213 49303A30
B217 20413030
B21B 302D4134
B21F 37460D
B222 1B1BEF           DB      27,27,EOS

B225 0D0A121B MSG5:   DB      CR,LF,GR,27,4,'DIR0:',27,27
B229 04444952
B22D 303A1B1B
B231 EF               DB      EOS
;Data storage...................................

B232          ADDR1:  DS      2
B234          COUNT:  DS      1
B235          SBLKX:  DS      2
B237          SIZEX:  DS      2

B239                  END     START
```

How to Merge 'BASIC' Programs with Assembly Language Programs

Rick Taubold
197 Hollybrook Road
Rochester, NY 14623

by Rick Taubold (and Tom Devlin, who helped but wants none of the credit)

Let me ask all of you a question. How many of you have seen one of those programs which you could LIST in BASIC but which obviously contained more? Perhaps there was a CALL instruction but no machine language had been loaded either from disk or by POKEing. How many raised hands do I see? The purpose of this article is to clear up this little mystery. In the process you will learn new things about your Compucolor II. What is described here is not limited to the CCII, but will work on any computer that employs Microsoft or similar BASIC.

It was Tom Devlin, maker of nifty hardware for the Compucolor II, who first shared the secret with me. I should also point out that this can be used to merge any number of machine language subroutines with one BASIC program as well as permitting you to SAVE a machine language program as if it were a BASIC program. BASIC is a flexible language. Unfortunately, it is occasionally too slow for all desired uses. Writing entire assembly language programs might be fun to some people. To most of us it's a lot of work. Therefore, it is often desirable to write in BASIC and to add short machine language routines where speed is required. The CALL function in BASIC allows interfacing to machine language subroutines. Since it involves a subroutine, it must always end with the machine language equivalent of a RETURN instruction (RET in mnemonic code, hex value = C9, decimal value = 201). Other times it is convenient to write most of the program in machine language but to write an introduction or instructions in BASIC.

In this case there are several options. Usually the programmer will simply use BASIC to load and run his machine language program directly in which case there is no difficulty. An alternative is to load both programs at the same time and to employ ESC USER to execute the machine language. Again, the programs exist separately on the disk. I will present ways of having both BASIC and machine language programs in memory together but merged as a single program on the disk. Interested? Read on.

Before I continue, I would like to clear up a few misconceptions about the CALL and ESC USER functions of the CCII. Many users seem to think that ESC USER is limited to a single function. This is untrue. Both the CALL and ESC USER commands represent what is called a 'JUMP VECTOR'. By way of explanation let's consider an analogy in BASIC. Assume that we have a command like GOTO X or GOSUB X, where X could be a variable instead of a specific line number. Wouldn't the capability be nice? Our variable X would be a set variable names but we could change the value of X whenever we wished.

A JUMP VECTOR is similar. It means that we are telling the computer to jump to a particular fixed location. At that location is another jump instruction. The CALL command uses the locations 33282, 33283 and 33284. ESC USER employs the three locations 33215, 33216, 33217. The first location (33282 or 33215) always contains a machine language JMP instruction (C3 hex, 195 decimal) which is similar to the GOTO and GOSUB instructions in BASIC. This instruction is followed by a 2-byte memory address. Most of the time the programmer only uses one jump address. A typical program will assign only one CALL function. However, this is not a requirement. In my 'FINAL FRONTIER' program several different machine language routines are used, depending on the need at the time. Since most of the program was written in BASIC and only one CALL command available, the 'jump vector' must be POKEd with a new jump address each time a different machine language subroutine is required.

The CALL command in BASIC operates as a GOSUB to a machine language subroutine. When the machine language subroutine is completed, the program will RETURN to the next BASIC statement. When the BASIC sees a CALL command, it immediately jumps to memory location 33282 and sees another JMP command. (Keep in mind that ANY instruction could be placed here.) The BASIC reset routine normally insures that a JMP command is placed here, but I like to POKE in the command just in case it is somehow wiped out along the way. The computer will execute whatever it sees, so it pays to be certain which command is there.

The jump address is calculated in an unusual way (unusual only if you're not used to it). As an example, let us assume that the machine language subroutine to be CALLed is at hex location F000 (61440 decimal). From this we must calculate two values to POKE: F0 and 00. These work out to be, in decimal, 240 and 0 respectively. However, when machine language reads an address, it expects the two bytes IN REVERSE ORDER! Therefore, we POKE them in backwards, and our final POKE instruction line to set up this particular CALL would be:

```
POKE 33282,195:POKE 33283,0:POKE 33284,240
       (JMP)         (00)         (F0)
```

When we use X = CALL(0) from BASIC, our program will first jump to location 33282, see the JMP F000 instruction, and go to F000 hex to begin execution.

The ESC USER works the same way. The only difference is that ESC USER acts like a GOTO and recognizes no RETURN instruction. Again, the important thing to remember is that we are not restricted to a single jump location. Your program can alter these jump vectors at any time,

and this makes them extremely powerful. ESC USER can be executed directly from a BASIC program by PLOT 27,30.

There are several other places where jump vectors are used in the CCII. One of these is USER TIMER -2, and another is INPCRT. If anyone out there is interested, I can cover these in a future article. Now let us return to our main topic. You'll see the relevance of the previous discussion shortly.

I will demonstrate the merging procedure using the Scrolling Patch, a simple but useful illustration. In its original form the Scrolling Patch used a BASIC program of some 500 bytes to POKE in a 32 byte machine language program. Somehow, this seems like overkill. For many applications the programmer can enter this Patch directly and 'throw away' the BASIC part. New parameters can be POKEd easily. This method saves memory and cleans up a program. A more or less complete description of this Patch appeared in the double issue Nov/Dec-Jan/Feb of FORUM. The first step in the procedure is to write and test the assembly language program. I have already done this in Listing -2. When everything works, you are ready to merge the two. For this demonstration enter the BASIC program in Listing -1, exactly as written, and SAVE it on disk. Next, using a screen or text editor, enter the source code of Listing -2 and save it on disk also. You may omit the comments. I placed the scroll parameters in EQU statements so that you can readily change them for your purposes. Do not assemble the source code yet!

You should now have the two key programs on disk. So far, nothing out of the ordinary has been done. Two methods of merging are presented, each having its own advantages and disadvantages.

RICK'S METHOD:

This method yields the most compact program but requires that you change and reassemble the source code whenever you change the length of the BASIC program it is to be used with. It assumes that the machine code will begin immediately after the BASIC code. In addition, any changes will require that you also change the CALL jump vector. When you write the BASIC program, you never know until you're done exactly where it will end. When you set up the POKEs for the CALL vector, the values can be 1, 2 or 3 digits long. You appear end up in a no-win situation. If the number of digits changes, the length of the BASIC program changes which in turn changes the CALL vector which means changing the BASIC program, and so on...

Simply make all 3 numbers three digits long. The first POKE will always be 195. Make the other two both 000. In this way you can change the numbers without changing the program length. BASIC won't care if you POKE 33283,019 instead of POKE 33283,19.

Before you can assemble the source code you must know the ORG address, that is, where it will be loaded. This address will become the same as the end address of the BASIC program. If you have 'The' BASIC EDITOR, load the BASIC program and note the END@ number (in hex) at the bottom of the screen. Otherwise, you can get this value from the disk directory. It's in the SADR column of the directory. This hex address now becomes your ORG address. It should be 8384 if you typed the program as written (watch the spacing in the REM). Use your screen editor to change the ORG then assemble the program. If you are using the original CCII assembler (as opposed to the Macro Assembler, which is frequently more trouble than it's worth), you can leave the file .LDA. There is no need to convert to .PRG. Note the address of the last assembled instruction at the ENDPRG label which the assembler prints out when it's done. This should be 83A4. You'll need this in a moment.

LOAD the BASIC program. From FCS, LOAD your machine language program. The two programs are now back to back in memory. Here's where the trick comes in. In order to SAVE the whole mess from BASIC, we need to tell BASIC where the new program ends. The only end address it currently has is the old one of the BASIC program. However, we've extended it by adding the machine language portion..

Now you need that last address at the ENDPRG label in your assembly printout. In the Programming Manual one of the 'Key Memory' locations listed is 32982 (Points to end of BASIC source and start of BASIC variables). This and 32983 are the locations which we must change to fool BASIC. The start of variables pointer (SOV) marks the end of the actual BASIC program and the start of the memory area where BASIC's variables can start. This location changes every time you add to or delete lines in the program. that's why a pointer is needed, so we don't waste space. The variables start right after the program ends. By using this pointer location we are fooling BASIC into thinking that our machine language program is part of the BASIC program. This protects the routine so it cannot be wiped out by variables, etc. When using this pointer remember to POKE the A4 first (164 decimal) then the 83 (131 decimal). Use IMMEDIATE MODE and type:

```
POKE 32982,164:POKE 32983,131 (RETURN)
```

We're all set. Simply, SAVE the program from BASIC. It can be LOADed from BASIC and RUN as any other program. Test it. Just remember that if you make even the tiniest change in the BASIC program, you'll have to remerge the two using a new value for the ORG address. This difficulty is overcome by--

TOM'S METHOD:

This procedure requires that you assemble the source code twice. It also yields a somewhat longer total program, although this may be inconsequential. The big advantage is that you can make minor changes to the BASIC program without having to reassemble the source code. For most applications, this will be the better method. First, change the source code in Listing 2 as follows (A.L. stands for assembly language):

After the instruction W EQU 30 add—

```
ENDAL   EQU     849AH   ;WHERE WE WANT A.L. TO END
```

Between the ENDPRG label and END START add—

```
ENDPRG: REORG   EQU     ENDAL-($-START)
        ORG     32982   ;START OF VARIABLES POINTER
        DW      ENDPRG  ;MOVE POINTER TO PROTECT A.L.

        END     START
```

In this procedure the main difference is the start address of the machine language program. One advantage is that we can make the total program exactly fill a given number of disk sectors. (One disk sector holds 128 bytes.) Because BASIC begins at 829A (hex), additional numbers ending in 001A hex (e.g. 831A) will fill an odd number of disk sectors and those ending 009A hex (e.g. 839A) will fill an even number of disk sectors. As an example, we chose 4 sectors, making the end address 849A hex. This will add sufficient 'space' between BASIC and the assembly language for future changes.

The next trick is to discover the ORG address for the assembly language. We want it to end at 849A, but, until we assemble the program, we don't know how long it will be. The first line after the ENDPRG label helps us to accomplish our goal. ENDAL is set at the start as 849A. In the expression the '$' symbol is a notation for the current assembler address. By subtracting the address of START from it, we get the difference, or the length of the program. Subtracting this result from the address ENDAL, we calculate the starting address of the machine language. Now assemble the program for the first time. The assembler will print the REORG address (assuming you look for it) in parentheses. It should come out as 847A. Go back and change the initial ORG with the editor from 8384 to 847A. Reassemble the edited program. If you did everything right, REORG should come out the same as ORG. All that remains to be done is to change the CALL vectors in the BASIC program to reflect the new location of the machine language and to merge the two programs, as with Rick's method. To change the CALL vector, line 120 of the BASIC program becomes:

```
POKE 33282,195:POKE 33283,122:POKE 33284,132
```

With Rick's method you had to manually POKE the pointer values at 32982 & 32983. With Tom's method, we let the assembler do it for us. By setting the second ORG at the end to 32982 and using the DW (define word, 2 bytes) directive, we can insert the end address (ENDPRG label) into the required memory locations. A word of caution is in order. You must use the old assembler's .LDA file to do this. The .PRG file won't work! If you must create a .PRG file, you will have to POKE 32982 and 32983 manually as with Rick's method. In either case, you can still SAVE the entire program from BASIC.

Before concluding, I need to mention a couple of possible bugs. The first one is that you cannot LOAD these hybrid programs using the DOS of 'The' BASIC EDITOR. Apparently this editor uses a different method to calculate the end of the BASIC program rather than using the SADR address on disk. The other possible problem is that using 'The' BASIC EDITOR's HELP feature will effectively strip off the assembly language program when you attempt to re-SAVE it. Therefore, don't take chances. Use BASIC's direct SAVE and LOAD commands and everything will be fine. □

LISTING #1

```
100 REM TEST OF SCROLL PATCH
110 PLOT 12,15
120 POKE 33282,195:POKE 33283,132:POKE 33284,131
130 LN=9
140 FOR J=1 TO 40
150 IF LN<19 THEN LN=LN+1:PLOT 3,10,LN:GOTO 180
160 X=CALL(0)
170 PLOT 3,10,LN:PRINT SPC(30)"":PLOT 3,10,LN
180 PLOT 6,J:PRINT J;" TESTING---SCROLLING---"
190 NEXT J
200 PLOT 8,6,2
210 END
```

LISTING #2 (Rick's version)

```
;SCROLL PATCH ADD ON TO 'BASIC' PROGRAM

;SCROLL AREA PARAMETERS.....

X       EQU     10      ;STARTING COLUMN ON SCREEN
Y       EQU     10      ;STARTING ROW ON SCREEN
H       EQU     10      ;# OF LINES TO SCROLL
W       EQU     30      ;# OF CHARACTERS WIDE TO SCROLL

        ORG     8384H   ;END ADDRESS OF 'BASIC' PROGRAM

        ;28672 is start of screen memory (X=0, Y=0)
        ; so first line below is starting screen location

START:  LXI H, 28672+128*Y+X+X
        MVI B, H-1      ;count lines

LOOP2:  MVI C, W*2      ;how wide before next line

LOOP1:  LXI D, 0080H    ;128 decimal (down 1 line)
        DAD D
        MOV A,M         ;get a byte
        LXI D, 0FF80H   ;-128 decimal (back up 1 line)
        DAD D
        MOV M,A         ;reload byte in new location
        INX H           ;next location
        NOP             ;becomes INX H w/no color scroll
        DCR C           ;done with this line?
        JNZ LOOP1       ;no
        LXI D, 128-W-W  ;yes, next line
        DAD D
        DCR B           ;done all lines?
        JNZ LOOP2       ;no
        RET             ;yes, back to BASIC program

ENDPRG:

        END     START
```

A PASCAL FOR THE COMPUCOLOR II

Part III. A Roadmap to successful installation and use.

Doug Van Putte
18 Cross Bow Drive
Rochester, NY 14624

In this part of the Tiny-Pascal series we will provide step-by-step instructions for installing Tiny-Pascal, hoping to provide more readers with enough inspiration to tackle this tutorial series. While the necessary documentation was referenced in Part I, the method of installation can be intimidating to those willing but unfamiliar with the approach used.

The installation instructions are taken from the implementation of Tiny-Pascal, written in the FORTH language, prepared by Dr. Jim Minor. Both the FORTH language and the Tiny-Pascal language are utilized in this installation, and both are available from the CHIP library by writing to the author. FORTH is supplied as a PRG file, while Tiny-Pascal is supplied in the form of FORTH programs, or 'screens.'

FORTH (and Tiny-Pascal) utilizes the disk in 1024 byte units called 'screens.' The screen contains the program code lines, and is used as the means of displaying, entering, and editing programs using a special editor. A screen consists of 16 lines of 54 characters. Only one screen can be displayed on the CRT at a time. A program may consist of more than just one screen, each screen linked to the other by a special coding. A disk side can hold fifty screens, numbered 0 to 49. Blank screens, or screen templates, are used to enter new programs, so it is convenient to have a 'starter screen set' from which to begin.

A starter set of screens for FORTH is supplied with contents on screens 0-19, starting at block 0. Screens 0 and 1 contain a conventional, but dummy, FCS directory. Screens 2 and 3 contain 'boilerplate.' Screens 4 and 5 contain compiler error messages. Screens 6 to 16 contain an editor which will be bypassed in favor of a better editor that comes as part of the Tiny-Pascal disk. Special FORTH words are contained on screens 17 to 19. I recommend that these FORTH starter sets be backed up by using a disk copy program that works without a directory. This copy should then be used as a 'program' disk, to hold programs written on blank screen templates.

The starter set of screens for Tiny-Pascal is supplied with contents on screens 0-9 and 20-37. Screens 0-1 and 4-5 are utilized for the same purpose as those on the FORTH starter set, ie: FCS directory and error messages. Screens 2 and 3 contain the Tiny-Pascal error messages, and 6-9 contain sample Tiny-Pascal programs. Blank templates for program development are provided on screens 10 to 19. Screens 20-37 contain a FORTH line editor which is superior to the editor provided on the FORTH disk. This is the editor we are using to enter and modify code. The Tiny-Pascal set should also be backed up before program development begins.

Our objective is to install Tiny-Pascal (the compiler) and to enter a Tiny-Pascal program using the screen editor. Once the Tiny-Pascal compiler has itself been compiled and saved in PRG format, subsequent sessions with Tiny-Pascal are significantly simplified. [This process consists of saving an expanded version of FORTH which contains Tiny-Pascal command words. With such a version, the FORTH responds to Tiny-Pascal commands as though it were a Tiny-Pascal system. Ed.] You will need CHIP FORTH Disk #46, and the two Tiny-Pascal disks, No. 83 and No. 84. In the text below, '(cr)' means press the carriage return key.

1) Place the FORTH disk, CHIP #46, in the disk drive. From FCS type **RUN FORTH**⟨cr⟩

You should see the FORTH prompt 'OK.' If it does not appear, press (cr).

2) Type at the FORTH prompt,

HEX 1 1A +ORIGIN ! COLD⟨cr⟩

This will turn on the error message text.

3) Place CHIP Disk - 83 in the disk drive. Type

6 LOAD⟨cr⟩

This will compile Tiny-Pascal into FORTH. Wait patiently.

4) Now place a fresh formatted disk into the drive. Type

SAVE TPAL4⟨cr⟩

This will save the augmented FORTH, containing Tiny-Pascal commands, to disk.

5) Place CHIP Disk # 84 into the drive. Type

20 LOAD⟨cr⟩

This will compile the line editor into our augmented version of FORTH (TPAL). Wait patiently again.

6) Replace the previously-used formatted disk in the drive. Type

SAVE TPALED⟨cr⟩

This will save our FORTH/PASCAL Editor to disk.

Having made these changes, we need no longer be concerned with them when using Tiny-Pascal. Our two Tiny-Pascal programs, TPAL4.PRG and TPALED.PRG, can be loaded from FCS using the RUN command from now on. These two programs are our Tiny-Pascal system.

Assume, now, that we wish to create a new program on a blank screen template and save it to disk.

7) Place the Tiny-Pascal system disk, containing TPAL4 and TPALED, into the disk drive. Type

RUN TPALED⟨cr⟩

This will load the line editor. Wait for the 'OK' prompt.

8) Now place your backup copy of Disk #84 into the drive. Type

10 LIST⟨cr⟩

This will load a blank screen template.

9) To invoke the line editor, type **EDITOR⟨cr⟩**

10) You may now type in a program. Part I of this series lists the editor commands. Refer to them for assistance. You may use the sample program given in Part II of this series for practice.

11) When the sample program is all typed in, type

FLUSH⟨cr⟩

to save screen #10 just entered.

12) To invoke the Pascal compiler, type

PASCAL⟨cr⟩

13) To compile the program you just wrote on screen #10, type

10 LOAD⟨cr⟩

14) To run the program just compiled, type just the program name at the prompt. For example, to run Rectanglearea, just type

RECTANGLEAREA⟨cr⟩

at the prompt.

The following commands may be used for editing a program. You will want to delete the compiled program, containing the errors, before compiling a new one of the same name.

14) To delete a compiled program, you may use the FORTH 'FORGET' command. For example, to delete a bad version of RECTANGLEAREA, type

FORTH FORGET RECTANGLEAREA⟨cr⟩

before re-compiling an edited program version. This will delete the 'bad' version from memory.

FORTH FORGET ⟨filename⟩⟨cr⟩

will delete any compiled program.

15) To re-invoke the screen editor for correcting mistakes or any editing of a program, type

EDITOR⟨cr⟩

16) To get screen #10 back again, type

10 LOAD⟨cr⟩

Now the text may be edited using the line editor commands. You will need to recompile following steps 11, 12 and 13 above.

What if your entire program takes more than one screen? There is a FORTH word (a symbol, really) that 'tells' the compiler there is yet another screen connected with this program. This word is the 'continuation command' and looks like this: **--⟩**

It is always preceeded by a space. The continuation command is entered after the last Pascal command on the screen. The continuation command assumes that the 'continuation' is on the screen with the next higher number from the screen on which it appears. You cannot, for example, continue screen #15 on screen #18. Screen #15 must continue on screen #16.

How do you get more blank screens? Use the Tiny-Pascal command(really a FORTH command we are borrowing) COPY to create blank screens. Assume screen #19 is blank, and screen #20 contains information you can erase.

17) Load TPALED by typing from FCS,

RUN TPALED⟨cr⟩

18) To invoke the line editor, type

EDITOR⟨cr⟩

19) To make a copy of screen #19 on screen #20, type

19 20 COPY⟨cr⟩

This overwrites the contents of screen #20 with a blank template.

Or you may do it another way. First perform steps 17 and 18 above.

20) Type **20 LIST⟨Cr⟩**

This will load the screen we want to blank out.

21) To clear screen #20 in memory, type **WIPE⟨cr⟩**

22) When you have entered the new program lines, type

FLUSH⟨cr⟩

to save the new screen #20 to disk.

It is not advisable to blank screens 0-5 on the disk, since they contain the directory and the error messages. Any screen beyond 5 may be used for programs. Preserve the original sampler disks, however. Use backup disks for your programs.

You may not change the screen number of any screen, but the contents of a screen may be moved to another screen using the COPY function from step 19, above. The source screen may then be blanked if desired.

How can you remember which screen a program is on? You can keep a logbook of screen contents, or the 'dummy directory' may be used to 'log' the screens from 6 to 49. Lets add a directory entry for RECTANGLEAREA. Assume this program is on screen #10.

23) Place the Tiny-Pascal system disk in the drive.

24) From FCS type,

SAVE RECTAR.TPL 0 10⟨cr⟩

This command makes a directory entry consisting of the program name, RECTAR, and the number 10, which references the originating screen of the program. The number of entries in your 'program catalog' (FCS directory) is limited to 13. This 'catalog' of programs can be listed directly from FCS with the DIR command.

Now you have all the resources to get started with Tiny-Pascal. Gather together your FORTH and Tiny-Pascal disks, follow the course provided, and journey to a fine experience. Best wishes on a safe trip! □

UNCLASSIFIED ADVERTISEMENTS

FOR SALE: One broken v6.78 CCII with 32K memory, switchable lower case character set. Eight related disks including Soundware, Muldowney's Assembly Language Tutorial, assorted games, formatter, and more. Scads and scads of documentation, including service manual, programming manual, "Basic Training"," Color Graphics", and most Colorcues. This unit has analog board and power supply problems. You may be able to repair it. It is suitable for spare parts at least. I'll take the best offer over $100 plus shipping. Write Tom Andries, 815 W. Douglas Road, Lot 1, Wishawaka, IN 46545, or call 219-272-6768.

FOR SALE: One unbroken 3651, 40K RAM, full keyboard, too much software, CRT filter, switchable lower case, custom key caps for Compuwriter and screen editor. One internal 5" MD drive. I'll take $1200, you pay shipping; and I'll convert your CD disks to MD where possible. COLORCUE, 609-234-8117.

Forum - back issues still available!

All back copies of FORUM are still available and loaded with informative articles, programs, and tips:

Vol I, No. 1 through 4, $3 each.
Vol I. No. 5 and 6 (double issue), $5.
Vol II, No. 1 through 4, $4 each.
Vol II, No. 5 and 6 (double issue), $6 each.
Vol III, No. 1 (final issue), $4 each.

All prices include postage to North America. Europe and South America please add $1.00 (US) per issue. Asia, Africa and the Middle East please add $1.40 (US) per issue. Order from:

Mr. Arthur Tack, 1127 Kaiser Road, S.W., Olympia, WA, 98502, U.S.A.

RAM/EPROM for the CCII

An add-on memory board with 8K RAM plus space for 8K EPROM is available for the Compucolor II or 3651. This 8K RAM/EPROM can hold assembler programs at address 4000H. Your existing 16K or 32K of user RAM is still available for other Basic or Assembler programs. Some minor modifications are required to the main logic board (five soldered links.) Some versions of v8.79 computers may also require an update FCS ROM. All versions of v6.78 require the update ROM. The RAM board can also be easily connected to a ROMPACK system, instead of the fixed 8K EPROM. In this case, two banks of memory are selected by a switch. To retain the contents of the 8K RAM after power-off, a battery backup is available. A second 8K RAM/EPROM can be connected to the first board. PROGRAM PACKAGE INSTALLERS. PO Box 37, Darligton, Western Australia 6070.

Back issues of COLORCUE contain a wealth of practical information for the beginner as well as the more advanced programmer, and an historical perspective on the CCII computer. Issues are available from October 1978 to current.

DISCOUNT: For orders of 10 or more items, subtract 25 % from total after postage has been added. **POSTAGE:** for U.S., Canada and Mexico First Class postage is included; Europe and South America add $1.00 per item for Air Mail, or $ 0.40 per item for surface; Asia, Africa, and the Middle East add $ 1.40 per item for Air Mail, or $ 0.60 per item for surface. **SEND ORDER** to Ben Barlow, 161 Brookside Drive, Rochester. NY 14618 for VOL I through VOL V; and to Colorcue, 19 West Second Street, Moorestown, NJ 08057 for VOL VI and beyond.

1978	**VOL I**	**$3.50 each**
	No. 1-3:	OCT/ NOV/ DEC
1979	**VOL II**	**$3.50 each**
	No. 1-3:	APR/MAY/JUN
	No. 4-5:	JAN/FEB/MAR
	No. 6-7:	AUG/SEP/OCT
	No. 8:	NOV XEROX COPY, $2.00
1980	**VOL III**	**$1.50 each**
	No. 1	DEC/JAN
	No. 2:	FEB
	No. 3:	MAR
	No. 4:	APR
	No. 5:	MAY
	No. 6:	JUN/JUL
1981	**VOL IV**	**$2.50 each**
	No. 0:	DEC/JAN
	No. 1:	AUG/SEP
	No. 2:	OCT/NOV
1982	No. 3:	DEC/JAN
	No. 4:	FEB/MAR
	No. 5:	APR/MAY
	No. 6:	JUN/JUL
	VOL V	
1983	No. 1:	AUG/SEP
	No. 2:	OCT/NOV
	No. 3:	DEC/JAN
	No. 4:	FEB/MAR
	No. 5:	APR/MAY
	No. 6:	JUN/JUL
1984	**VOL VI**	**$3.50 each**

-----DISK VOLUME SPACE-----				-------1ST FILE ENTRY-------				-------2ND FILE ENTRY-------				-------3RD FILE ENTRY-------				-------4TH FILE ENTRY-------				-------5TH FILE ENTRY-------			
ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION	ADDR	BYTE	TRANS	FUNCTION
8200				8217			Attribute	822C			Attribute	8241			Attribut	8256			Attribute	826R			Attribute
01			Marker	18				2D				42				57				6C			
				19				2E				43				58				6D			
02			Attribute	1A			File	2F			File	44			File	59			File	6E			File
				1B			Name	8230			Name	45			Name	5A			Name	6F			Name
03				1C				31				46				5B				8270			
04				1D				32				47				5C				71			
05				1E			File	33			File	48			File	5D			File	72			File
06			Volume	1F			Type	34			Type	49			Type	5E			Type	73			Type
07			Name	8220				35				4A				5F				74			
08																							
09				21			Version	36			Version	4B			Version	8260			Version	75			Version
0A																							
0B				22			SBLK	37			SBLK	4C			SBLK	61			SBLK	76			SBLK
0C				23			0005	38			000A	4D				62			0005	77			000A
0D				24			SIZE	39			SIZE	4E			SIZE	63			SIZE	78			SIZE
0E				25			0005	3A			1202	4F				64			0005	79			1202
0F																							
8210			Not	26			LBC	3B			LBC	8250			LBC	65			LBC	7A			LBC
11			Used																				
12				27			LADR	3C			LADR	51			LADR	66			LADR	7B			LADR
13				28			8200	3D				52				67			8200	7C			
14																							
15				29			SADR	3E			SADR	53			SADR	68			SADR	7D			SADR
16				2A				3F				54				69			8200	7E			
				2B			Spare	8240			Spare	55			Spare	6A			Spare	7F			Spare

COLORCUE

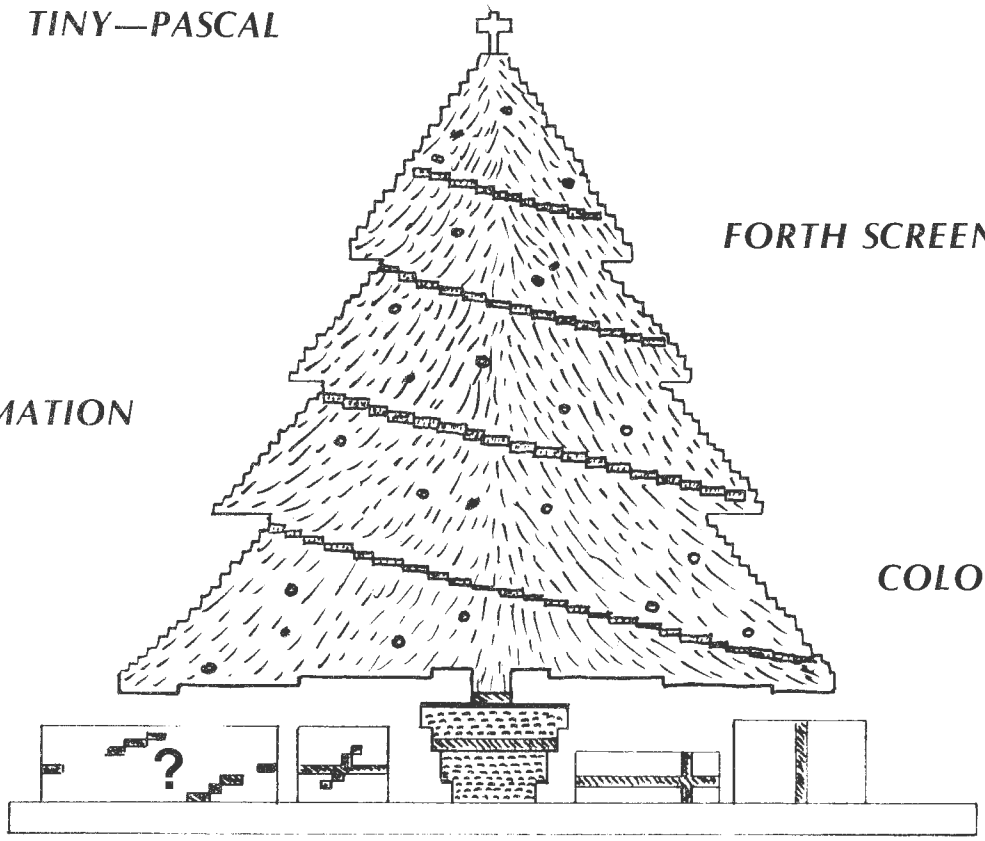
A BI-MONTHLY PUBLICATION BY AND FOR INTECOLOR AND COMPUCOLOR USERS

VOLUME VI
NUMBER 5

TINY—PASCAL

FORTH SCREEN EDITOR

ANIMATION



COLORWORD

ZIP

HEX TO ASCII CONVERSION

Colorcue

VOLUME VI, NUMBER 5 SEPTEMBER/OCTOBER 1984

CONTENTS

EDITOR: JOSEPH NORRIS COMPUSERVE: 71106, 1302

3651 PRICE REDUCTION

Intecolor Corporation has announced a price reduction on the 3651 computer. With 32K memory, deluxe keyboard, one 90K internal disk drive, manual and sampler disk, this excellant computer is available for $1450.00. The 3651 carries FCS v9.80, which is nearly identical to v8.79 of the CCII. [COLORCUE uses a 3651 for all its editing and data base work.] A CDBKUP.COM program is provided for transferring CD programs to the MD disk format used by the 3651. In order to use this program you must have an external CD drive available for the transfer. [COLORCUE can provide this service for you if you wish at very low cost.] In our experience, all files with BAS, SRC, and TXT extensions transfer easily. Most PRG files written for v8.79 transfer readily. A few require modest changes to esoteric ROM calls. This is easily accomplished with IDA and is straightforward.

The 3651 is a one-piece computer, with all the features of the CCII, plus faster and more reliable disk reads, improved internal and external commands, a proper RS232 port, and the ability to handle four drives, either 5" or 8", for a total drive capacity of 2.4 megabytes. FCS accepts both upper and lower case commands. It is a truly professional version of the CCII and a pleasure to use. All important software for the CCII is available for the 3651. In fact, nearly all your present software is usable once it is transferred to the MD disk. The new price includes your selection of 10 software packages from the Intecolor catalog. Intecolor Corporation. 225 Technology Park, Norcross, GA 30092. (404)-449-5961.

COLORCUE is published bi-monthly. Subscription rates are US$12/year in the U.S., Canada, and Mexico (via First Class mail), and US$30 elsewhere (via Air Mail). All editorial and subscription correspondence should be addressed to COLORCUE, 19 West Second Street, Moorestown, NJ 08057, USA. (609-234-8117) All articles in COLORCUE are checked for accuracy to the best of our ability but cannot be guaranteed error free.

from the Editor's Desk

Anything involving computers is in a constant state of flux these days, and we at Colorcue aren't exempt from the trend. While we are 200 subscribers strong, we are rather weak in terms of the kind of activity that keeps an expensive magazine such as Colorcue alive and energetic. In short, we have had to accept the unavoidable truth that we cannot continue to publish in our present format.

Volume VI will be the last for Colorcue (one issue to come) and we will be combined with CHIP, the newsletter of the Rochester User Group, under the guidance of Rick Taubold. The details are explained elsewhere in this issue, so please take note of them, and lend your support to this splendid group as they continue their long history of Compucolor activity. I will continue as an author for CHIP, and have offered my support as my time and means permit.

Since my work keeps me at one keyboard or another most of the time (on three different operating systems!) I share the turmoil felt by most of the industry on a day-to-day basis. With the avalanche of technological sophistication and the increased demands of business on software efficiency, those of us who indulge in computer interests as avocation are finding a major change in the way we must approach our hobby, principally in the ways we differentiate between hobby and our business activity.

Small computers are great fun, relaxing to operate, and elegant learning tools. Most of the commercial "small" machines are rapidly disappearing as bankruptcy claims its victims left and right. Those of us accustomed to being in the vanguard of computer experimentation and development, pounding keyboards late into the night, assembling equipment from surplus houses, and pushing forth the frontiers of technology, are now replaced by the huge engineering teams and multi-billion dollar resources of industrial giants. The new technology belongs to the rich and powerful.

While we have lost our mainstream importance as hobbyists, I greet this change as a positive one overall. The hobby can now flourish as a hobby without the oppression of changing business needs and economic necessity. While we will continue to benefit from the reduced prices of obsoleted equipment and software, we will also be banding together as a computer-interest community, consolidating our resources, much as we did in the beginning; sharing more, learning more, and contributing more. Expect to see new magazines devoted to the hobby of computer electronics, such as the proposed hardware periodical, Computersmith, to be launched in the Spring of 1985 by Ed Dell in New Hampshire. (Ed publishes The Audio Amateur and Speaker Builder, two very fine magazines for audio enthusiasts.) Expect to see more magazines for the amateur devoted to experimental programming philosophies, and an ever increasing proliferation of public-domain software. Relieved from the pressures of profit making, many talented people will find more time to devote to the avocational aspects of their interests.

It is important to realize that no computer is really obsolete in the face of this kind of activity. Much as the humble Sinclair computer challenged many professionals to overcome its limitations, so can we pursue the challenge of implementing many new and fascinating procedures on the 8080 and Z80 machines (such as the "window", pull-down menu, and multi-tasking.) Experimenting, at least on a small scale, is easier on a simple computer with a straightforward operating system (and FCS is ideal for this!). Programming is quick and easily changed.

Many of you have not yet tapped the abundance of excellant programming tools available from the CHIP library, such as Pascal, Forth, Tiny-C, and the FASBAS and ZIP compilers. Colorcue subscribers have written a word processor with spelling checker, an analog circuit analysis program complete with Bode plots, and several very clever disassemblers. We have in our software bank the best programs ever written for program development for any computer, including editors, assemblers, disassemblers, and compilers. It is no longer appropriate to look only for programs written specifically for the Compucolor. We are equipped to do almost anything any other computer can do. I would like to see programs to emulate the batch files of MSDOS, and the power of dBASE II on the Compucolor. They are within our grasp.

If you are planning to purchase new equipment, and have given your best effort to the CCII, you are in a very fortunate position to make wise choices and advance your skills. Few computer owners have been in a position to get as much for their investment as we have, even without factory support.

Intecolor has been in communication with Colorcue and CHIP in the last months with offers of a renewed effort to be of service to CCII owners. While there isn't a great deal they can do at this late date, we appreciate their interest and spirit of good will. We appreciate their acknowledgement of our independent achievements as a community.

You will notice the announcement of a new compiler by Peter Hiner in this issue. There is irony and humor in the fact of its appearance in this particular issue of Colorcue. I remember very well that FASBAS was first reviewed in the final issue of FORUM several years ago.

So as we prepare to bid "farewell" to our present format, we are also proceeding with continued energy to improve the resources for the CCII, and looking forward to an expanded CHIP. We extend our best wishes, and our appreciation, to this organization, for coming to the rescue still another time.

This issue has been greatly delayed for lack of sufficient material to fill its pages. The final issue will be subject to the same restraints, so I urge you to submit your materials as promptly as you can. I would like to see the Nov/Dec issue long before the Spring of 1985, and I will need your help to achieve that goal. My wife, Susan, and I extend our best wishes to you all for a happy and fulfilling New Year. □

Joseph

PETER HINER *11 Penny Croft, Herpenden, Herts, AL5 2PD, ENGLAND*

A NEW COMPILER CALLED

I expect that most of you already know of my compiler FASBAS, which speeds up Basic programs. Those of you who have used FASBAS will know that it is easy to use, that it accepts Basic programs with virtually no restrictions, and that speed increases of up to five times can be achieved. Now I have produced a new compiler called ZIP, which is not as easy to use and which imposes some restrictions, but which can make programs run much faster than when compiled by FASBAS.

ZIP achieves much greater speed by treating all variables and constants as integers (instead of using floating-point arithmetic like Basic and FASBAS). This means that ZIP is not suitable for all applications, and that Basic programs may need to be modified to suit the requirements of integer arithmetic. The manual supplied with ZIP explains how to modify your programs, and the disk includes an Integer Basic Interpreter with debugging facilities. So you should be well equipped to overcome any problems, and you will find that ZIP opens the way to using Basic for fast graphic displays and arcade games.

A review in the September, 1982 issue of BYTE included some benchmark programs for comparing the relative speed increases achieved by integer and floating-point compilers for the Apple computer. The table in Fig 1. compares ZIP and FASBAS results with the ranges of results for Apple compilers. The figures in the table are the approximate number of times faster than normal Basic for each benchmark program. The ZIP results for benchmark programs 7 and 8 are bracketed because integer arithmetic does not give correct answers. (One of the Apple integer compilers handled this situation by reverting to floating-point arithmetic.)

You can see that, while FASBAS comes somewhere in the middle of the field of floating-point compilers, ZIP beats them all.

You can use ZIP to compile existing Basic programs. Many will work with little or no modification, some will require care and patience, and others simply will not be able to give the required results using integer arithmetic. However, the best way to use ZIP is to consider it as a tool, which allows you to use familiar Basic functions for the framework of your program, while exploring techniques similar to those used in Assembly Language programs to optimize the speed of critical routines.

The ZIP manual describes an example where the PRINT function is replaced by POKE to display a row of stars on the screen. The times for 1000 executions of this routine are shown in Fig 2, in seconds.

The POKE method gives ZIP an 18-fold speed improvement over the normal Basic PRINT method. To see how this compared with Assembly Language (machine code), I wrote an Assembly Language routine which took 3.3 seconds for 1000 executions. With further optimization of the ZIP routine I could cut the time to seven seconds, achieving half the speed of machine code (and more than 26 times the speed of Basic) for this routine. This shows that fast graphic displays and arcade games in Basic are really made possible by ZIP.

How, then, should you treat ZIP? Should you consider it as another Basic compiler, which can produce much faster results once you have overcome the obstacles of integer arithmetic? Or should you treat it as a new way of writing programs, almost like using a new language which you nearly know already? Whichever way you choose, ZIP offers you the excitement of holding a tiger by its tail.

The minimum requirements for running ZIP are 16K of user memory, one disk drive, and v6.78, v8.79, or v9.80 Basic. The programs compiled by ZIP are generated as PRG files for convenience of operation, and will automatically run on v6.78, v8.79, or v9.80 machines.[*]

I propose to distribute ZIP in the same way as I distributed FASBAS, encouraging user groups to make the program available to their members on payment of a fee of $15.00 per copy. Individuals are welcome to order direct from me, in which case the price is $25.00 for the disk (in CCII format), manual and airmail postage. Personal

Fig 1.

BENCHMARKS	1	2	3	4	5	6	7	8
FASBAS	33	24	29	16	28	8	(5)	(17)
Apple Integer	11-16	9-15	7-14	5-15	8-17	8	5	4
ZIP	4	4	3	3	3	2	2	4
Apple F. P.	3-6	2-4	2-4	2-4	2-5	2-3	2-5	3-5

zzzzzzzippppppp

checks are welcome as the method of payment.

The deal is further complicated by an offer of the latest version (v12.24) of FASBAS, which now includes operation on machines with v9.80 Basic, output files in PRG form (instead of LDA) and compiled programs which will automatically run on v6.78, v8.79, or v9.80 machines.[*] The new version also fixes a few obscure bugs and makes some improvements to the size and speed of compiled programs.

Using	BASIC	FASBAS	ZIP
PRINT	187	143	82
POKE	346	132	10

Fig 2.

Previous purchasers of FASBAS will get a free update included on the ZIP disk. New purchasers may order the pair of compilers (ZIP and FASBAS) on one disk for a combined price of $40.00 from me (or a fee of $30.00 if obtained through a user group). As always, I will send a free replacement for any disk which is unreadable or damaged in transit. □

[* This means, for example, that a program written in v6.78 will run on any other version of FCS without a jump table or conversion of any kind. ED.]

HEX TO ASCII CONVERSION (intecolor 8000)

BOB MENDELSON *27 Somerset Place, Murray Hill, NJ 07974*

Whenever the ISC 8001 computer is used for BASIC programming there is a need for decimal equivalents to the hex memory addressses used for PEEK and POKE statements. While there are tables for these conversions, unless the table is exceedingly long, interpolation is required. It is true that Texas Instruments makes a hand calculator for direct hex-decimal-hex conversions, but even here it is necessary to make a manual subtraction of 65536 if the original hex number is greater than 8000. This is because the ISC ROM requires that numbers greater than 8000H be used in Basic as 65536—hex#. This program will take any hex number as input, convert and print it as a decimal number, including numbers greater than 8000H. The program is only 180 bytes long and is well worth adding to your general utility disk.

The initial steps from START to HEX set up the title and column names. At HEX, the routine HEXIN calls from ROM a program that takes the ASCII input for a hex number and converts it to a true hex number which is stored in memory at ORIG.

The standard hex to decimal routine STDEC is then called. It first loads 5 bytes of memory defined as UNIT, TEN, HUN, THOU and TTHOU designating each of the 5 digits in the decimal number. The value 30H that is loaded is the ASCII value for zero. As we shall see, this is very important.

Now the actual conversion starts. By use of the 2's complement method, starting with the most significant digit, successive additions are made of the hex equivalent of 10000 decimal. Each addition is counted by adding one to the TTHOU memory byte. This raises ASCII 30H to successive ASCII numbers. This continues until a borrow occurs and the carry is 'true.' The same procedure is followed for each of the next three digits. The unit digit is converted by adding it to 30H and saving it.

The program is now ready to print the result under the decimal column, beginning at routine PRINT. Register C is set to 5 as a counter. Register B is set to 30H as a detector of unneeded zeros to the left of small numbers, and register D is set to zero as a detector of the first digit greater than zero that is printed. This will be used to retain the print of zeros in the middle of a number.

The most significant digit is recovered and compared with 30H. If it is not a zero, it is printed and the D register is set to 1. The next digit is recovered and the same sequence is followed. A zero recovered while D = 0 will be replaced by a space. When D = 1, the zero will be printed. The standard decimal conversion process is now over.

The conversion to the ISC decimal equivalents requires that the input number be checked to see if it is greater or less than 8000H. This is done at DSPASC. If less than 8000H, the print will be the same as described above. If it is greater than 8000H, then 65536 will have to be subtracted to get the ISC minus number. A modified 2's complement method is used to achieve this. Since the hex value for 65536 is 10000H, it is too large to use 16 bit arithmetic. Therefore a 24 bit routine is used. Further, the need for a minus sign can be met by merely printing it, since we won't be in this portion of the software if it is not required. The routine starts at NEG, where '00' is added in front of the input hex number (1 byte) and the constant 010000H is provided by loading three bytes. It is time, now, to perform the subtraction-with-carry on the least significant byte. The result replaces the input hex number at the ORIG memory address.

As a result of all of this arithmetic, the ORIG address now holds the hex equivalent of the ISC negative number. The STDEC routine is recalled to convert this new number to 5 ASCII digits which are then printed in the ISC column. The number 8000H cannot be defined since it is neither positive nor negative, so a routine at ISC is used to detect this number and print a 'Not Defined' error message.

The program now recycles and asks for a new input. Any non-hex character at the input will cause a jump out of the program. This may be used for an orderly escape, including a vector to a utility menu. □

```
;LISTING I. HEXASC.SRC - Hex to ASCII Conversion

;          For ISC 8001 Computer
;          by R. M. Mendelson, 7/8/84

;          Program origin = FE00H

;Equates .....................................

       CI      EQU     0103H   ;Console in
       CO      EQU     0109H   ;Console out
       EXPR    EQU     0133H   ;Hex evaluation
       KEYBF1  EQU     9FFFH   ;Keyboard flag
       LO      EQU     010FH   ;List out
       OSTR    EQU     012AH   ;Output string
       UTIL    EQU     5006H   ;Utility ROM, may
                               ; be any address
       BA7ON   EQU     14
       BA7OFF  EQU     15
       CR      EQU     13
       CYAN    EQU     22
       EL      EQU     11
       EOS     EQU     239
       EP      EQU     12
       GREEN   EQU     18
       LF      EQU     10
       RED     EQU     17
       UP      EQU     28
       YEL     EQU     19

       ORG     FE00H

START: XRA     A
       STA     KEYBF1  ;Clear keyboard
       CALL    MESS
       DB      YEL,BA7ON,EP
       DB      'HEX TO ASCII DECIMAL CONVERSIONS-'
       DB      BA7OFF

       ;Column labels

       DB      CR,LF,LF,GREEN,'   HEX    STD     ISC'
       DB      CR,LF,RED,'   ----  ',YEL,' -----  '
       DB      CYAN,'  -----',CR,LF,EOS

       ;Main Routine. Input will appear in the HL
       ;              register as hex #

       JMP     REPEAT  ;Get input arrow
HEX:   CALL    HEXIN   ;Input kybd ASCII input to hex#
       SHLD    ORIG    ;Save it for later use
       CALL    STDEC   ;Convert to std ASCII decimal #
       CALL    MESS    ;Align cursor for dec printout
       DB      CR,LF,'           ',YEL,UP,EOS
       CALL    PRINT   ;Print it out
```

```
;This routine will check if number is greater than 8000H
; and if so will subtract 65536 and print it with a minus
; sign as required by the ISC 8001. If the number is less
; than 8000H, it will print it as standard.
       LHLD    ORIG    ;Recover original hex# from mem.
DSPASC: XRA    A       ;Set A=0
       DCR     A       ;A=FFFFH
       ANA     H       ;If H=> 8xxx, sign = 1 or minus
       JM      ISC     ;If minus print '-' sign
       CALL    PRINT   ;Print it. Dup of col 1( 8000H
       JMP     REPEAT  ;Ready for new input
;Check for input of 8000H which is not defined as (-)#
ISC:   MOV     A,H     ;Get most significant byte
       SUI     80H     ;If H=80 ANS=0, zero flag=1
       ADD     L       ;Any digit but 00 resets zero fl
       JZ      OVFLOW  ;Number is 8000H
;Print minus sign and perform subtraction
       MVI     A,26    ;Cursor left for '-' sign
       CALL    LO      ;Move it
       MVI     A,'-'   ;Minus sign
       CALL    LO      ;Print it
       CALL    NEG     ;Subtract 65536
       CALL    STDEC   ;Print difference as dec #
       CALL    PRINT   ;Print it
       JMP     REPEAT  ;Ready for new input

;Conversion to std decimal number. Fill storage
; memory with ASCII 0 (30H)

STDEC: LHLD    ORIG    ;# to be converted must be
       XCHG            ; in DE to start
       LXI     H,UNIT  ;Point to 5 digit storage
       MVI     C,5     ;Counter
SETTOO: MVI    M,30H   ;Fill with ASCII 0
       INX     H
       DCR     C       ;Count down
       JNZ     SETTOO  ;Re-do

;Convert hex to ASCII (0-65536)

       DCX     H       ;Back up to ten-thou digit.
       LXI     B,0D8F0H ;-10000 (2's comp of 10K)
       CALL    DIGIT   ;Add to the content of DE
                       ; reg until borrow occurs.

       LXI     B,0FC18H ;-1000 (2's comp)
       CALL    DIGIT   ;Do again til borrow occurs

       LXI     B,0FF9CH ;-100 (2's comp)
       CALL    DIGIT

       LXI     B,0FFF6H ;-10 (2's comp)
       CALL    DIGIT   ;Final addition
```

```
        MOV     A,E     ;Get the number of units
        ADI     30H     ;Add ASCII 0 to it
        MOV     M,A     ;Save it.
        RET
;Print out the ASCII standard decimal #

PRINT:  LXI     H,TTHOU ;Point HL to MSB ten-thous
        MVI     C,5     ;Set counter: print 5 digits
        MVI     B,30H   ;ASCII 0 (LXI B,3005H)
        MVI     D,0     ;Marker for digit or zero
PR0:    MOV     A,M     ;Get ASCII digit
        CMP     B       ;Is it ASCII zero?
        JNZ     PR1     ;No! Set marker
        XRA     A
        CMP     D       ;Yes! Check marker for 0
                        ; No digits yet!
        JNZ     PR2     ;Marker is '1'. Print 0
        MVI     A,20H   ;Marker is '0', so
        JMP     PR3     ; print space
PR1:    MVI     D,1     ;Digit for print, set marker=1
PR2:    MOV     A,M     ;Recover non-zero digit
PR3:    CALL    CO      ;Print it
        DCX     H       ;Dec pntr to next sig. digit
        DCR     C       ;Dec counter
        JNZ     PR0     ;Re-do till counter = 0
;Set cursor for last # printout
        CALL    MESS
        DB      '     ',CYAN,EOS
        RET
;Subtract 65536 to get ISC negative number. Must
; get input number and 2's comp of FFFF in 3 bytes
; each into memory, in order LSB, NSM, MSB. The
; 2's complement of 65536 is -10000d.

NEG:    LXI     D,MINUS
        XRA     A
        STA     MINUS   ;MINUS has 10000
        STA     MINUS+1
        STA     ORIG+2  ;Blank MSB for 3rd byte of
                        ; input number.
        INR     A
        STA     MINUS+2 ;The '01' of '010000'

SUB3:   MVI     C,3     ;Counter
        LXI     H,ORIG  ;Point to lsb of original #
        XRA     A
SUBAGN: LDAX    D       ;Get lsb of constant
        SBB     M       ;Subtract lsb
        MOV     M,A     ;Replace ORIG digit
        INX     D       ;Next significant bits
        INX     H
        DCR     C       ;Reduce counter
        JNZ     SUBAGN  ;Not done. Re-do, or
        RET             ; return

;Get ready for next hex input
REPEAT: CALL    MESS    ;Set arrow prompt
        DB      CR,LF,LF,YEL,' ) ',EOS
        JMP     HEX

;Routines for 0-65536 decimal conversion

DIGIT:  PUSH    H       ;Save mem pointer on stack
        XCHG            ;Get # for conver. in HL
        DAD     B       ;Add 2's comp test #
        JNC     ADDIT   ;If borrow then carry=True
        XCHG            ;No borrow, get HL & DE back
        POP     H       ;Recover memory address
        INR     M       ;Incr mem count by 1
        JMP     DIGIT   ;Try subtraction again. Loop
                        ; will cont until carry=True
ADDIT:  MOV     A,C     ;Form 2's complement
        CMA
        MOV     E,A
        MOV     A,B
        CMA
        MOV     D,A
        INX     D       ;DE contains 2's comp
        DAD     D       ;Add it to test #
        XCHG
        POP     H       ;Pop mem pointer off stack
        DCX     H       ;Decrement memory pointer
        RET             ;Back to main program

;Subroutines

HEXIN:  MVI     C,1     ;Counts 4 bytes of ASCII input
        CALL    EXPR    ; of a hex# and converts to
        POP     H       ; true hex. Result in HL reg.
        RET

OVFLOW: CALL    MESS
        DB      RED,'NOT DEFINED',EOS
        JMP     REPEAT

MESS:   POP     H       ;[This is very tricky! Seasoned
        CALL    OSTR    ; programmers, take note! ED]
        PCHL

;Memory address for 0-65536 decimal output

UNIT:   DS      1       ;5 digit memory storage
TEN:    DS      1
HUN:    DS      1
THOU:   DS      1
TTHOU:  DS      1
ORIG:   DS      3       ;Original input hex# + 00 msb
MINUS:  DS      3       ;2's complement of 65536=10000d

        END     START
```


REFERENCES: 8080 Software Design, Book 1. H. W. Sams, 1978, p 257. Computer Design, April 1978, p 168.

.....a new FORTH screen editor.....

TOM NAPIER 12 Birch Street, Monsey, NY 10952

In early 1982 I decided that I would like to try FORTH on my Compucolor II, so I did the logical thing. I bought a listing of FIG-Forth, typed in some 28000 characters, and spent several weekends adapting CP/M based code to the idiosyncrasies of the CCII disk system. Eventually it worked, and it wasn't until several months later that I discovered the DATACHIP library already contained a version of Forth for the CCII. [1]

I found the FIG-Forth editor to be so slow and cumbersome that I often ended up writing programs in assembler rather than in Forth. Recently I took the plunge and wrote my own screen-based editor. It is such a delight to use that I even find myself touching up the layout of Forth screens for fun, something I would never have dreamed of doing before.

Since this editor is written in standard FIG-Forth, it should work with any version of CCII Forth. It should also be Intecolor model independent. Since it uses the editing keys of the expanded keyboard it won't be so easy to use with only the standard keyboard. The allocation of screen numbers in the accompanying listing is quite arbitrary and can be changed to suit the user's disk space, provided the order of the definitions is not changed.

This editor uses one main word and two auxiliary words. The latter two are CLEAR and COPY, which are copied from the FIG-Forth editor. Their usage is 'N CLEAR' to clear disk screen N, and 'N1 N2 COPY' to copy screen N1 to screen N2. The editor is invoked by typing "SEDIT" (for Screen EDITor). At run time it prints the message "NEXT SCREEN NUMBER" and waits for the user to type in the number of the screen to be edited. I've been lazy and omitted the backspace function from the number input, so you'll have to get it right the first time. Type an out-of-range screen number if you want to get a disk error message, and therefore abort a typing mistake.

The specified screen is loaded from disk and displayed on lines 4 through 19. Above these lines is a display of the screen number and the current cursor position. The cursor appears as a blue background travelling under the green ASCII characters. Under the screen, the prompt "PAD", followed by a right arror, appears. This indicates that the line below, initially blank, is a display of the contents of the pad.

One may now type in any characters and have them inserted at the current cursor position. Existing characters on the same line will be moved to the right to make room for the insertion. Any characters moved off the end of the 64 character line will be lost. Note that the insertion process takes a macroscopic time. I occasionally lose one of a double letter pair, and if you are a blindingly fast touch typist you will have to slow down a little.

The DELETE CHAR key deletes the character to the left of the cursor and moves the remaining characters in the line to the left. Lines are treated independently; deleting characters on one line does not move up those on the line below, useful though this might sometimes be.

The cursor keys move the blue cursor around the screen. They auto-repeat and also wrap around at the sides and the top and bottom of the screen. The quickest way from the extreme right of a line to the extreme left is to use the right arrow key. The HOME key returns the cursor to the top left corner.

The RETURN key acts as a "new line" key, moving the cursor to the first left position on the line below.

Complete lines may be manipulated with the ERASE LINE, INSERT CHAR, DELETE LINE, and INSERT LINE keys. INSERT LINE moves the text downward starting with the line the cursor is on, thus enabling one to insert a new line at that point. The last line on the screen is lost in this case. DELETE LINE copies the line the cursor is on to the pad, appearing under the screen display of text, then moves all following lines up one line. ERASE line copies a line to the pad without deleting it. The INSERT CHAR key is used as a 'paste' key; it moves the text from the cursor down to make room for a new line, then copies the text from the pad to the screen. This can be used to transfer text between screens as well as to duplicate lines on the same screen.

ERASE PAGE is used to change from one screen to another. It generates the 'next screen number' prompt and displays the contents of the screen specified. The current screen is flagged as UPDATEd whenever any changes are made to it.

Pressing the ESC key FLUSHes the changes to disk and exits from the editor to FORTH.

One thing to watch—the DELETE CHAR command deletes to the left of the cursor so it cannot erase the last character on the line without one other character being erased first. Also, if one attempts to insert beyond the last character the keyboard input will replace the last character on that line. None of this should matter since the last character of a line should always be a space, otherwise the FORTH compiler will concatenate it with the first character of the next line and be unable to find the resulting word. It took me some minutes of gazing at the message ";:?" before I realized that a FORTH screen contains nothing to indicate where a line ends. (Another tip from hard experience; if a screen contains only a few lines, end them with ";S". If there are more than 255 spaces between the last character and the end of a screen the compiler will hang up.) [Thank you, Tom! I have been scratching my head! ED].

In this listing I've defined four words; CLRS, PLOT, SMOVE, and VHTAB, which my system has built in. This means that this published version displays the screen more slowly than my original version. Happy editing. □

[1] Two versions of FORTH, and three screen editors, to date. [ED]

SCR # 40

```
 0 ( SCREEN EDITOR, 5/1/85   COPYRIGHT 1985 T. M. NAPIER )
 1
 2  6 LOAD  ( CASE FUNCTION )
 3 41 LOAD
 4 42 LOAD
 5 43 LOAD
 6 44 LOAD
 7 45 LOAD
 8 46 LOAD
 9 47 LOAD
10 ;S
11
12
13
14
15
```

SCR # 41

```
 0 ( SCREEN EDITOR  INSERT DELETE )
 1
 2 0 VARIABLE L#    0 VARIABLE C#    0 VARIABLE K#
 3
 4 : PLOT          64 * + DUP + 28672 + ; ( SCREEN ADDRESS )
 5 : LINE          SCR @ (LINE) DROP ; ( GET LINE ADDRESS )
 6 : CHAR          L# @ LINE C# @ + ; ( CHARACTER ADDRESS )
 7 : -MOVE         LINE C/L CMOVE UPDATE ;  ( MOVE LINE UP ONE )
 8 : WIPE          LINE C/L BLANKS UPDATE ; ( CLEAR LINE )
 9 : CLEAR-PAD     C/L PAD C! PAD 1+ C/L BLANKS ;
10 : <SLIDE        CHAR DUP 1 - C/L C# @ - CMOVE ;
11 : SLIDE>        CHAR DUP C/L + C# @ - 1 -
12                   DO I 1 - C@ I C! -1 +LOOP ;
13 : DASHES        0 SWAP PLOT DUP C/L 2 * + SWAP
14                   DO 45 I C! 2 +LOOP ;
15
```

SCR # 42

```
 0 ( SCREEN EDITING COMMANDS )
 1
 2 : CLEAR   SCR ! 16 0 DO I LINE C/L BLANKS LOOP UPDATE ;
 3
 4 : COPY    B/SCR * OFFSET @ + SWAP B/SCR * B/SCR OVER + SWAP
 5             DO DUP I BLOCK 2 - ! 1+ UPDATE
 6             LOOP DROP FLUSH ;
 7
 8 : SMOVE   ROT SWAP OVER + SWAP DO   DUP I C@ SWAP C! 2+
 9                                  LOOP DROP ;
10 : COLOR-CURSOR   C# @ L# @ 4 + PLOT 1+ C! ;
11 : MARK           34 COLOR-CURSOR ;
12 : -MARK           2 COLOR-CURSOR ;
13 : CLRS           6 EMIT 2 EMIT 15 EMIT 12 EMIT ; ( GREEN TEXT )
14 : VHTAB          3 EMIT EMIT EMIT ;
15
```

SCR # 43

```
 0 ( SCREEN EDITOR DISPLAY )
 1
 2 : SHOW-PLACE    1 4 VHTAB ." SCR# " SCR @ 2 .R
 3                      ."      LINE# " L# @ 2 .R
 4                      ."      CHAR# " C# @ 2 .R
 5                 21 0 VHTAB ." PAD>"  24 0 VHTAB ;
 6 : SHOW-PAD      PAD DUP 1+ 0 22 PLOT ROT C@ SMOVE ;
 7 : SHOW-SCREEN   CLRS  SHOW-PLACE  3 DASHES 20 DASHES
 8                 SCR @ BLOCK 0 4 PLOT 1024 SMOVE MARK SHOW-PAD ;
 9 : SHOW-LINE     SCR @ BLOCK L# @ 64 * + ( LINE START ADDRESS )
10                 0 L# @ 4 + PLOT 64 SMOVE  MARK ;
11 : +MARK         -MARK  C# @ + 63 AND C# !  L# @ + 15 AND L# !
12                   MARK  SHOW-PLACE ;
13 : HOME       -MARK 0 C# ! 0 L# !  MARK  SHOW-PLACE ;
14 : NEW-LINE   -MARK 0 C# ! L# @ 1+ 15 AND L# ! MARK SHOW-PLACE ;
15
```

SCR # 44

```
 0 ( SCREEN EDITOR  INSERT DELETE )
 1
 2 : COPY-LINE     LINE PAD 1+ C/L DUP PAD C! CMOVE SHOW-PAD ;
 3 : INSERT-LINE   DUP 1 - 14 DO I LINE I 1+ -MOVE -1 +LOOP WIPE ;
 4 : PASTE-LINE    DUP INSERT-LINE PAD 1+ SWAP -MOVE ;
 5 : DELETE-LINE   DUP COPY-LINE DUP 15 = IF DROP ELSE 15 SWAP
 6                 DO I 1+ LINE I -MOVE LOOP ENDIF 15 WIPE ;
 7 : DELETE-CHAR   C# @ IF <SLIDE BL C/L 1 - L# @ LINE + C!
 8                        0 -1 +MARK UPDATE ENDIF ;
 9 : INSERT-CHAR   C/L  C# @ - 1 > DUP  IF SLIDE> ENDIF
10                     K# @ CHAR C! IF 0 1 +MARK ENDIF  UPDATE ;
11
12 : CHANGE-SCREEN   ." NEXT SCREEN NUMBER = "
13                  0 BEGIN KEY DUP DUP EMIT 13 = 0=
14                    WHILE 48 - SWAP 10 * +
15                    REPEAT DROP SCR ! ;
```

SCR # 45

```
 0 ( NON-CURSOR KEY TABLE )
 1
 2 : INSERT/DELETE     K# @ CASE
 3     ( INSERT LINE KEY )     3 OF L# @ INSERT-LINE 1 ENDOF
 4     ( DELETE LINE KEY )     4 OF L# @ DELETE-LINE 1 ENDOF
 5     ( INSERT CHAR KEY )     5 OF L# @  PASTE-LINE 1 ENDOF
 6     ( ERASE LINE KEY  )    11 OF L# @   COPY-LINE 0 ENDOF
 7     ( DELETE CHAR KEY )   127 OF      DELETE-CHAR 0 ENDOF
 8     ( ANY OTHER KEY   )   INSERT-CHAR 0 SWAP
 9                          ENDCASE ;
10 ;S
11
12
13
14
15
```

SCR # 46

```
 0 ( CURSOR KEY TABLE )
 1
 2 : STEP   BEGIN 33252 C@  ( STEP WHILE KEY DOWN )
 3          WHILE OVER OVER +MARK 100 0 DO LOOP ( TIME DELAY )
 4          REPEAT DROP DROP ;
 5
 6 : PROCESS        0 K# @ CASE
 7      ( CURSOR RIGHT  )    25 OF 0  1 STEP ENDOF
 8      ( CURSOR LEFT   )    26 OF 0 -1 STEP ENDOF
 9      ( CURSOR UP     )    28 OF -1 0 STEP ENDOF
10      ( CURSOR DOWN   )    10 OF  1 0 STEP ENDOF
11      ( HOME KEY      )     8 OF      HOME ENDOF
12      ( RETURN KEY    )    13 OF  NEW-LINE ENDOF
13      ( ANY OTHER KEY )    INSERT/DELETE ROT ROT DROP
14                         ENDCASE
15            IF SHOW-SCREEN ELSE SHOW-LINE ENDIF ;
```

SCR # 47

```
 0 ( SCREEN EDITOR MAIN LOOP )
 1
 2 : SEDIT    CLRS  CLEAR-PAD ." SCREEN EDITOR"
 3            CR CR CHANGE-SCREEN
 4            BEGIN HOME SHOW-SCREEN
 5              BEGIN
 6                KEY DUP DUP K# ! 27 = SWAP 12 = OR 0=
 7              WHILE PROCESS ( KEY NOT ESCAPE OR ERASE PAGE )
 8              REPEAT
 9              K# @ 27 = 0=
10            WHILE CHANGE-SCREEN ( ERASE PAGE KEY )
11            REPEAT
12            HOME -MARK FLUSH ; ( ESCAPE UPDATES DISK AND EXITS )
13 ;S
14
15
```

SCR # 6

```
 0 ( CASE, ARRAY )
 1
 2 : CASE   ?COMP CSP @ !CSP 4 ; IMMEDIATE
 3
 4 : OF     4 ?PAIRS COMPILE OVER COMPILE = COMPILE OBRANCH
 5          HERE 0 , COMPILE DROP 5 ; IMMEDIATE
 6
 7 : ENDOF     5 ?PAIRS COMPILE BRANCH HERE 0 , SWAP 2
 8             [COMPILE] ENDIF 4 ; IMMEDIATE
 9
10 : ENDCASE   4 ?PAIRS COMPILE DROP BEGIN SP@ CSP @ = 0=
11         WHILE 2 [COMPILE] ENDIF REPEAT CSP ! ; IMMEDIATE
12
13 : ARRAY   <BUILDS DUP C, * ALLOT DOES> ROT 1 - OVER C@ * + + ;
14 ;S
15
```

REPRINT TO CLARIFY FIG. 2

ANIMATION *(Concluded)*

CHRIS ZERR *10932-156th Court NE*
Redmond, WA 98052
Compuserve: 71445,1240

In the last article we plotted a rectangle on the screen and moved it through a blue field at various speeds using OSTR to print db strings.. Let's try a more appropriate figure this time, constructed from plot blocks, increase the action, and poke the animaton directly into screen memory.

We will construct an 'alien' with a missile launcher, moving feet, and piercing 'eyes' that blink menacingly at us, all through the use of plot blocks poked directly into screen memory. Constructing the figure of the alien is a matter of laying out the plot blocks and identifying the code required to illuminate them in the desired way.

CONSTRUCTING THE ALIEN'S PLOT BLOCKS.

Figure 1 shows a single plot block, consisting of eight controllable portions .. A1 through A4 and B1 through B4. These eight portions coincide with the eight bits of a byte. The portions of the plot block that will be illuminated will depend on whether or not the corresponding bit in the plot byte is set. To illuminate each of the four corners of a plot block in turn, we would set our plot byte equal to each of the following numbers:

```
Upper left corner : set A1 bit (bit 0)= PLOT value 1
Lower left corner : set A4 bit (bit 3)= PLOT value 4
Upper right corner: set B1 bit (bit 4)= PLOT value 16
Lower right corner: set B4 bit (bit 7)= PLOT value 128
```

So to construct a plot block, we shade in the bit sections, add all the individual bit values to the find the total plot value, and we have it! But we are not quite finished, because plotting the plot value alone will not quite do it. The CCI code must also be assigned, and for a plot block it must have a value between 128 and 255. The effect of the CCI code will be the same, by and large, as it would be for an ASCII character attribute, but the number will be increased by 128. For example, to set the color yellow, we issue PLOT 6,3 for ASCII, but for a yellow plot block it becomes 3 + 128 or 131. (Now we know how to peek at a screen memory location to tell if there is an ASCII character there or a plot block.)[1]

You may easily experiment with plot blocks and their CCI codes in BASIC to observe the diffference various CCI codes make. Whatever character we design must be derived from a combination of plot blocks within the possibilities available to us. A crude sketch on graph paper is the easiest way, possibly, and permits an instant translation into the correct plot codes.

Assembly Language Programming

Part XVI

Fig 1.

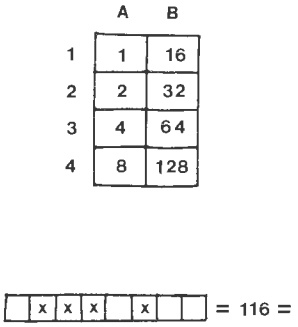
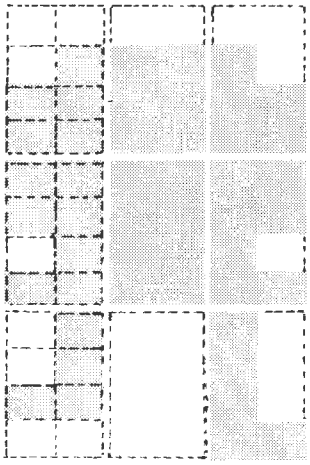


Fig 2.



Our alien has two different forms in this program. In the first form his left foot is raised; in the other his right foot is raised. Figures 2 and 3 illustrate the composition of these two figures. The outline of the plot blocks has been exaggerated in Fig 2 so you can see how the plot codes for this figure were derived. The upper left plot block has bits 2, 3, 5, 6, and 7 set, giving a value of 4 + 8 + 32 + 64 + 128 or 236 for this plot block. You should try to derive the plot codes for the remaining blocks as an exercise. The lower left block is shown with the proper bits set for you in Figure 2.

The plot blocks from Figure 2 contain the values 236, 238, 206, 251, 191, 116, 0, 143. The zero (0) count is necessary to act as a filler which says a null block is to be plotted in that location. In actual practice, each of these plot blocks must also have a CCI code following it, so a completed command byte string for Figure 2 would be 236, 131, 238, 131, 206, 131, etc.

We may mix plot blocks with ASCII characters in the plot command string, and our alien requires this to plot his 'eyes', which are quotes in ASCII. You will find them in the plot string from this Basic example, which plots the alien in one of his two forms on the screen:

```
100  X=30780 : REM INITIAL SCREEN MEMORY LOCATION
110  FOR A=1 TO 3 : REM PAINT 3 VERTICAL PAIRS OF BLOCKS
120  FOR B=1 TO 6 : REM THREE PAIRS OF BYTES FOR X AXIS
130  READ Z : REM GET POKING DATA
140  POKE X,Z : REM POKE ALT PLOT CODES & CCI
150  X=X+1
160  NEXT B
170  X=X+122 : REM PLOTS DOWN ONE BLOCK ON Y AXIS
180  NEXT A : REM PLOT NEXT VERTICAL PAIR
190  END
200  DATA 236,131,238,131,206,131,251,131,34,25,191,131
210  REM  34,25' IS ASCII ["] IN RED
220  DATA 116,131,0,0,143,131 : REM '0,0' IS FILLER PAIR
```

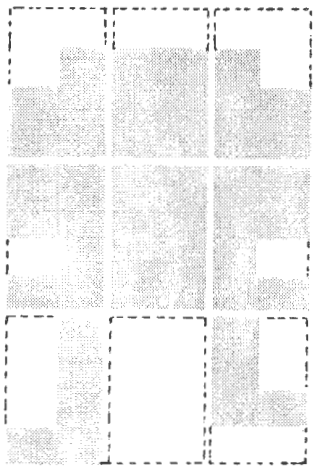


Fig 3.

This program may also be written as one line:

```
10 PLOT 3,30,10,19,2,254,236,238,206,255,3,30,11,2,254,251,
        255,6,25,34,6,3,2,254,191,255,3,30,12,116,0,143,255
```

All we have done so far is paint one of the two alien forms. The numbers for the other form, including the 'eyes', and the CCI code are:

```
236,131,238,131,206,131,251,131,
34,27,191,131,248,131,0,0,71,131
```
(See Figure 3.)

The principle is very simple. We now have a subject ready for use.

BORDERS.

Every game has a border, or a boundary, but it isn't always a visible one. For example, the edges of the CRT are always a finite boundary for any plotting. The programmer needs a border, visible or not, so he can check to see if his animaton has gone as far in one direction as it can go. If such a check were not possible, animation would stop when the CRT boundary were reached, or an animaton might leave the screen when this was not desired. If you look at an invisable boundary (in a properly programmed routine) you will probably see an ASCII character from 1 to 255, but a CCI code of zero (0), so it doesn't show. Whatever the choice of boundary encoding, it must be selected so it is unique to the boundary, and not used anywhere else on the screen. It is entirely possible to make each of the four boundary codes different so the program knows exactly which boundary it has reached. This is much more easily done if the actual boundary is invisible, for otherwise the encoding changes would have to be visible sooner or later (if the CCI code were not zero, for example.)

Our program uses a border made of the hatch character (96 decimal) and a CCI code of 38 (blue on white). We are not going to let the alien move off the screen, so our border encoding will be a warning to us that a boundary is present, and that the direction of the next move must be changed.

The border is poked directly into screen memory by the subroutines TANDB and SIDES, beginning at memory address 7000H. The blocks representing the animaton are also poked directly into screen memory. The inital location of these blocks is established in the nineth line of START, where location 30790 is recorded in LOC. You may change this location, provided you replace it with a valid screen memory address.

MOVEMENT

We will use keyboard scanning to let the operator move the alien in one of four directions and to fire the alien's missile.[2] The scanning procedure makes it unnecessary for the program to wait for a key press before proceeding with the automation; otherwise, the left foot might just stay 'put' until we pressed a key. The technique we use was described by Steve Perrigo in Colorcue, June/July, 1982. The following assignments have been made to direct alien motion:

"8" = move up, "2" = move down, "4" = move left, and "6" = move right.

Entering a "5" will 'fire' the missile. This pattern coincides with most joystick wiring. If your joystick uses the arrow keys, these may be substituted for the numbers readily. Notice that diagonal movement is possible in our routine by pressing two keys at the same time (or moving the joystick between two absolute directions.)

FIRING THE MISSILE

When the "5" key is pressed, the routine FIREX is called. It checks first to see if a missile is already being fired. If so, the routine is aborted; otherwise FIREX finds the alien's current position and direction of movement. We are choosing to fire the missile in the direction of alien travel, and so we must plot a course for the missile. To make things 'neat' we fire the missile from the center of the alien. You will notice that the MISSLE routine is called several times for each single movement of the alien. This is necessary to prevent the alien from moving right along with the missile. The missile will travel until it hits the border.

The code in Listing I is well documented, and, as with all assembly code, it needs to be read to be understood. You might try experimenting with the time delay at label MOVEIT. You might also try moving the alien on a totally blue background (as with the rectangle, last time.) This kind of experimenting can be very inspirational when designing a game. As an additional exercise, find a way to remove the call to OSTR, and make this program completely FCS-version independent.

We also know, now, how to expand the program developed in the last last article, making use of a border sensor, and introducing a method by which the rectangle 'knows' when it is about to tred onto the blue field, and when it is retracing a previous path. You may reprogram that listing to guide the rectangle through its first complete connecting loop, after which it will maneuver itself without straying from the initial path. □

[1] The CCI codes are discussed thoroughly in the CCII Programmer's Manual. Also see 'Color Graphics' by David Suits.

[2] Note: This routine will not operate properly on a 3651. To duplicate the function, use the GTCHA routine and CHRINT setup from David Suit's input routine in Colorcue. Write to Colorcue for details if you need some help.

```
          ;LISTING I.
;
;*****  A GAME SIMULATION  by Chris Zerr. 6/30/84  *****

       OSTR    EQU     182AH   ;For v8.79,9.80
                               ;See ROM tables for 6.78
       ORG     9000H

;---------------------- SETUP PROCEDURES --------------------

START: XRA     A               ;SET A=0
       OUT     8               ;DISABLE KEYBOARD
       STA     DIREC           ;CLEAR DIRECTION
       STA     SWITCH          ; AND SWITCH
       LXI     H,0             ;MISSILE LOCATION
       SHLD    FLOC            ; SET TO ZERO
       LXI     H,CLEAR         ;CLEAR SCREEN
       CALL    OSTR
                               ;INITIAL POSITION FOR ALIEN
       LXI     H,30780         ; START SOMEWHERE IN THE
       SHLD    LOC             ; MIDDLE OF THE SCREEN

       LXI     H,7000H         ;LETS MAKE A BORDER
       CALL    TANDB           ;TOP & BOTTOM - DO TOP
       LXI     H,7F80H
       CALL    TANDB           ;DO BOTTOM
       LXI     H,7000H
       CALL    SIDES           ;DO LEFT SIDE
       LXI     H,707EH
       CALL    SIDES           ;DO RIGHT SIDE

;---------------------- MAIN PROGRAM ----------------------

REPETE: LHLD   LOC             ;LOAD HL WITH ALIEN LOCATION
       LDA     SWITCH          ;GET CHANGE SWITCH
       ORA     A
       JZ      CLOSE           ;IF ZERO, USE LEFT FOOT DOWN
       LXI     D,ALIENR        ; ELSE ALIEN RIGHT
       XRA     A               ; FOOT DOWN
       STA     SWITCH
       JMP     MOVEIT

CLOSE: LXI     D,ALIENL        ;ALIEN LEFT FOOT DOWN
       MVI     A,255           ;SET THE FLAG SO WE CAN
       STA     SWITCH          ; ROTATE THE ALIEN MOVEMENT

MOVEIT: CALL   MOVE            ;MOVE THE ALIEN
       MVI     C,70            ;DELAY ABOUT 1.5 SECONDS
AA:    MVI     B,255           ;   ( DELAY ROUTINE )
LL:    DCR     B               ;
       JNZ     LL              ;
       DCR     C               ;
       JNZ     AA              ;
       CALL    KEY             ;CHECK KEYBOARD INPUT
       JMP     REPETE          ; AND DO IT AGAIN AND AGAIN!

; ------------------- END OF MAIN PROGRAM --------------------

;---------------------- SUBROUTINES -----------------------

KEY:   MVI     A,10    ;CHECK FOR "5"
       OUT     7       ;SEND IT OUT
       IN      1       ;GET STATUS
       CPI     254     ;IS IT 5??
       CZ      FIREX   ;YES..PUT MISSILE IN MISSILE TABLE
       MVI     A,7     ;CHECK FOR "8"
       OUT     7
       IN      1
       CPI     254
       CZ      UP      ;YES...MOVE THE ALIEN UP
       CALL    MISSLE
       MVI     A,13    ;CHECK FOR "2"
       OUT     7
       IN      1
       CPI     254
       CZ      DOWN    ;YES...MOVE THE ALIEN DOWN
       MVI     A,11    ;CHECK FOR "4"
       OUT     7
       IN      1
       CPI     254
       CZ      LEFT    ;YES...MOVE THE ALIEN LEFT
       MVI     A,9     ;CHECK FOR "6"
       OUT     7
       IN      1
       CPI     254
       CZ      RIGHT   ;YES...MOVE THE ALIEN RIGHT
       CALL    MISSLE  ;IF THERE IS A MISSILE TO MOVE,
                       ; MOVE IT AGAIN...
       RET

UP:    LHLD    LOC             ;PROCESS ALIEN TO GO UP
       LXI     D,ERASE         ; BUT FIRST ERASE
       CALL    MOVE            ;  THE OLD ALIEN
       LHLD    LOC
       LXI     D,0FF80H        ;SUBTRACT 128 FROM HL REG
       DAD     D
       MOV     A,M             ;IF WE ARE AT THE BORDER
       CPI     96
       RZ                      ; WE DON'T ADVANCE THE ALIEN
       SHLD    LOC
       CALL    MCHECK          ;CHECK TO SEE IF A MISSILE
                               ; IS TRAVELLING-
       RNZ                     ; IT IS SO FORGET IT!
       MVI     A,1             ; IT ISN'T SO-
       STA     DIREC
       RET

DOWN:  LXI     D,ERASE         ; SO, YOU WANT TO MOVE
       LHLD    LOC             ;  DOWN!?  WELL FIRST
       CALL    MOVE            ;  ERASE THE OLD ALIEN
       LHLD    LOC
       LXI     D,128           ;GET OUR DOWNWARD OFFSET
       DAD     D               ; WE MUST CHECK
       DAD     D               ; ONE BELOW THE
```

```
        DAD     D               ; BOTTOM OF THE
        MOV     A,M             ; ALIEN -
        CPI     96              ; TO SEE IF WE ARE AT
        RZ                      ; THE BORDER -
        LXI     D,0FF80H        ;WE'RE NOT!
        DAD     D               ;SINCE WE ONLY NEED TO MOVE
        DAD     D               ; ONE 128 BYTES, WE MUST
        SHLD    LOC             ; REMOVE THE EXTRAS FROM ABOVE
        CALL    MCHECK          ;MOVE A MISSILE (IF ANY)
        RNZ                     ;IF A MISSILE IS IN PROGRESS
                                ; DON'T BOTHER WITH THIS NEW
                                ; LOCATION
        MVI     A,2             ; OUR DIRECTION IS 2 FOR DOWN
        STA     DIREC
        RET

LEFT:   LXI     D,ERASE         ;AGAIN ERASE THE OLD LOCATION
        LHLD    LOC
        CALL    MOVE            ;DO THE ERASE
        LHLD    LOC             ;REGAIN OUR CURRENT LOCATION
        DCX     H               ;DOWN BY ONE
        MOV     A,M             ;THIS IS OUR CCI COLOR CODE
        CPI     38              ;IF WE ARE HERE, WE WILL NOT
        RZ                      ; MOVE
        DCX     H               ;BECAUSE WE CAN MOVE
        SHLD    LOC             ; BUMP IT DOWN AGAIN AND SAVE
                                ; THIS PUTS US ON AN EVEN X,Y
                                ; OR "CHARACTER,CCI" CODE
        CALL    MCHECK          ;CHECK FOR MISSILE IN PROGRESS
        RNZ                     ;YES..SO RETURN TO THE CALLER
        MVI     A,3             ;OUR DIRECTION IS 3
        STA     DIREC           ; WHICH IS LEFT
        RET

RIGHT:  LXI     D,ERASE
        LHLD    LOC
        CALL    MOVE
        LHLD    LOC             ;GET OUR LOCATION
        LXI     D,0006          ;JUST 6 BYTES TO THE RIGHT
        DAD     D               ; IS ALL WE WANT..NOT THE WORLD
        MOV     A,M             ;DID WE HIT THE BORDER?
        CPI     96
        RZ                      ;SORRY CHARLIE..ONLY THE BEST
        LXI     D,0FFFCH        ;NOW BACK UP 4 BYTES
        DAD     D               ;DO IT
        SHLD    LOC             ;SAVE OUR NEW HOME
        CALL    MCHECK
        RNZ
        MVI     A,4             ;OUR DIRECTION IS 4
        STA     DIREC           ; WHICH IS RIGHT...RIGHT?
        RET

FIREX:  CALL    MCHECK          ;IF A MISSILE IS STILL IN
        RNZ                     ; PROGRESS..LEAVE
        LDA     DIREC           ;GET ITS DIRECTION
        ORA     A               ; IF ZERO
        RZ                      ; THERE IS NOWHERE TO GO
        LHLD    LOC             ;THE FOLLOWING WILL
        CPI     1               ; POSITION THE HL REG
        JZ      POSUP           ; TO POINT AT A LOCATION
        CPI     2               ; IN WHICH THE MISSILE
        JZ      POSDN           ; WILL LEAVE THE ALIEN
        CPI     3
        JZ      POSLF
        CPI     4
        JZ      POSRT
        RET

;)))---) HOW THE FOLLOWING ROUTINE WORKS
;
;       WHAT I HAVE CHOSEN IS TO LAUNCH A MISSILE
; FROM THE CENTER OF THE ALIEN IN WHICH THE DIRECTION
; OF THE ALIEN IS OR WAS PRESENTLY MOVING.  SO FOR OUR
; FIRST POSITION, POSUP, WE GET OUR CURRENT LOCATION,
; THEN ADD 2 BYTES TO IT.  THIS WILL THEN BE THE FIRST
; LOCATION OF THE LAUNCHED MISSILE.

POSUP:  INX     H               ;POSSIBLE LOCATIONS ARE:
        INX     H
        JMP     SAVPOS
                                ;         ^
POSDN:  LXI     D,0258          ;       PCPCPC
        DAD     D               ;     < PCPCPC >
        JMP     SAVPOS          ;       PCPCPC
                                ;         V
POSLF:  LXI     D,0128
        DAD     D               ;HL IS ALWAYS POINTING
        JMP     SAVPOS          ; TO THE FIRST POSITION
                                ; OF THE ALIEN
POSRT:  LXI     D,0132
        DAD     D

SAVPOS: SHLD    FLOC            ;SAVE THE POSITION
        RET

MISSLE: LHLD    FLOC            ;PLOT THE MISSILE
        MOV     A,H             ;IF HL ZERO,
        ORA     L               ; THEN THERE IS NO
        RZ                      ;  MISSILE TO PLOT
        LDA     DIREC
        CPI     1
        JZ      FIREUP
        CPI     2
        JZ      FIREDN
        CPI     3
        JZ      FIRELT
        CPI     4
        JZ      FIRERT
        RET
```

```
FIREUP: CALL    MERASE          ;ERASE THE OLD MISSILE
        DCX     H               ;HL RETURNED UP'ED BY ONE
        LXI     D,0FF80H        ;SUBTRACT 128 BYTES
        DAD     D
        SHLD    FLOC            ;SAVE IT
        MOV     A,M
        CPI     96
        JZ      NOMORE          ;CAN'T PASS THE BORDER
        MVI     M,110           ;PLOT THE NEW MISSILE
        INX     H
        MVI     M,1             ;USE COLOR OF RED
        RET

FIREDN: CALL    MERASE          ;ERASE THE OLD MISSILE
        DCX     H
        LXI     D,0128          ; TO PLOT IT DOWN 1
        DAD     D               ; ADD 128 BYTES
        SHLD    FLOC
        MOV     A,M
        CPI     96              ;WE ARE AT THE BORDER
        JZ      NOMORE          ; SO RETURN
        MVI     M,110
        INX     H
        MVI     M,2             ;USE COLOR GREEN
        RET

FIRELT: CALL    MERASE          ;ERASE OLD MISSILE
        DCX     H               ;TO GO LEFT
        DCX     H               ; WE MUST SUBTRACT 3
        DCX     H               ; BECAUSE MERASE ADDS ONE
        MOV     A,M
        CPI     96
        JZ      NOMORE          ;CAN'T GO THROUGH THE BORDER
        SHLD    FLOC
        MVI     M,105           ;LOWER CASE I
        INX     H
        MVI     M,5             ;COLOR MAGENTA
        RET

FIRERT: CALL    MERASE          ;ERASE
        INX     H               ;INCREASE HL BY ONE
        MOV     A,M
        CPI     96
        JZ      NOMORE          ;SORRY PAL
        SHLD    FLOC            ;IT'S GOOD! SAVE IT
        MVI     M,105
        INX     H
        MVI     M,7             ;COLOR WHITE
        RET

NOMORE: LXI     H,0             ;MISSILE NO MORE, SO
        SHLD    FLOC            ;CLEAR IT FROM THE
        RET                     ; MISSILE TABLE

MCHECK: LHLD    FLOC            ;CHECK IF MISSILE IS
        MOV     A,H             ;IN PROGRESS
        ORA     L
        RET

MERASE: MVI     M,' '           ;ERASE THE OLD MISSILE
        INX     H
        MVI     M,0
        RET

MOVE:   MVI     B,3             ;PLOT THE ALIEN ON THE SCREEN
ROW:    MVI     C,6             ;THIS ROUTINE DOES THE ACTUAL
COLUMN: LDAX    D               ;PLOTTING OF THE ALIEN.  H&L
        MOV     M,A             ; ARE PASSED TO THIS ROUTINE
        INX     D               ; AND THE ALIEN IS THEREFORE
        INX     H               ; PLOTTED.
        DCR     C
        JNZ     COLUMN
        PUSH    B
        LXI     B,0122          ;AT THE END OF 6 WE MUST
        DAD     B               ; POSITION OURSELVES AT THE
        POP     B               ; NEXT LINE.
        DCR     B               ;EXAMPLE:
        JNZ     ROW
        RET                     ;HL=7000H
                                ; AT THE END OF 6 COUNTS,
                                ; HL=7006H, SO WE ADD 7AH(122)
                                ; HL=7080H, WHICH IS RIGHT
                                ; BELOW WHERE HL WAS (7000H)

;PAINT BORDER-----------------------------------------------


TANDB:  MVI     C,64            ;TOP AND BOTTOM OF BORDER
        MVI     M,96
        INX     H
        MVI     M,38
        INX     H
        DCR     C
        JNZ     TANDB+2
        RET

SIDES:  LXI     D,128           ;SIDES OF BORDER
        MVI     C,31
SIDE1:  MVI     M,96
        INX     H
        MVI     M,38
        DCX     H
        DAD     D
        DCR     C
        JNZ     SIDE1
        RET
```

```
;---------------------- DATA STORAGE ----------------------

SWITCH: DS      1       ;ALIEN SWITCHING
DIREC:  DS      1       ;MISSILE DIRECTION
FLOC:   DS      2       ;MISSILE LOCATION
LOC:    DS      2       ;ALIEN  LOCATION

CLEAR:  DB      6,0,15,27,24,12,3,64,0,239

;       ALIEN WITH LEFT FOOT DOWN

ALIENL: DB      236,131,238,131,206,131
        DB      251,131,34,25,191,131
        DB      116,131,0,0,143,131

;       ALIEN WITH RIGHT FOOT DOWN

ALIENR: DB      236,131,238,131,206,131
        DB      251,131,34,28,191,131
        DB      248,131,0,0,71,131

;       DATA TO ERASE THE ALIEN

ERASE:  DB      32,0,32,0,32,0
        DB      32,0,32,0,32,0
        DB      32,0,32,0,32,0

        END     START
```

Colorcue will be saying 'goodby' .....

The next issue of Colorcue, Vol VI, No 6, will be the last for this seven year old publication. Diminishing support is the primary reason, along with greatly increased publication costs. We might have simply economized on our formatting and means of distribution, but it has been clear that what is most needed is a consolidation of effort among the various CCII organizations. With consolidation, we have an opportunity to maintain our activity and still provide the necessary publication services to subscribers.

The Rochester User Group is the most appropriate center for this consolidation. Its publication, DATA CHIP, and its fine user library provide subscribers with a wealth of continuing materials for the Intecolor computers.

Beginning early in 1985, CHIP will be expanded to include materials normally published in Colorcue. Authors will continue to write for publication in CHIP. With this added contribution, CHIP can be a more frequent and expansive publication, which it well deserves to be.

The price of membership in the Rochester User Group has been increased from $10 to $15 per year ($20.00 USA for overseas) to cover the expanded publication of CHIP. I urge you to lend your total support to this effort. Send your check for $15.00 today to

GENE BAILEY: 28 Dogwood Glen, Rochester, NY 14625

Your prompt response is necessary for good planning in Rochester, and to be sure you don't forget and therefore miss some of the exciting material ready for release (including a new article by Ben Barlow on expanding the disk capacity of the CCII!) Send materials for publication in CHIP to Rick Taubold, 197 Hollybrook Road, Rochester, NY 14623.

The last issue of Colorcue will contain a complete index of the major CCII publications, and a fine set of articles recently submitted for publication. It will, no doubt, be a large issue, and will also, with certainty, be late. There is a large amount of midnight oil to be burned before it is ready. Meanwhile, I hope we will see our first issues of the expanded CHIP newletter. As always, these are your publications, and your active support is required to make them valuable to CCII users. The end is not yet in sight, friends!

A reminder, too, that Australia has been very active lately in the CCII department. You will enjoy a subscription to CUVIC. See Colorcue, Vol VI, No 3, page 31 for membership information.

COLORWORD *A re-review*

DOUG GRANT

Doug Grant, the librarian of CUWEST in Australia, sent a very enthusiastic letter to David Ricketts, the author of the GEMINI printer review in the last issue. His subject was COLORWORD, the word processor distributed by PPI in Australia. Doug feels the COLORCUE review was less laudatory than it might have been and points out specific features of the software that have proven valuable to him.

We are pleased to excerpt it here, with permission, and print a few examples of type from Doug's Epson printer.

"(Dear David,)..We are about even in our standard of Tightwadishness. I took a good look at Comp-U-Writer long ago and decided that I didn't want a WP which couldn't use all the codes of the Epson printer. I purchased Wordthis and Wordthat and Thisaword and Thataword and probably got to around Comp-U-Writer's price in total anyway. Now, we have a very good programmer in our group, Chris Teo who came up with COLORWORD, and if you don't send off the $50 to PPI as soon as you can drag out your chequebook then you just don't deserve to own a printer as good as the Gemini appears.

"(The enclosed page is a printout of a disk file I use to demonstrate COLORWORD's use of printer control codes) and you will see that with expert use...you can really get a message through. All of these codes are added to your text following your pressing INSERT CHAR, of course, and this includes CTL @ and CTL G. ESC appears as '[', in blue, and CTL @ and CTL G as @ and G in blue. You will find out for yourself about the 'A7' and etc.

"Have a good look at the review in Colorcue of May/June 1984 but don't let it deter you. We have members who have both Comp-U-Writer and COLORWORD v4.5 who rarely use C-U-W except when they require two column work or lots of text centering. COLORWORD misses these two features, but Chris will probably come up with that later. The only thing you may get a bit excited about is that when you go back in the text a little way to add a few more words here and there, the word wrap still does its job but sometimes moves the remainder of the line down a line, and if you move the cursor past that line you may think you have lost a line or so. No worries though, you simply go back to the beginning of the paragraph and hit CMD R to reform the paragraph. Even if you didn't do this, the printout would still be correct, as you have to hit RETURN twice to force new lines anywhere and you can't lose any text without really trying.

"The underline is a true underline and not a series of dashes and in editing you can use the colour pad to whip through by paragraphs or lines in either direction. There is no 'undelete' so you must make fewer errors in this regard. If you accidentally hit AUTO you go off into a 'search' mode but don't panic, just hit AUTO a couple more times and you are back to the text again.

"You can set up your output baud rate, double spacing, spaced print, etc. at each sitting, but there is a provision for you to make up your own text file incorporating all the parameters you wish, and load it up when you start work. I have 24 good sized letters on one side of the disk (initialized to 07) which I have in the drive at this moment, and I can exit the text and delete any of them if I run out of disk space; or put another disk in. At the top of the screen I see that I still have 12701 characters of space left in this text if I want to go on all night at my rate of typing, but I won't. Oh well; back to the cot for another read of COLORCUE.

"All the best to you from,

Doug Grant."

INTECOLOR BULLETIN BOARD FOR CCII

Steve Reddoch's proposal for a bulletin board for the CCII has not had any response from readers. The Santa Barbara subscriber has prepared a program for free distribution to CCII owners in machine language to service both 300 and 1200 baud. Our two hundred users are not enough to support such a project, it seems, and there are many of us who can sympathize with Steve's disappointment that this project has not seen the light of day. But there has been progress! Intecolor Corporation has recently announced the establishment of a Bulletin Board for all its products, including the CCII, 3600 and 8000 computers. It is located in the Northwest USA. The telephone number is (206) 4833460, and the Sysop ("System Operator") is Bob Morgan, an Intecolor dealer. While it is a CP/M board, TERMII software should work well. We encourage you to take advantage of this new opportunity. The board will also include news on Intecolor's DataVue computers. These are being offered at a special price to CCII owners. They are CP/M machines, with built-in hard disks and full CP/M features. The computer is available without a terminal, and the CCII may be used as a terminal with it. Software is available for the terminal conversion from Rick Taubold and Tom Devlin, which they developed with Myron Steffy for interfacing with the Morrow computer line.

KEYBOARD UPGRADES

It's not too late to upgrade your CCII keyboard to the full 117 keys. Howard Rosen (PO Box 434, Huntington Valley, PA 19006) has enlarged keyboard covers available for $20.00 and switches for $2.00 each. Key caps may be ordered from Arkay Engraving (see last issue). Intelligent Systems Corporation in Huntsville has some of the same materials. Give them a call at (205) 881-3800 or write 12117 Comanche Trail, Huntsville, AL 35803.

John Ker in Los Angeles is a network subscriber. His Compuserve network address is 71735, 1673, and his Source address is TCP733. Welcome aboard, John! I've enjoyed our modem talks.

Control CODES for **COLORWORD** and EPSON with GRAFTRAX PLUS ROM Set.

ESC 4 turns on *Italics*
ESC 5 turns off Italics

A7 on turns on enlarged characters
BLUE KEY turns off enlarged characters

BL/A7 OFF turns on condensed characters
GREEN KEY turns off condensed characters

ESC E turns on **emphasized** characters
ESC F turns off emphasized characters

ESC G turns on **double** character
ESC H turns off double character

ESC S turns on subscript mode
ESC S followed by CONTROL @ turns on Superscript mode
ESC H turns off Subscript or Superscript mode.

ESC O gives 8 lines per inch
ESC 2 gives 6 lines per inch

ESC - turns on Underline
ESC - followed by CONTROL @ turns off Underline

Control G Sounds the Printer Bell.

ESC @ Initializes the Printer and resets the top of form position.

COMBINATIONS OF VARIOUS CONTROL CODES GIVE **VARIOUS** RESULTS

tiny — PASCAL (conclusion)

This final part of the series is dedicated to the 'bread-n-butter' Tiny-Pascal commands. We can not hope to give a textbook treatment of these commands, so we will restrict ourselves to some typical examples of their use. Your familiarity with other programming languages, and the use of a good Pascal tutor will enable you to understand and apply our contents on the CCII.

A NEW SCREEN EDITOR

A new tool is now available to help in the exploration and implementation of Tiny-Pascal on the Compucolor in the form of a full screen editor, written by Bill Greene, and housed in the Chip Users Group Library. This editor significantly reduces the task of entering and editing both Forth and Tiny-Pascal programs on Forth screens. The screen editor, along with two Forth disassemblers and an 8080 assembler are contained in screens on Chip Disk #121. This disk complements Bill Greene's implimentation of Forth, called Forth8, contained on Chip Disk #120. The editor, and any of the other features, may be compiled into the Forth8 core program and saved on disk as an augmented version of Forth (call it FORTHE.) The screen editor is not compatible with Jim Minor's version of Forth on Chip Disk #46. However, the editor can be used to enter and edit Tiny-Pascal screens. On the other hand (all confusion aside, folks), Jim Minor's Tiny-Pascal compiler, which resides in screens in Chip Disk #83, can be compiled into Greene's Forth8 and saved as an augmented version of Forth8 (and call it FORPAS.) It is unfortunate that both the screen editor and the Tiny-Pascal compiler can not be compiled into the same Forth version, due to memory requirements, but using them in two separately augmented versions each will facilitate Tiny-Pascal operations. a) Use FORTHE to enter, edit and save the programs, and b) use FORPAS to compile and run the programs.

If this seems too much, use the line editor already described previously in these articles, or you can send two disks, and $2.00 for return postage, to the author at the above address for these new materials and instructions. Be sure to specify your FCS version, and add $10.00 to have Chip supply the disks. The four sides you receive will contain Chip Disk #120, #121, FORTHE/FORPAS, described above, and all the program examples in the Tiny-Pascal series. Back up the disks when you receive them. [You do realize what a bargain this is! Two languages for the price of none. ED]

TINY PASCAL COMMANDS

It is called 'Tiny-Pascal' because it contains a subset of the fully-implemented version of the language. This subset is sufficiently complete to allow for effective programming in many instances. When viewing Tiny-Pascal you will find many commands similar to those of the more common Basic language dialects, such as IF..THEN..ELSE and FOR..TO..DO. Other commands are found only in very extended Basic dialects, like BEGIN..END, WHILE..DO, REPEAT..UNTIL, CASE..OF, and the PROC and FUNC structures.

You will learn best by working the examples at the computer. Most examples have been limited to a single screen to conserve time for entry and editing. You must carefully observe the punctuation and program structure as it is shown in the listings. Review Part 2 of this series in Colorcue, Vol VI, No 3 to remind yourself of the commands already covered, like WRITE, READ, and NEWLINE.

This version of Tiny-Pascal is intimately intertwined with the Forth language as well, as we have seen, which is used to facilitate program execution. Defining Forth commands in such a way that they may be injected into Tiny-Pascal makes the programs significantly more versatile and useful. In mastering Tiny-Pascal, there is great incentive to master Forth, one language helping to clarify the other. We will demonstrate this in some examples.

The Pascal statement format is very important. Pascal commands are used in both single and compound statements in the listings. Single statements use a command word followed by a semicolon (;). A compound statement begins with a BEGIN and ends with an END and semicolon, surrounding multiple command words and their arguments. (If a compound statement is the last statement before an END, the trailing semicolon may be omitted. This is the only place where the semicolon is optional.) These compound statements form a functional sector, much like a subroutine, and which holds the great strength of a structured language like Pascal. Such a structure makes reading programs, without line numbers, fluent and logical.

To combine the functional sectors into a unified program we group them into a 'block'. The block is terminated with a period (.) to tell the assembler where the program ends. You will see in the examples that compound statements may be nested within a block.

We use two types of comment delimiters in Tiny-Pascal, ( ) for Forth and // for Pascal. Our programs always begin in the Forth domain, so the parentheses will be seen there. Following the Forth word PASCAL, we are in a Pascal environment, and the slash must be used, since parentheses are not defined as comment delimiters in Tiny-Pascal. Note that a space between the delimiter and the comment is required in Forth, but not in Pascal. Finally, following the Tiny-Pascal END statement, the computer is returned to the Forth environment, and rules for Forth will again prevail.

Each of the sample programs explores a different major statement function. The WHILE..DO and REPEAT..UNTIL statements are shown in Listings 1 and 2 respectively. Note that the WHILE and UNTIL are followed by a conditional expression (telling 'while' and 'until' what?) which is either true or false. (Conditional expressions are familiar to Basic programmers from their use in the IF..THEN format.) Examples include 'X = Y', 'U greater.than V', and 'Z not.equal A.' Note also that no colon is used in a conditional expression before the 'equals' sign as it is in the assign statement. In the WHILE..DO statement, a 'true' condi-

tion will cause the statement following the DO (whether single or compound) to be repeated until the conditional expression becomes 'false.' The 'true' condition invokes a looping effect, similar to the FOR..NEXT statement in Basic. When the conditional expression becomes 'false', program control will be advanced to the point following the semicolon (;) just after the statement associated with the DO. The final ';S' in this and other listings is a FORTH command (remember we are back in the FORTH environemt after the END. appears!) This FORTH symbol has various meanings in different circumstances. Here it means simply 'stop execution of screen and go back where you came from.'

Although similar to WHILE..DO, the REPEAT..UNTIL statement has one basic difference; for a given conditional expression, the statements following REPEAT will be executed one time more (than in WHILE..DO) because WHILE..DO tests the conditional before executing the statements, and REPEAT..UNTIL tests the condition after executing the statements. Following a 'false' condition, control is passed to the first statement following the first semicolon to appear after the UNTIL.

Experiment with the programs of Listings 1 and 2 by first loading the FORTH editor. Then, using a disk with blank screens,use the editor to type and save the program on a screen of your choice. Now, load the Tiny-Pascal compiler from its disk and run it. Reload the screen disk. Type the program screen number followed by 'LOAD' and a carriage return. This will compile the program. If the compiler finds an error, the screen editor must be reloaded and the code corrected. When the code has finally compiled correctly, it may be 'run' by just typing the program name (such as 'WHILEEXAMPLE'). Follow similarly for all the other listings.

```
SCR # 15
  0 ( #LISTING 1: WHILE DO EXAMPLE ) PASCAL DECIMAL
  1 PROGRAM WHILEEXAMPLE;
  2 VAR
  3   X, Y, SQUARE : INTEGER;
  4 BEGIN
  5   NEWLINE; WRITE ( 'ENTER NO. TO SQUARE ' );
  6   READ ( #Y );
  7   X := 1;
  8   SQUARE := 0;
  9   WHILE X <= Y DO
 10     BEGIN
 11       SQUARE := SQUARE + Y;
 12       X := X + 1
 13     END;
 14   NEWLINE; WRITE ( #Y, ' SQUARED IS ' , #SQUARE );
 15 END. ;S
```

In contrast to the looping statement types we have just discussed, the IF..THEN..ELSE statement, demonstrated in Listing 3, is designed for branching, just as in BASIC, and the statements which follow are executed only once. In this statement format, a 'true' conditional will cause action by the statements assigned to THEN, passing next beyond any statements allied with the ELSE. Otherwise action will

```
SCR # 16
  0 ( #LISTING 2: REPEAT UNTIL EXAMPLE ) PASCAL DECIMAL
  1 PROGRAM UNTILEXAMPLE;
  2 VAR
  3   X, Y, SQUARE : INTEGER;
  4 BEGIN
  5   NEWLINE; WRITE ( 'ENTER NO. TO SQUARE ' );
  6   READ ( #Y );
  7   X := 1; SQUARE := 0;
  8   REPEAT
  9     BEGIN
 10       SQUARE := SQUARE + Y;
 11       X := X + 1
 12     END;
 13   UNTIL X > Y;
 14   NEWLINE; WRITE ( #Y, ' SQUARED IS ' , #SQUARE );
 15 END. ;S
```

```
SCR # 17
  0 ( #LISTING 3: IF THEN ELSE EXAMPLE ) PASCAL DECIMAL
  1 PROGRAM IFTHENELSEEXAMPLE;
  2 CONST
  3   X = 7;
  4 VAR
  5   Y : INTEGER;
  6 BEGIN
  7   NEWLINE; WRITE ('GUESS A NUMBER (1 TO 10)' );
  8   READ ( #Y );
  9   NEWLINE;
 10   IF (Y <> X)
 11     THEN
 12       WRITE( 'SORRY, WRONG NUMBER ' )
 13     ELSE
 14       WRITE ( 'YES, THE NUMBER IS ' , #X );
 15 END. ;S
```

derive from the statements associated with ELSE, then continue onward. The ELSE portion is optional, which means it may be omitted. In this instance, the IF..THEN rules apply just the same as before (just as they do in BASIC.)

FOR..DO is very similar to the FOR..NEXT loop in BASIC, and is illustrated in Listing 4. DO is usually followed by a compound statement, but single statements are acceptable. What is 'done' lies in the commands between DO and the first semicolon (which is at the end of line 12 in our example). From there, the program advances to the next command. Notice that if one enters '0' or '1' in this program as a response to line 8, the DO function will not be honored because it asks for a 'Y' greater than or equal to '2'. (The correct factorial will be printed in these cases, however.) It is possible to step down in a FOR..DO statement. The syntax in this instance, in line 11, would be....

'FOR X := 2 DOWNTO Y DO'.

A flexible and interesting branch command is the CASE..OF..ELSE command illustrated in Listing 5. Depending on the integer or string value of the expression following CASE, the line following OF which begins with this value will be executed to the first semicolon. An ELSE provision is available with this command also, as illustrated in line 13. Note that an END statement is required to mark the end of the CASE options (see line 12; it terminates the OF path, so to speak). Compare the punctuation in this listing and Listing 3.

```
SCR # 18
  0 ( #LISTING 4: FOR DO EXAMPLE )
  1 PASCAL DECIMAL
  2 PROGRAM FORLOOPEXAMPLE;
  3 VAR
  4   X : INTEGER;
  5   Y : INTEGER;
  6   FACTORIAL : INTEGER;
  7 BEGIN
  8   NEWLINE; WRITE ('ENTER NUMBER FOR FACTORIAL (<8) ');
  9   READ(#Y);
 10   FACTORIAL := 1;
 11   FOR X := 2 TO Y DO
 12     FACTORIAL := FACTORIAL * X;
 13   NEWLINE; WRITE ('FACTORIAL OF ', #Y, ' IS ' , #FACTORIAL);
 14 END. ;S
 15
```

```
SCR # 19
  0 ( #LISTING 5: CASE OF ELSE EXAMPLE ) PASCAL DECIMAL
  1 PROGRAM CASEEXAMPLE;
  2 VAR Y : INTEGER;
  3 BEGIN
  4   NEWLINE; WRITE ( 'ENTER A NUMBER (0-2) ' );
  5   READ ( #Y ); NEWLINE;
  6   IF ((Y > -1) AND (Y < 3))
  7     THEN
  8       CASE Y OF
  9         0 : WRITE( 'THE NUMBER YOU ENTERED IS ZERO ' );
 10         1 : WRITE( 'THE NUMBER YOU ENTERED IS ONE ' );
 11         2 : WRITE( 'THE NUMBER YOU ENTERED IS TWO ' )
 12       END
 13     ELSE
 14       WRITE( 'SORRY, WRONG NUMBER! ' )
 15 END. ;S
```

```
SCR # 20
  0 ( #LISTING 6: PROCEDURE EXAMPLE ) PASCAL DECIMAL
  1 PROGRAM  PROCEDUREEXAMPLE;
  2 VAR  X, Y, SQ : INTEGER;
  3 PROC GETNO;
  4   BEGIN NEWLINE; WRITE('ENTER NO.(<182) TO FIND SQUARE, 0 TO STO
  5 P '); READ(#Y); NEWLINE; END;
  6 PROC SQUARE(U, V : INTEGER);
  7   BEGIN SQ := 0; U := 1;
  8    WHILE (U <= V) DO BEGIN SQ := SQ + V; U := U + 1 END;
  9   END;
 10 PROC PRINTNO;
 11   BEGIN WRITE('THE SQUARE OF ',#Y,' IS ',#SQ) END;
 12 BEGIN  \  MAIN  \
 13   REPEAT BEGIN GETNO; SQUARE(X, Y ); PRINTNO; END
 14   UNTIL SQ = 0;
 15 END.   ( END MAIN )
```

SUBSCRIBER WRITES....

Subscriber Vance Pinter owns an IBM-XT he purchased to use in his daily office work and "hasn't regretted it for a minute." He included the color monitor to ease the transition from four years of CCII experience in his practice of law. The old machines are still used at home, for fun. Vance is especially interested in hardware articles and has added many options to his CCII, including lower case, the 4000H ram card by Tom Devlin, Frepost Computer's 16K RAM add-on, a bell, and second disk drive. He noticed an improvement from the disk drive modifications recommended by John Newman (Feb/Mar and Jun/Jul 1983).

"The Dynamic Ellipse Doodler by Tom Napier was fun. I keyed it in Sunday A.M. and it worked first try." Vance says he has thought of selling the old machines but just can't bring himself to do it. (I heard a whisper of regret from the Suits family not long ago that they no longer had the GOM on hand. ED)

CRT INTENSITY CONTROL

Instructions for mounting a front panel potentiometer to control CRT intensity have been prepared by subscriber Norman Johnson. A copy is available by writing to

The PROC (procedure) command is the single, most important concept in Pascal for creating a structured program. In a 'top down' structured programming approach, the functions of a program are separated into modules, each one programmed and tested separately, then joined together into one complete program. This removes complex debugging from program development, for if each parts works correctly, the whole will operate correctly.

To use a procedure, it must be defined before the program proper begins, and it may then be 'called' as many times as needed. Listing 6 demonstrates some sample procedures following the VAR and CONST commands. Here a procedure was written for each main part of the program. READNO reads in a number to be squared. SQUARE actually computes the square. PRINTNO prints the input number and its square to the screen. See how simple the main program becomes! The procedures are 'called' in the compound statement which follows the REPEAT command.

Note that the procedure SQUARE is the only one with a parameter list. In Pascal, this is a way of 'passing' a value or a number of values, back and forth between the main program and its procedures. The CALL to the procedure SQUARE, in the main program, also has a parameter list. The parameter list in the PROC and FUNC statements are optional, but if used, it must appear in both places. The names in the parameter list need not be the same, but their position in the list is important. This kind of parameter listing makes processes in the extended Pascal language 'portable' from one program to another, without renaming the variables they contain. Inserting variables in the right order is critical, however.

While full implementations of Pascal accommodate such a parameter list, in Tiny-Pascal, there is no provision for returning them. This is probably an oversight by its designer, Zimmer (or perhaps a deliberate part of the design!) So the result, SQ, is 'stuck' in the process. This really makes the

ANOTHER RETURN TO FCS

Chris Zerr submits his favorite way of returning to FCS or any other ESC [vector] from an assembly program. It is only applicable to v6.78 computers, however. If you first fill the A register with the vector code, such as [D] for FCS, [E] for Basic, [P] for 4000H, etc. If programs are ORiGined at 8200H they will be damaged by this method. The routine that completes the exit is located at 053AH in v6.78. A similar function has not been identified for v8.79 as yet. Here is a sample code:

```
START:  CALL   KEYIN  ;Get Keyboard input
        CPI    'D'    ;Do some
        JZ     EXIT   ; error
        CPI    'E'    ; checking...
        JZ     EXIT
        ;
        ; [Error routines here]
        ;
EXIT:   PUSH   PSW    ;Save registers
        MVI    A,12   ;Clean up the screen
        CALL   LO
        POP    PSW    ;Restore registers
        JMP    053AH  ;Exit program
```

Colorcue. We advise that while the project is simple, it should not be undertaken by inexperienced hands. Damage to the CCII and/or injury through shock to the installer is a potential danger.

```
SCR # 21
 0 ( *LISTING 7: FUNCTION EXAMPLE 1 ) DECIMAL : CLS 12 EMIT ;
 1  0 VARIABLE FUNCTIONVALUE : VALTOSTACK FUNCTIONVALUE @ ; PASCAL
 2 PROGRAM FUNCTIONEXAMPLE; VAR X,A,B,C,Z:INTEGER;
 3 FUNC QUADRATIC( XVAL,AVAL,BVAL,CVAL:INTEGER);
 4   BEGIN
 5    FUNCTIONVALUE := AVAL*XVAL*XVAL+BVAL*XVAL+CVAL; VALTOSTACK;
 6   END;
 7 BEGIN  CLS; \ MAIN PROGRAM \
 8  REPEAT
 9   BEGIN NEWLINE; NEWLINE; NEWLINE;
10    WRITE('ENTER X, A, B, & C TO EVALUATE (AX↑2+BX+C)*2+9 ');
11    READ(#X,#A,#B,#C); NEWLINE; NEWLINE;
12    Z := QUADRATIC(X,A,B,C) * 2 + 9;
13    WRITE('THE Z VALUE IS ',#Z)
14   END; UNTIL (X=0)
15 END. ( MAIN )  DECIMAL  #S
```

```
SCR # 22
 0 ( *LISTING 8: FUNCTION EXAMPLE 2 ) DECIMAL : CLS 12 EMIT ;
 1  0 VARIABLE FUNCTIONVALUE : VALTOSTACK FUNCTIONVALUE @ ; PASCAL
 2 PROGRAM FUNCTIONEXAMPLE; VAR X,A,B,C:INTEGER;
 3 FUNC QUADRATIC( XVAL,AVAL,BVAL,CVAL:INTEGER);
 4   BEGIN
 5    FUNCTIONVALUE := AVAL*XVAL*XVAL+BVAL*XVAL+CVAL; VALTOSTACK;
 6   END;
 7 BEGIN  CLS;   \ MAIN PROGRAM \
 8  REPEAT
 9   BEGIN  NEWLINE;  NEWLINE;
10    NEWLINE; WRITE('ENTER X, A, B, & C TO EVALUATE AX↑2+BX+C ');
11    READ(#X,#A,#B,#C); NEWLINE; NEWLINE;
12    WRITE('THE QUADRATIC VALUE IS ',#QUADRATIC(X,A,B,C))
13   END;
14  UNTIL (X=0)
15 END. ( MAIN )  DECIMAL  #S
```

```
SCR # 23
 0 ( *LISTING 9: MEMC ] EXAMPLE ) : CLS 12 EMIT ;
 1 : HOME 8 EMIT 11 EMIT ; PASCAL DECIMAL
 2 PROGRAM POINTPLOT;
 3 CONST SCREENSTART = 28672;
 4 VAR X, Y, Z : INTEGER;
 5 FUNC PLOT(XVAL,YVAL:INTEGER);
 6   BEGIN
 7     IF ((XVAL>-1) AND (XVAL<64) AND (YVAL>-1) AND (YVAL<31))
 8     THEN MEMC[SCREENSTART+X*2+Y*128] := 42
 9     ELSE BEGIN WRITE('RANGE ERROR '); Z := 1 END
10   END;
11 BEGIN  Z := 0; CLS;
12   REPEAT BEGIN HOME; WRITE('ENTER X,Y ');
13     READ(#X,#Y); PLOT(X,Y) END;
14   UNTIL Z = 1
15 END. #S
```

parameter list for a PROC in Tiny-Pascal superfluous, and we suggest that you not try use one. It really isn't needed since the variables defined in the VAR statement, in the declaration part of the program, are 'global' type variables (that is; accessible to all parts of the program, as opposed to just a particular process.) So in Tiny-Pascal, the portability concept is lost, an unfortunate, but not serious, circumstance.

A subset of the PROC command, in Tiny-Pascal, is FUNC, the function command. For this command, we have devised a way to circumvent the direct inability to pass values from a procedure back to the main program. The 'function' is defined in Pascal to compute a single value, whereas a PROC may compute any number of values. The procedure SQUARE may be written as a function instead. Alas, in Tiny-Pascal, procedures and functions are defined in the same way, and we are stopped again from returning parameters. FORTH comes to the rescue here by allowing us to define a FORTH variable in which to store the value of a function, and then define a means of placing that value on the FORTH stack so that it may be used in an equation. The FORTH code to accomplish this appears in the second line of Listing 7.

To use a FUNC in this way (for otherwise it is the same as a PROC), simply code the second line of your screen in a fashion similar to Listing 7. (Those interested in understanding the commands may refer to the FORTH definitions in a suitable reference.) The function, like the procedure, is defined in the first lines of the program (line #1 of Listing 7.) Its purpose is to clear the CRT whenever it is called. Note that the program in Listing 7 requires four input values. Separate the entry of each value with a carriage return. (Commas must not be used to punctuate inputs to any of the programs in this article.)

In general, FORTH may be used to extend the power and versatility of Tiny-Pascal by using FORTH to define com-

```
SCR # 24
  0 ( *LISTING 10: INC ] EXAMPLE )
  1 PASCAL DECIMAL
  2 PROGRAM INTEST;
  3         VAR A3:INTEGER;
  4 BEGIN
  5         REPEAT READ(A3)
  6         UNTIL A3 INC 30+2,'Y' ];
  7         REPEAT READ(A3)
  8         UNTIL A3 INC 30+2,'E' ];
  9         REPEAT READ(A3)
 10         UNTIL A3 INC 30+2,'S' ]
 11 END.  ;S
 12
 13
 14
 15
```

```
SCR # 25
  0 ( *LISTING 11: ARRAY EXAMPLE ) 0 VARIABLE N ; COLOR N @ EMIT ;
  1 : BON 31 EMIT ; : BOFF 15 EMIT ; : CLS 12 EMIT ;
  2 PASCAL DECIMAL
  3 PROGRAM USEARRAY;
  4 VAR IA,IB:ARRAY[8] OF INTEGER;
  5     C,I,Z:INTEGER;
  6 BEGIN
  7   IAC 1 ]:=16; IAC 2 ]:=17; IAC 3 ]:=18; IAC 4 ]:=19;
  8   IAC 5 ]:=20; IAC 6 ]:=21; IAC 7 ]:=22; IAC 8 ]:=23;
  9   IBC 1 ]:='B';IBC 2 ]:=82; IBC 3 ]:=71; IBC 4 ]:= 'Y';
 10   IBC 5 ]:=68; IBC 6 ]:=80; IBC 7 ]:=67; IBC 8 ]:=87;
 11   CLS;
 12 WHILE C<>'F' DO
 13    BEGIN
 14     NEWLINE; NEWLINE; Z:=0; I:=1;
 15
```

mands which then become part of the FORTH/Tiny-Pascal dictionary. In addition, the core directory of FORTH, once mastered, may be referred to in creating special effects not obtainable otherwise through its own basic commands. This kind of language is said to be "extensible."

Listing 8 is a modification of Listing 7 showing an alternate way of referring to a FUNC so its output may be 'passed' back to the program.

Our next two commands are not present in extended Pascal. MEM is used in Tiny-Pascal to place a value in an absolute memory location. The program of Listing 9 uses MEM to 'plot' (as in Compucolor: PLOT 3) using screen memory. MEM uses the [ ] style of delimiter to contain the memory address. We set MEM[addr] to a legal value (using :=) between 0 and 255.

[Note: The MEM command did not function until screen #39 of the Tiny-Pascal compiler was altered. The '!' word, which follows the word COMPILE, must be changed to 'C!'. Note that CLS is also defined in FORTH, ahead of the Pascal program, to use later to clear the screen. If you haven't surmised already, 12 EMIT is the same as PLOT 12, in Basic. Note on the following line another FORTH word, 'HOME', is defined by using '8 EMIT 11 EMIT' (PLOT 8 PLOT 11), to return the cursor to the upper left corner and erase the line.]

The IN command allows one to check a single keyboard character for a match with a list of possibilities within [ ] delimiters. Only ASCII numbers or their single character string equivalents, enclosed in single-quotes, are permitted in the brackets. Listing 10 presents an example of this command in use. When the program is 'run', it screens the keyboard for the single characters 'Y', 'E', and 'S', in that order. Any other input is ignored.

Our last program example, the use of the integer array, is demonstrated in Listings 11 and 12. This is our only example employing two screens working together. This program makes liberal use of FORTH-defined commands, to execute the equivalent of BASIC's PLOT command. In the program, two parallel arrays are defined by assignment. One array contains single character strings or ASCII values, and the other contains the equivalent of BASIC's PLOT numbers. The user is asked to choose a color by selecting a single letter key. With a simple table 'look-up', controlled by a WHILE..DO sequence, the ASCII value of the letter is sought. When it is found, the value at the same index in the parallel array is used to set the color, prior to screen printout. Screen 25 has no final ';S' so the program continues on to the next screen.

Tiny-Pascal has provision for most commonly-used operators. These include MOD, NOT, OR, SHL, SHR (shift

CHIP REPAIR NETWORK FOR THE CCII

A network of private service facilities is being formed with the help of Intecolor Corporation, who have offered to assist us by supplying a parts inventory. So far, three locations have been designated. I have confirmation on only the first two listed here. You are invited to contact these service personnel regarding CCII repairs. All three come highly recommended, and all three have had extensive experience.

Steve Wooten: 155 Barington Street, Rochester, NY 14607. Telephone: (716) 442-4914. Call evenings.

Steve charges $20.00 per hour if he fixes your computer. There is no charge if he doesn't. Customer pays for parts and shipping both ways.

Gary Sipple: 27750 Golfview Street, Southfield, MI 48034. Write to Gary for details.

Bill Freiberger: Box 207, Mountain Lakes, NJ 07049. Telephone: (201) 263-2859. Write to Bill for details.

When having a CCII repaired the biggest hurdle is in the shipping. Severe damage has occurred in the past to computers improperly packaged. It is your responsibility to prepare the computer properly for shipment. Make certain that you consult with the repair agent you select before making shipment. The cathode ray tube, #13VAXP22, used in the CCII was proprietary and is no longer available!

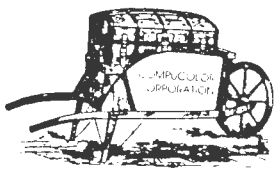
```
SCR # 26
  0  \ #LISTING 12: ARRAY EXAMPLE CONTINUED \
  1    WRITE('ENTER COLOR TYPE (B,R,G,Y,D,P,C,W)? F-END ');READ(C);
  2    WHILE Z<>1 DO
  3     BEGIN
  4      IF C=IBC[I] THEN
  5       BEGIN N:=IAC I ]; Z:=1
  6       END;
  7      I:=I+1; IF I>9 THEN Z:=1
  8     END;
  9     NEWLINE; NEWLINE; COLOR;
 10     IF I<10 THEN WRITE('THIS REPRESENTS THE COLOR CHOSEN ')
 11             ELSE
 12              BEGIN BON; WRITE('ENTRY ERROR '); BOFF
 13              END;
 14   END; C:='B'
 15 END. DECIMAL #S
```

bits left or right), 'less than' (and 'or equal to'), 'greater than' (and 'or equal to'), '=', '+', '-', '*', DIV, TRUE, and FALSE.

There are several commands in Tiny-Pascal we have not covered, such as TYPE and CALL. CALL is supposed to be a FORTH routine to 'call' non-alphabetic routines in Tiny-Pascal. Perhaps it will be the subject of a future article in DATA CHIP. The TYPE command does not function in the normal Pascal sense, so no immediate explanation of this command is available.

There is greater depth to Tiny-Pascal commands than has been conveyed in these articles. A more thorough explanation may be found in Jim Minor's Tiny-Pascal documentation in the PASCAL Syntax Diagrams. You may have copies of this material from the CHIP library.

This ends our Tiny-Pascal series in Colorcue. I apologize to those of you who have been 'stumped' by some features of Tiny-Pascal because of insufficient coverage here. Feel free to telephone me, evenings, at (716) 889-4994, if you would like to explore any particular feature in greater depth. My very best wishes to my friend, Joe Norris, and my heartfelt thanks to him for his contributions to the Compucolor Community by piloting Colorcue during its last year [].



BACK ISSUES

Vol VI, numbers 1 and 2 are out of print. They will be available in Xerox form at $4.00 each if requested.

CHIP Library Reviews

We are interested in having your review of *CHIP* library holdings which you have found useful. Describe what the program does and how you use it. Include games, utilities, languages, etc.

COLORCUE CONTEST

Not one entry! The prize money will be returned to Colorcue's publication account. Shame! Shame!

SOFTWARE STILL AVAILABLE

You may still purchase CCII software from Intelligent Computer Systems in Huntsville, AL. The Muellers are permanently in the United States and are pursuing a diversity of activities involving computers and alternative energy sources. It is very considerate of them to maintain the CCII software library for us even though it must be more a nuisance than a business at this point. They are a good source for diskettes at a very low price.

Glen Gallaway 'found!'

Intecolor 8000 users will be happy to know that Glen Gallaway, the 8000 coordinator, has been 'found' again, following some confusing address changes. His new address is 1637 Forestdale Avenue, Beavercreek, OH 45432. Glenn is relocated now and is looking forward to some permanence.

Work continues to compile a useful set of memory addresses for conversion of CCII programs to the 8000 series computers. All 8000 users are invited to participate. Write to Glen at the above address if you are ready to help.

Tom Teaser!

Tom Napier wants to know if any readers can, without using the MVIinstruction, or making any preconditions, load the A-register with '06' using only 2 bytes!

TECH TIP, *DISK DRIVES*

One user had a problem with his disk drive that turned out to result from a drive belt near the breaking point. Doug Van Putte has found a source of drive belts for the CCII. Try Floppy Disk Services, 741 Alexander Road, Princeton, NJ 08540. Their phone number is (609) 799-4440. Ask for a standard 5-1/4" Siemans drive belt.

CONNECTING EXTERNAL DRIVE

Gary Dinsmore has submitted plans for connecting an old internal drive as a second CCII drive. If you have such a drive available, Colorcue will be happy to send you a copy of his procedure for its installation.

NEXT ISSUE: CCII publications index; reviews of software by Bill Stanton and Wallace Rust; a captivating program adaptation by Tom Napier (a magnum opus 'Cuties'); an article on Recursion by Doug Van Putte; W. S. Whilly's final installment on debugging, a biographical sketch of Peter Hiner, and more. Get your contributions in promptly for our last issue!

CURRENT COLORCUE SUSCRIBERS

ACT DISTRIBUTOR
8522 CRANSTON DRIVE
WESTLAND, MI 48185

GEORGE R. ADAMS
4452 HIGHLAND CIRCLE
MARIETTA, GA 30066

JAY C. ALBRECHT
4686 FREEMAN ROAD
MIDDLEPORT, NY 14105

ED ALLEN
6040 SMITH LANE
REDDING, CA 96002

CHARLES L. ANDERSON
2728 HASTE STREET
BERKELEY, CA 94704

HAROLD L. ANDERSON
1106 WILSON STREET
RICHLAND, WA 99352

TOM ANDRIES
1625 N. IOWAS STREET
SOUTH BEND, IN 46628

B. J. ARBUNIC
5210 MARIT DRIVE
SANTA ROSA, CA 95405

GENE BAILEY
28 DOGWOOD GLEN
ROCHESTER, NY 14625

BEN BARLOW
161 BROOKSIDE DRIVE
ROCHESTER, NY 14618

MICHAEL P. BARRICK, VAL FRG HS
9999 INDEPENDENCE BOULEVARD
PARMA HEIGHTS, OH 44130

JOHN E. BEAM
4807 VALERIE
BELLAIRE, TX 77401

VINN BEIGH
1221 WEST GLENLAKE AVENUE
CHICAGO, ILL 60660

CREIGHTON BELL
3332 WAYSIDE
EL PASO, TX 79936

JOHN BELL
8300 4TH AVENUE
NORTH BERGEN, NJ 07047

WILLIAM R. BOCK
8317 LOUIS DRIVE
HUNTSVILLE, AL 35802

JACK H. BOGHOSIAN
1843 N. THORNE AVENUE
FRESNO, CA 93704

DAVID B. BOUCHER
33 ST JOSEPH AVENUE
FIRCHBURG, MA 01420

FATHER GEORGE BRUNISH
MATH DEPT/WESTMINSTER COLLEGE
NEW WILMINGTON, PA 16142

STEVE BRUNO
2626 CORONADO AVENUE
SAN DIEGO, CA 92154

R. S. BUCY
420 SOUTH JUANITA
REDONDO BEACH, CA 90277

MICHAEL R. BURCHAM
1707 GLEASON
IOWA CITY, IA 52240

ROBERT L. BURTON
114 AMANDA PLACE
OAK RIDGE, TN 37830

ARTURO L. CAZARES
2366 EAST FREMONT STREET
STOCKTON, CA 95205

JOSEPH J. CHARLES
130 SHERWOOD DRIVE
HILTON, NY 14468

DR. ORPHIA C. CHELLAND
501 NORTH PROVIDENCE ROAD
MEDIA, PA 19063

GENE COLLINS
ROUTE 7 BOX 354
CONWAY, SC 29526

PAT COLLEY
52 QUEENSWAY, CAVERSHAM PARK
READING, ENGLAND RG4 05J

COMMACK HIGH SCHOOL, MR WETJEN
VANDERBILT PARKWAY
COMMACK, NY 11725

VINCENT CORDOVA/NATIONAL MEDIC
PO BOX 433-A
WILLOW GROVE, PA 19090

BRIAN CRUSE
8 ULVA STREET, BALD HILLS
QUEENSLAND 4036, AUSTRALIA

MICHAEL DALEY
4643 CLOVER LANE
TOLEDO, OH 43623

WILLIAM L. DARKE
3310 SOUTH DEXTER STREET
DENVER, CO 80222

HUGH DARRAGH
32 McCAUL STREET/TARINGA
EAST BISBANE 4068 AUSTRALIA

S. DE SANTIS
700 AMSTERDAM ROAD
MT LAUREL, NJ 08057

MICHAEL DEVITO
2729 WIEDRICK ROAD
WALWORTH, NY 14568

JANE AND TOM DEVLIN
3809 AIRPORT ROAD
WATERFORD, MI 48095

THEODOR DIDRIKSSON
5916 AMAPOLA DRIVE
SAN JOSE, CA 95129

GARY A. DINSMORE
ROUTE 3 BOX 3216
WARREN, OR 97053

P. ALLEN DOW
260 NORTH MATHILDA AVENUE/A5
SUNNYVALE, CA 94086

PAUL DUDLEY
140 RAILROAD MILLS ROAD
PITTSFORD, NY 14534

RANDALL DUNSMOOR
615 SOUTH JACKSON STREET
GREEN BAY, WI 54301

RONALD EISENSMITH
22 KINGSWOOD
ORCHARD PARK, NY 14127

WILLIAM A. EMOND
1450 OAKLAND RD. SPACE #8
SAN JOSE, CA 95112

DUAINE ESBENSHADE
RIVER ROUTE, BOX 875
SILETZ, CA 97380

JOHN EWEL
DEPT OF BOTANY/UNIV OF FLORIDA
GAINESVILLE, FL 32611

MARK D. FAIRBROTHER
HC78 BOX 442 PENNVIEW APTS
BINGHAMTON, NY 13901

DR. MARJORIE FIRRING
8385 CHIANTI COURT
SAN JOSE, CA 95135

HOWARD FLANK/FLANK ASSOCIATES
2809 ATLANTA DRIVE
WHEATON, MD 20906

HENRY G. FLUCK
304 RANDLE COURT
CHERRY HILL, NJ 08034

J. FORD
PO BOX F
KIMMSWICK, MO 63053-0010

M. B. FRASER
1 LILY STREET/NORTH RYDE 2113
NEW SOUTH WALES, AUSTRALIA

BILL FREIBERGER
BOX 207
MOUNTAIN LAKES, NJ 07049

BRUCE A. GEIL/ G&B AUTO
7017 51ST AVENUE SOUTH
TAMPA, FL 33619

JAMES GICZKOWSKI
THE WURLITZER COMPANY BOX 591
CORINTH, MS 38834

IRVING GILES
172 WALNUT CREEK LANE
TOMS RIVER, NJ 08753

CHARLES H. GOULD
317 COCOA AVENUE
INDIALANTIC, FL 32903

D. B. GRANT
2 BROOKSIDE AVENUE/SOUTH PERTH
WEST AUSTRALIA 6151

EDWARD GREANEY
BOX 421 CHESTER AVENUE
NESHANIC, NJ 08853

DANIEL P. GREEN
905 BEECH
DINCAN, OK 73533

EVAN GREEN
11520 38TH NE
SEATTLE, WA 98125

WILLIAM L. GREENE
3601 NOBLE CREEK DRIVE, NW
ATLANTA, GA 30327

ART GRUSENDORF
BOX 605, MAGRATH, ALBERTA
CANADA TOK 1J0

FREDRIC HAERICH
1020 BROADWAY STREET
ALTAMONTE SPRINGS, FL 32714

JIM HALDEMAN
353 S WILLIE
WHEELING, IL 60090

ROBERT HALLEY
1714 MALDEN STREET
SAN DIEGO, CA 92109

CHARLES E. HAMMETT JR
611 BARKFIELD STREET
BRANDON, FL 33511

HAROLD R. HAMM
11739 LORETTO WOODS COURT
JACKSONVILLE, FL 32223

ROBERT G. HARDIN
1424 CHARLES AVENUE
KALAMAZOO, MI 49001

A. P. HARGREAVES
BRIDGE STREET/ ELTHAM
TARANAKI, NEW ZEALAND

BOB HARRIS
2954 SUNWOOD DRIVE
SAN JOSE, CA 95111

GLENN HAYHURST
9595 PECOS #14
DENVER, CO 80221

PETER HINER
11 PENNY CROFT/HARPENDEN
HERTS/ ENGLAND AL5 2PD

HARLEN HOWARD
832 SAN RAFAEL STREET
SUNNYVALE, CA 94086

FRED HUDSON
643 BROOKS ROAD
W. HENRIETTA, NY 14586

GRAHAM HUNT
17 BAROSSA CLOSE/ST CLAIR 2759
NEW SOUTH WALES, AUSTRALIA

INTEGRATED LOGISTICS SYSTEMS
PO BOX 518
ALTADENA, CA 91001

CHARLES IVY
10926 SHADOW WOOD DRIVE
HOUSTON, TX 77043

RONAN JAMES
11770 TIMBERLINE LANE
HALES CORNERS, WI 53130

NORMAN JOHNSON
STONE MARINA
JOHNS ISLAND, SC 29455

R. JONES
33383 LYNN AVENUE/ ABBOTSFORD
BRITISH COLUMBIA/CANADA V2S1E2

JAMES R. KENNEY
257 BERRY ROAD
BEAUMONT, TX 77706

JOHN KER
11839 WEST TRAIL
KAGEL CANYON, CA 91342

HARRY J. KEROP
76 RATTLING VALLEY ROAD
DEEP RIVER, CT 06417

KEN KERRISON
5 BELTANA ROAD/PIALLAGO A.C.T
AUSTRALIA 2609

ALDOLPH KLUKOVICK/KAY ENTERPR
2325 KINGSBRIDGE LANE
OXNARD, CA 93030

WILLIAM G. KNAPP
1761 CARMEL DRIVE
IDAHO FALLS, ID 83402

PAUL D. KOERBER
17090 SAN BRUNO/APT G23
FOUNTAIN VALLEY, CA 92708

FREDRICK G. KOMMRUSCH
5670 N. PARADISE LANE
MILWAUKEE, WI 53209

DR C. W. KREITZBERG
2311 N. FEATHERING ROAD
MEDIA, PA 19063

DENNIS L. LEPARD
120 S. ELLINGTON
DEPEW, NY 14043

A. LEWIS
PO BOX 228/ PARABURDOO 6754
AUSTRALIA

GOTE LILJEGREN
MARGARETAV 12 G/ 42 TABY
SWEDEN

FRANK M. LOCKE
2213 SOLORWAY
LAS CRUCES, NM 88001

ROBERT C. LOVICK
88 HILLHURST LANE
ROCHESTER, NY 14617

RIC LOWE
80 CAWSTON ROAD/ ATTADALE 6156
WESTERN AUSTRALIA

TERRY LUND
45 EVERGREEN STREET
SPENCERPORT, NY 14559

RONALD MACKENZIE
1 BRIAN STREET
COMMACK, NY 11725

GLEN MANN
6002 - 152ND AVENUE NE
REDMOND, WA 98052

JAMES MANUELLE
138 SHALE DRIVE
ROCHESTER, NY 14615

JOHN H. MASCARENHAS
PO BOX 78
LEDYARD, CT 06339

ROBERT H. MASKREY
1041 MILL ROAD
EAST AURORA, NY 14052

ALAN MATZGER
960 GUERRERO
SAN FRANCISCO, CA 94110

ANDY MAU
5 ELDRIDGE STREET
NEW YORK, NY 10002

PAUL F. MCCARRON
PO BOX 108
BELGRADE LAKES, ME 04918

FRED W. MCILROY, III
24426 24TH S STREET
KENT, WA 98031

MED/ASSIST DISTRICT AUDITORS
6815 WEST CAPITOL DRIVE
MILWAUKEE, WI 53216

ALAN MEGHRIG
2491 BUCKSKIN DRIVE E 13
LAGUNA HILLS, CA 92653

CAROLYN MEITLER
6110 MANSFIELD DRIVE
GREENDALE, WI 53129

BOB MENDELSON
27 SOMERSET PLACE
MURRAY HILL, NJ 07974

H. G. METZLER
235 BELCODA DRIVE
ROCHESTER, NY 14617

ALBERT J. MILLER
179 WALTER HAYS DRIVE
PALO ALTO, CA 94303

JACK E. MILLER
BOX 200
CARSON CITY, NV 89702

JOHN J. MINERD
BOX 191
YORKSHIRE, NY 14173

DR. JAMES MINOR
22 BRYN MAWR ROAD
ROCHESTER, NY 14624

GEORGE C. MOENCH, MD
1952 49TH STREET SOUTH
ST PETERSBURGH, FL 33707

THOMAS W. MONTEMARANO
1321 SWAN DRIVE
ANNAPOLIS, MD 21401

EARL H. MOORE
2112 BANCROFT
LAKE CHARLES, LA 70605

BRUCE R. MOREHEAD
3408 W. KIRBY
TAMPA, FL 33614

EIKE MUELLER
12117 COMANCHE TRAIL
HUNTSVILLE, AL 35803

TOM NAPIER
12 BIRCH STREET
MONSEY, NY 10952

JOHN E. NEWBY
4532 - 167TH AVENUE SE
ISSAQUAH, WA 98027

JOHN NEWMAN
8 HILLCREST DRIVE/ DARLINGTON
WESTERN AUSTRALIA 6070

DAVID C. NORMAN
3883 SAN MARCOS COURT
NEWBURY PARK, CA 91320

JOSEPH H. NORRIS
19 WEST SECOND STREET
MOORESTOWN, NJ 08057

LT COL DAVID NOWLIN
1103 SOUTH GRANDVIEW
PAPILLION, NEBRASKA 68046

H. T. ODUM
2106 NW 9TH AVENUE
GAINESVILLE, FL 32603

BRIAN E. O'HEARN
78 COLUMBUS AVENUE
SOMERVILLE, MA 02143

WILLIAM PARKER
2812 BERKLEY
FLINT, MI 48504

ASHOK S. PATWARDHAN
4260 CLAYTON ROAD APT #22
CONCORD, CA 94521

A.PAREIGIS
4411 SOUTH CROSS STREET
DOWNERS GROVE, IL 60515

STEVE PERRIGO
16925 INGLEWOOD ROAD NE B-306
BOTHELL, WA 98011

DULIN B. PERRY
506 HILLHURST DRIVE
BAYTOWN, TX 77521

LARRY W. PETERSON
6325 ELEANOR AVENUE
OAKDALE, CA 95361

MICHAEL J. PETREYCIK
11 STAG LANE
TRUMBULL, CT 06611

MELVIN F. PEZOK
1381 IGNACIO BOULEVARD
NOVATO, CA 94947

ALEX V. PINTER, PC
PO BOX 230
COLUMBUS, GA 31902

RALPH J. PORTER
6157 SOUTH 700 W
MURRAY, UT 84107

WILLIAM J. POWER
8 PETER COOPER ROAD
NEW YORK, NY 10010

THOMAS PRICE, JR
129 HOMESTEAD AVENUE
DE BARY, FL 32713

J. RAMSEY
3115 S. ATLANTIC AVE #504
COCOA, FL 32931-2137

CARL E. REMLEY
149 BAYWOOD ROAD
BILOXI, MS 39532

HERBERT RICHARDSON
5805 ANTIONE ROAD
MOBILE, AL 36609

DAVID R. RICKETTS
108 BRYCE AVENUE
RED BANK, TN 37415

ALBERT RIDNER
CENTRO ATOMICO BARILOCHE
8400 BARILOCHE, ARGENTINA

JACK RIPPLE
65 APPLECROSS CIRCLE
CHALFONT, PA 18914

R. M. ROCKWELL
235 BELMONTE ROAD
WEST PALM BEACH, FL 33405

HERBERT ROSE/ WILCOMP OFF SVCS
465 WILSON AVENUE
DOWNSVIEW, ONTARIO M3H 1T9

HOWARD ROSEN
PO BOX 434
HUNTINGTON VALLEY, PA 19006

MICHAEL J. ROUSSE
15 SOUTH OWEN DRIVE
MADISON, WI 53705

WALLACE R. RUST
533 BRITTON ROAD
GREECE, NY 14616

BENJAMIN R. SAGE
16608 EAST STANFORD PLACE
AURORA, CO 80015

WILLIAM J. SEMBER
PHILIPS ECG, INC/JOHNSON ST
SENECA FALLS, NY 13148

ROY A. SHAFFER
530 SPRINGSIDE LANE
BUFFALO GROVE, IL 60090

WILLIAM SHANKS
1345 WEST ESCARPA STREET
MESA, AZ 85201

DONALD E. SHAUB
6294 HIGH STREET
EAST PETERSBURG, PA 17520

RONALD D. SHOOK
160 ANDOVER STREET
WILKES-BARRE, PA 18702

PHIL SIMON
6275 CARY AVENUE
CINCINNATI, OH 45224

GARY SIPPLE
27750 GOLFVIEW STREET
SOUTHFIELD, MI 48034

BOB V. SMITH
498 BROWN STREET
NAPA, CA 94558

BYRON E. SMITH
1189 LOCKE AVENUE
SIMI VALLEY, CA 93065

DAVID R. SMITH
5051 DIERKER ROAD / APT A4
COLUMBUS, OH 43220

MEL SMITH
NSSA/STDN
DANDAN, GU 96919

RALPH S. SMITH JR MD
5600 MAC CORKLE AVE SE SUITE 9
CHARLESTON, WV 25304

DR. JACK M. SPURLOCK
293 INDIAN HILLS TRAIL
MARIETTA, GA 30067

RICHARD F. SQUAILIA
818 MAIN AVENUE
SCHENECTADY, NY 12303

PETER STANDEN
12 KENDALL STREET/ CHARLESTOWN
AUSTRALIA 2290

BILL J. STANTON
8115 HELM COURT
COLORADO SPRINGS, CO 80918

DAN STIEFLER
S 3660 FULLER STREET
BLASDELL, NY 14219

JOSEPH STRATMAN
52029 US 33 NORTH
SOUTH BEND, IN 46637

DANA G/ STREBECK
8034 SHADY ARBOR LANE
HOUSTON, TX 77040

TED STUCKEY
BOX 420/ CAMBERWELL, 3124
VICTORIA, AUSTRALIA

DAVID SUITS
49 KARENLEE DRIVE
ROCHESTER, NY 14618

DICK SWARM
12709 GREENHALL DRIVE
WOODBRIDGE, VA 22192

JAMES P. SWEENEY
1566 WOMACK ROAD
DUNWOODY, GA 30338

ARTHUR TACK
1127 KAISER ROAD SW
OLYMPIA, WA 98502

BOB TALBOT
909 EXPLORER #2
RAPID CITY, SD 57701

T. R. A. P. S.
PO BOX 8297 STATION "F"
EDMONTON, ALBERTA T6H 4W6

ROBIN WERNICK
9516 CARROLL CANYON DRIVE
SAN DIEGO, CA 92126

STEVE WOOTTEN
155 BARINGTON STREET
ROCHESTER, NY 14607

RICK TAUBOLD
197 HOLLYBROOK ROAD
ROCHESTER, NY 14623

MARKE UNDERWOOD
1750 DYSON DRIVE NE
ATLANTA, GA 30307

MAYNARD WILCOX
107 EAST AVENUE
FRANKFORT, NY 13340

DAVID R. WRIGHT
1305 N EIRE / APT 23
LEXINGTON, NE 68850

JOHN B. THIRTLE
105 COIFER LANE
ROCHESTER, NY 14622

DOUG VAN PUTTE
18 CROSS BOW DRIVE
ROCHESTER, NY 14624

MARC A. WILLIAMS
3 AMES STREET
CAMBRIDGE, MA 02139

WILLIAM B. WRIGLEY
4931 REBEL TRAIL, NW
ATLANTA, GA 30327

MR. D. G. THOMAS
52 GROVE WAY/ WEMBLEY
MIDDLESEX, ENGLAND HA9 6JT

CHRIS VERBEEK
14721 35TH AVENUE SE
BOTHELL, WA 98012

WAYNE C. WILLIAMS, DIR
SCH OF MED/EAST CAROLINA U.
GREENVILLE, NC 27834

DAVID ZAWISLAK
5729 N CALIFORNIA AVENUE
CHICAGO, IL 60659

TOMMY THYSTRUP
VIRKELYST 20/ NR. SUNDBY 9400
DENMARK

RICKI ANDREW VICK
702 WEST HOLLY AVENUE
STERLING, VA 22170

WIS SURVEY RESEARCH LAB
610 LANGDON STREET/ 109 LOWELL
MADISON, WI 53703

CHRIS ZERR
10932 - 156TH COURT NE
REDMOND, WA 98052

J. P. TOOHEY
24 HOSKEN STREET/ NORTH BALWYN
MELBOURNE 3104/ AUSTRALIA

JAMES D. WARNER
11647 YUBA RIDGE DRIVE
NEVADA CITY, CA 95959

THOMAS R. WOOLF
80 BOWEN ROAD
CHURCHVILLE, NY 14428

ANTHONY ZUVLIS
1117 SEQUOIA
FORT COLLINS, CO 80525

STEFFAN TOTH
PO BOX 1779, STN A
KELOWNA, BC V1Y 8P2

ROY WEISENBARGER
1201 CHESHIRE ROAD
MAITLAND, FL 32751

W. BRYANT WOOSLEY , JR
PO BOX 728
SHELBYVILLE, TN 37160

BASIC PRECISION

Subscriber Wallace Rust, of the Rochester User Group points out that Intecolor computers, including the CCII, can store numbers up to 16777216. While BASIC truncates and rounds them to six digits for display, it still stores them internally with at least 7-digit accuracy, and with 8-digit accuracy to the number stated above. You have nothing to lose, therefore, by entering constants with 8-digit accuracy. This will guarantee highest accuracy in computations before the rounding, prior to display, takes place. You can prove this rather easily with some simple experiments, perhaps using PI. [See some exploratory examples below. ED]

```
10  REM  Sample calculations to test numerical precision
20
30  X = 1.2345
40  PRINT 12 * X                          RUN
50
60  Y = 1.2345678                          14.814
70  PRINT 12 * Y
80                                         14.3148
90  PRINT 1/X
100                                        00.810045
110 PRINT 1/Y
120                                        00.81
130 PI(1) = 3.1416
140 PRINT PI(1) * 12^2                    452.39
150
160 PI(2) = 3.141592654                   452.389
170 PRINT PI(2) * 12^2
```

A 'no—echo' patch without assembly language!!

Here's one from Tom Devlin—a new version of the no-echo patch that uses no machine language! And it works!

Most users are familiar with the ESC USER jump location (if not, see Rick Taubold's detailed explanation in Colorcue). There is also a user-defined input flag vector which may be used to produce a deviation from the 'normal' flow of computer processes. Normally, the CCII performs its I/O with BASIC, FCS, the serial port, etc., according to default settings of the various flags. These settings are initialized by FCS at turn on time to prepare the computer for 'normal' operation. The programmer holds the power to alter this normal state of affairs. If the input flag at 33221-33223 (81C5H-81C7H), for example, is set to any of a group of special values, input is directed to a special jump vector (user defined), just like ESC USER. This jump vector can transfer operations to an unusual place, determined by the programmer.

As an example, the keyboard flag is at location 33247 (81DFH). This flag value is normally '0' and input keyboard characters are sent to the CRT. When it is '13' they go to FCS, which might send them back to the CRT or to a disk drive. When the flag is '12', the keyboard input is ignored. But if the flag holds any of the values 10, 11, 15-17, 19-22, 24, 26, or 28-31, then a jump to the memory location stored at address 33221—33223 occurs. If these addresses hold 8355H, for instance, then program execution continues at that location.

The routine below proceeds in just this way. The value '30' is placed in the jump vector address, 33221—33223. What we place in this address is the address of the routine that gets and stores the next keyboard press, but rather than allowing it to print, we trap these values for our own use.

Program lines 250-290 access the keyboard character, now in location 33278. From here on, Basic may do as it likes, we we have controlled the timing of this event to suit our needs. The following listing will demonstrate this technique for you. Type it in, and play with the variations that are possible. □

```
00010 REM   FIRST WE HAVE TO LOAD THE PROPER ADDRESS
00020 REM   INTO JUMP VECTOR "INPCRT" .THIS MUST BE
00030 REM   DONE BEFORE 'NO ECHO' IS REQUIRED
00040
00050 POKE 33221,195:REM POKE 'JMP' INTO FIRST BYTE
00060
00070 REM      FOR V6.78 THE ADDRESS IS 3FA0H
00080 IF PEEK (1)=108 THEN POKE 33222,163:POKE 33223,63
00090
00100 REM      FOR V8.79 OR V9.80 IT'S  0A01H
00110 IF PEEK (1)=186 THEN POKE 33222,1:POKE 33223,10
00120
00130 REM   FOR NO ECHO POKE 30 INTO "INPFLG" (33247),
00140 REM   TO RETURN TO ECHO POKE 12. (AN INPUT STATEMENT
00150 REM   OR THE END OF THE PROGRAM WILL ALSO DO IT)
00160
00170
00180 REM       THE FOLLOWING PROGRAM IS FOR DEMO ONLY
00190 REM       IT ECHOS THE ASCII EQUIVALENT OF THE
00200 REM       KEY YOU PRESS. PRESS "HOME" TO END
00210
00220 POKE 33247,30:REM TURN OFF ECHO
00230
00240
00250 POKE 33278,0
00260 K=PEEK (33278):IF K=0 THEN 260
00270 PRINT K;
00280 IF K=8 THEN END
00290 GOTO 250
```

CCII Color Adjustment

Wallace Rust

Compucolor users often have trouble distinguishing cyan from white, and yellow from green. The culprit is the green CRT phosphor, which emits not only green light, but also unwanted energy in the yellow, orange, and red regions of the spectrum. Thus, cyan images contain some unwanted red which makes them look white, and green images contain some unwanted red which makes them look yellow.

The trick to getting better colors is to adjust the green channel brightness so that its unwanted emissions are significantly weaker than the red emission of the red channel.

Page 6.06 of the Compucolor Maintenance Manual (ISC 216 978 999208) explains how to adjust the three screen grid controls R1 (red), R2 (green), and R3 (blue) located at the top of the circuit card at the base of the CRT. A similar procedure is outlined in the 3651 Manual, described here.

CAUTION: HIGH VOLTAGES ARE PRESENT IN THIS AREA OF THE COMPUTER. DO NOT ATTEMPT THESE ADJUSTMENTS UNLESS YOU KNOW EXACTLY WHAT YOU ARE DOING. NEVER USE BOTH HANDS TO MAKE ADJUSTMENTS. STAND ON AN ELECTRICALLY ISOLATED SURFACE WHEN WORKING ON THE COMPUTER. IF IN DOUBT, OBTAIN THE ASSISTANCE OF A QUALIFIED SERVICE PERSON.

(A convergence and color purity adjustment should be made first. Write to Colorcue for details of these adjustments.)

1. Erase the screen with a background color of white.

2. Turn potentiometers R1, R2, and R3 fully to their 'off' position. Turn the brightness control, mounted on the back of the unit, on the 50 pin bus side, below the FOCUS control, to maximum brightness.

3. Turn the red control, R1, until the red retrace area is just visable. Repeat the procedure for the green and blue controls (R2 and R3.)

4. Adjust the brightness control until there is no visable retrace line, and until brightness is at a comfortable level with a minimum of saturation.

5. Vary the adjustment of these controls to obtain the best white display at a comfortable level of brightness.

After following that procedure, do the following:

1. Put some test samples of each color on the screen at the same time.

2. Confirm that the red/blue ratio is such that you get a good magenta, which is neither too blue nor too red. Make minor pot adjustments to meet this criterium.

3. Turn R2 (green) to reduce the brightness of the green channel, until yellow becomes slightly orange (like KODAK yellow), and white becomes pink, relative to daylight white.

That's all there is to it!! □

UNCLASSIFIED ADS

FOR SALE: CCII, v6.78, lower case, 2 drives, full keyboard. Computer is in excellant condition, and comes with most issues of *COLORCUE* and *FORUM*, and many program disks. $1000.00. Call or write Bill Stanton, 8115 Helm Court, Colorado Springs, CO 80918; 303 599-4089.

FOR SALE: CCII, v6.78,32K, 1 drive. Maintenance Manual and following software: Personal Data Base, Basic Language 1-10, Formatter, Basic Editor, Screen Editor, Assembler, Fortran, miscellaneous games and disk utilities. All back issues of *COLORCUE*. Computer in good working order, $700.00, money order or certified check, prepaid. Bill Anthony, 655 Wells Way, Camano Island, WA 98292. 206 387-1576.

FOR SALE: CCII, v6.78, 32K, full keyboard, switchable lower case, handshake option, printer cable, sound, dust covers. Over 70 disks including Helm's Data Base, Screen Editor, Print; Com-Tronics TERMII; Income Tax Program, and many more. Manuals and periodicals included. $1500.00, prepaid. Phil Simon, 6275 Cary Avenue, Cincinnati, OH 45224. 513 681-8370.

BASIC VARIABLES IN FILE STATEMENTS

FILE "N" and FILE "R" statements in Basic use a combination of string and numeric parameters to specify the file name and attributes. It is not obvious that variables can be used to specify these quantities in the course of a Basic program. The examples below show the extremes to which variables may be used. Suppose you have a RND file of 128 bytes containing some data for each week in the year and want to read the file for a specific week. The following coding identifies the week number as a 'decimal' version number [1-52]:

```
420 INPUT "ENTER WEEK NUMBER ) ";W$
430 FILE "R",2,"WEEK.RND;"+W$,1;1,128,1
```

String concatenation may be used to construct a proper file name as illustrated above. Note that the file name must be in string form. We could alternately call the RND files 'WK1.RND', 'WK23.RND', etc:

```
420 INPUT "ENTER WEEK NUMBER ) ";W$
430 FILE "R",4,"WK"+W$+".RND",1;1,128,1
```

As long as we assign string variables where strings are required, and numeric variables where numbers are required, we can successfully operate on files with variables. Consider this example:

```
120 A$="WK" : C$=".RND" : F=1 : H=128
170 INPUT "ENTER WEEK NUMBER ) ";W$
200 FILE "R",1,A$+W$+C$,1;F,H,F
```

A final example is taken from a working business program of mine (which has a well-documented file table in the manual!):

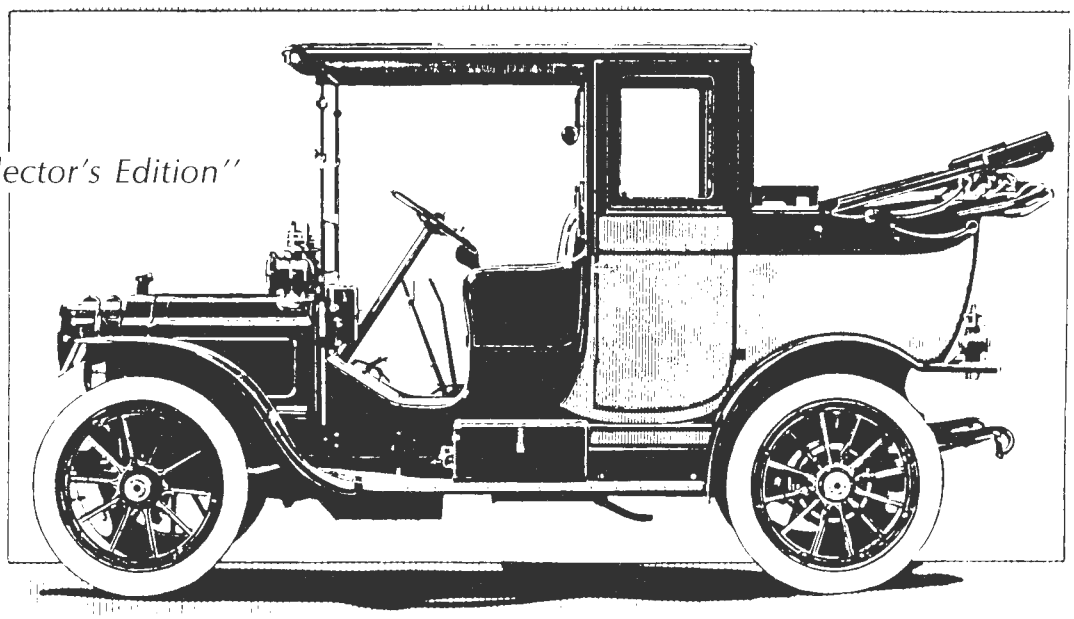
```
950 FILE "R",N,"CD"+D$+":"+Z$(2)+
    MID$(P$(2*CK),GH-5,T),F;4*P,KL-6,C
```

As always, a few 'controlled' experiments will help give you a confident grasp of the rules for using variables in this way. [JHN]

COLORCUE

A BI-MONTHLY PUBLICATION BY AND FOR INTECOLOR AND COMPUCOLOR USERS

VOLUME VI
NUMBER 6



"Collector's Edition"

from The Intecolor Corporation

The COMPUCOLOR II.

with tributes by

John Newby, *geologist*
Doug Van Putte, *miracle worker*
W. S. Whilly, *nuisance*
Wallace Rust, *astronomer*
Chris Zerr, *experimenter*
Bob Mendelson, *8000 specialist*
Tom Napier, *groom.*
Thomas Wulff, *programmer*

Colorcue

VOLUME VI, NUMBER 6 NOVEMBER/DECEMBER 1984

CONTENTS

EDITOR: JOSEPH NORRIS COMPUSERVE: 71106, 1302

UNCLASSIFIED ADS

FOR SALE: CCII, v6.78, 32K, deluxe keyboard, switchable lower case, handshake option, printer cable, sound, dust covers. Over 70 disks including Helm's Data Base, Screen Editor, Print; TermII, CC's Income Tax, plus many more. Many manuals and magazines. $1500 postpaid. Phil Simon, 6275 Cary Avenue, Cincinnati, OH 45224, 513-681-8370.

FOR SALE: CCII, v6.78, 32K, maintenance manual, all back issues of Colorcue, and the following software: Personal Data Base, Basic Language Tutorial 1-10, formatter, Basic editor, screen editor, assembler, Fortran, plus miscellaneous games and disk utilities. The computer is stil in good working order. $700, money order or certified check postpaid. Bill Anthony, 655 E. Wells Way, Camano Island, WA 98292. (206) 387-1576.

FOR SALE: CCII in good working condition, v6.78, 32K, 1 internal drive, extended keyboard. 'THE' Basic Editor in EPROM, Colorword, Tiny C, Tiny Pascal, and Forth. Lots of utilities and games, Maintenance manual, all Colorcue's. $500. Michael Burcham, 1707 Gleason, Iowa City, IO 52240. 319 354-2131.

FOR SALE: CCII, v6.78, 32K, 2 disk drives, switchable lower case, 101 key keyboard, and two Soundware attachments. In excellant condition. Programs include 10 Soundware games and 35 additional games, 20 applications including Equity, Bonds and Securities, Statistics I, text and Basic editors, Personal Data Base, Basic Tutor, assembler. Manuals include the Programming Manual, Maintenance Manual, Colorcue II-1 through II,6, Assembler Operating Manual. $1095, shipping prepaid. Mike Rousse, 15 South Owen Drive, Madison, WI 53705. (608) 238-1825, or leave message at (608) 233-6751.

COLORCUE is published bi-monthly. Subscription rates are US$12/year in the U.S., Canada, and Mexico (via First Class mail), and US$30 elsewhere (via Air Mail). All editorial and subscription correspondence should be addressed to COLORCUE, 19 West Second Street, Moorestown, NJ 08057, USA. (609-234-8117) All articles in COLORCUE are checked for accuracy to the best of our ability but cannot be guaranteed error free.

from the Editor's Desk

"Some assorted thoughts, thanks and numbers."

I suppose the greatest reward for an editor is the privilege of producing an issue like this one. So many articles have been submitted, and of such high quality and excitement, that I feel a sense of having been renewed to meet the challenge. Several people have asked me at one time or another why I would want to take over a 'dying' publication. Well folks, I'm very suspicious of 'death' in the first place, and tend to see it as only transitional in the second. With this issue we pass our mantel to CHIP, and before starting my next article for that august publication I have some old business to complete.

Special thanks to the following: my friend and wife, Susan Hardee Norris for tutoring and supervision in use of her spectacular typesetter and showing me how to communicate successfully on the modem, and for her forbearance; to Tom Devlin for keeping me alive in desperate moments; to Doug Van Putte for endless encouragement, moral support and a steady stream of articles; to David Suits, Rick Taubold, Chris Zerr, Peter Hiner and all of you who have submitted materials to COLORCUE during my tenure; to FRIENDS JOURNAL, in Philadelphia for their generousity in providing the typesetting facilities; to all of you who, through your continued subscription, afforded me this enviable opportunity to know and work with you. I have made many special friendships through COLORCUE, and I have not know such a splendid collection of people as I have found among CCII enthusiasts.

My thanks, too, to my employer, The David Hafler Company, for the opportunity to gain some valuable computer experience on Intecolor products and for the motivation afforded by their utility of my computer output.

Some statistics: letters received, 387; letters written, 532; subscription revenues, $3800; publication costs and expenses, $5300; current subscribers, 205; pages in Volume VI, in excess of 200; subscriber cost per page, about 10 cents.

I hope you will expend the effort to assemble Tom Napier's program in this issue. It is simply spectacular, and an extraordinary example of first rate programming. If you would like a measure of his talent, read the article in Scientific American and then look at his code. You may find it, as did I, a very humbling experience. John Newby's gift is beyond value. Any subscriber interested in pursuing a hard disk installation is encouraged to contact John or me for all possible assistance.

Any materials submitted for publication that didn't make it in this issue will be forwarded to CHIP for their consideration....and get your subscription into CHIP!

With my very best wishes,

Joseph Norris

"....*recursion*?"
"What the dickens is '*recursion*'?"

Doug Van Putte
18 Cross Bow Drive
Rochester, NY 14624

When an object is defined by the application of a simpler case of itself, we have a recursive definition. Recursion can be useful in programming, as we shall see, but first let's explore the definition further. Consider a recursive definition of the factorial of the number 4. Ordinarily, we think of the factorial of 4 to be

```
4! = 4 * 3 * 2 * 1
```

But the factorial of 4 could also be expressed as

```
4! = 4 * 3!
```

which is a recursive definition, since 3! is used to define 4!.

Another distinctive feature of a recursive definition is that it must lead to a definite ending point. Let's expand the definition of 4! by continually repeating the above definition:

```
4! = 4 * 3!
3! = 3 * 2!
2! = 2 * 1!
1! = 1 * 0!
0! = 1
```

Our ending point is 0!, which is '1' by mathematical definition. Now let's compute the factorial by back-tracking from the ending point:

0! = 1

1! = 1 * 0! = 1 * 1 = 1

2! = 2 * 1! = 2 * ! = 2

3! = 3 * 2! = 3 * 2 = 6

4! = 4 * 3! = 4 * 6 = 24

We have computed our way back to the correct answer from the ending point using recursion.

The BASIC program in Listing I uses this same scheme to compute the factorial of a given number. After providing for the entry of a number whose factorial is to be computed, the main program stores this number in the first element of a 'stack', A(1). Then it successively breaks down this number by subtracting one from the number until zero is reached, each time storing the intermediate result in the next available location of the stack.

As long as the number entered is not zero (0! = 1), control is now sent to the subroutine. The subroutine will begin the factorial calculation by starting with the bottom element in the stack, A(I) = 1, and storing the factorial value of that element in FA. Note that unless there is only one element in the stack, the subroutine continues the calculation by calling itself recursively. Then, by the scheme shown above, when the first element is reached (our original number, at last!), the last factorial is computed and control is transferred back to the main program where the factorial of the number entered is printed.

```
00010 REM ****************** LISTING 1 ********************
00020
00030 REM ********RECURSIVE ROUTINE TO COMPUTE N!*********
00040
00050 PLOT 12:DIM A(20):REM INCREASE DIMENSION FOR N>20
00060 INPUT "ENTER N ";N:NN=N
00070 IF NN=0 THEN FA=1:GOTO 160:REM SIMPLE CASE OF N=0
00080 I=0:REM INITIALIZE STACK INDEX
00090 REM TAKE APART N ONE BY ONE AND PUSH VALUES ON STACK
00100 I=I+1
00110 A(I)=N
00120 N=N-1
00130 IF N>0 THEN 100:REM DON'T STOP UNTIL N=0
00140 FA=1:GOSUB 200:REM 0!=1 IS WHERE FACTORIAL VALUE (FA)
00150 REM IS INITIALIZED
00160 PRINT:PRINT "FACTORIAL ";NN;" IS ";FA:PRINT
00170 GOTO 60:REM LETS GO AROUND AGAIN
00180 REM SUBROUTINE: POPS A VALUE FROM THE Ith LOCATION OF A
00190 REM STACK & COMPUTES ITS FACTORIAL BY USING A(I)*A(I-1)!
00200 IF I=0 THEN RETURN:REM ENDING POINT TEST
00210 FA=A(I)*FA:REM FACTORIAL ACCUMMULATION
00220 I=I-1:REM RESET STACK POINTER
00230 GOSUB 200:REM RECURSIVE CALL
00240 RETURN:REM END OF SUBROUTINE
```

PETER HINER:

A biographical sketch.

The editor has requested a brief biographical sketch of all the Colorcue authors. Peter Hiner has graciously responded and so we present this background on such an extraordinary talent as the author of a compiler must be.

Peter Hiner is 40 years old, married with two children, a boy of four and a girl of 18 months. He is employed by STC (Standard Telephones and Cables) as a System Design Manager in the Switching Division, which supplies telephone and data switching equipment, primarily to British Telecom. His engineering background is in logic and switching design for electronic switching systems. He claims his primary source of experience with software has been through the CCII which he bought in 1979. How did he get started?

"I followed what is probably the usual route of transcribing Basic games from magazines, to get used to our dialect of Basic, and then (had) a go at writing simple programs for games and graphics displays. From the beginning I had always believed that Assembly Language programming was the purest and most noble form of the art, so I soon started trying to break into this area. I am sure the first hurdles are the most difficult and the written materials now available for the beginner (particularly on input and output routines) must be of great help to those taking their first steps today. I tried the impossible task (for a beginner) of getting the required input and output routines from ROM, using a disassembler. This exercise left me totally lost and confused, although it probably started me on the path to writing a Basic compiler.

"I wrote a rudimentary sort of Invaders game—not very thrilling—and then got hooked on the idea of writing a ver-

The problem with making a recursive call in BASIC is that of preserving the values of the identifiers (e.g. the decreasing integer values in the above example.) This is done in our example by using a stack. The integer values are pushed onto the stack before the recursive call, and later popped off the stack in reverse order when executing the call. Some other languages, such as Pascal and 'C', can simplify this process considerably by 'keeping track' of the identifiers in a less cumbersome way.

"Can this recursion do anything useful?", you might ask. The answer is "You bet!". For example, consider the binary search program in Listing II. In this case we don't need a stack, just a scheme with a definite ending point. The scheme looks for a given value in an ascending array (it could be descending) and, if found, prints the index in the array where a matched element is located. A zero is printed for the index if a matching element is not found. The subroutine begins by comparing the middle element of the array with our value. If a match is not found, the array is split in two and only one half continues to be searched by the routine, recursively calling itself. Each time, then, the remaining part of the array is split until the middle element matches our value, or until the ending point is reached with no match. The ending point is reached when the lower bound of the array to be searched exceeds the upper bound. In either case (match or no match), control is returned to the main program to print out an index value. The value of recursion, here, is that it has been used to anchor a search method which is far more efficient than a simple linear search.

You might experiment with these principles by developing your own recursive routine in BASIC to raise a number to a power. I'll send you the answer to that one on request. I'd be interested in seeing any of your efforts.

```
00010 REM ****************** LISTING 2 ******************
00020 REM
00030 REM ************ BINARY SEARCH PROGRAM ************
00040 REM
00050 PLOT 12:DIM A(21 ):BM=1:TP=21:REM TP IS NO. IN ARRAY
00060 FOR I=BM TO TP:REM READ IN ARRAY TO BE SEARCHED
00070 READ A(I)
00080 NEXT I
00090 PRINT:INPUT "ENTER ELEMENT FOR SEARCH ";X
00100 L=BM:H=TP:REM INITIALIZE SEARCH LIMITS
00110 GOSUB 130
00120 PRINT "ELEMENT INDEX IS ";B:GOTO 90
00130 IF L>H THEN B=0:RETURN:REM TEST FOR ENDING POINT
00140 M=INT ((L+H)/2):REM COMPUTE ARRAY MID POINT
00150 IF X=A(M) THEN B=M:RETURN:REM MATCH W/ MIDDLE ELEMENT?
00160 IF X<A(M) THEN H=M-1:GOTO 180:REM SEARCH BOTTOM 1/2 ARRAY
00170 L=M+1:REM SEARCH TOP 1/2 ARRAY
00180 GOSUB 130:REM RECURSIVE CALL
00190 RETURN:REM RETURN TO MAIN
00200 DATA 1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41
```

---

sion of the original adventure game to fit a 16K CCII machine without repeated disk reads. This was a magnum opus, using quite sophisticated techniques. The program consisted of a small interpreter (about 2K) driven by a 4K table of data. The huge amount of text (25K) for descriptions and messages was compressed into about 10K using a mixture of dictionary and other text compression techniques. This huge task took about a year to complete, but left me with a powerful tool for writing other adventures, and I was later able to produce a version of Scott Adam's Pirate Adventure in a few weeks.

"After that, I started on my Basic compiler which grew over a couple of years from humble origins to a complete implementation. By-products of this process were a tokenized form of Assembly Language to reduce file size, a pair of programs for disassembly and assembly, and a program for comparing files.

"I am approaching the end of the road as far as utility programs is concerned and am now turning my attention to artificial intelligence. I am interested in some simple forms of export system to assist in the process of reaching decisions based on a mixture of fact, experience and opinion. I don't expect that a Compucolor would support much more than a very simple system, but it should be an interesting field for experiment."

(The disassembler Peter refers to is available from the CHIP library. It permits entry of your own labels, and is designed to give every programming aid when disassembling totally foreign code. ED)

Peter includes in his note this challenging invitation: "One of the by-products of ZIP has been an Integer Interpreter with built-in debugging aids. It would be a relatively simple task to provide a similar disk-based version of normal Basic with built-in debugging aids, or anything else built in, if there is a requirement. The only limitation will be the reduced space available for user programs. If you have any ideas on this subject, let me know. I would intend to provide this Basic interpreter as a free program for the CHIP library."

A HARD DISK FOR THE CCII

John Newby
4532 167th Avenue SE
Issaquah, WA 98027

For years I have operated with the limited storage capacity, the read/write errors, and speed variations of the Compucolor Disk (CD) drives. My CCII is used for many functions at work and I have an incredible amount of custom software, most of which uses large data files. An increasing disk problem has suggested a change of computer systems, even with the necessary conversion effort. I have always dreamed of a system with a Hard Disk (HD), but never figured one for the good old CCII.

In the March, April, and May 1983 issues of Byte, there appeared a series of articles by Cruce and Alexander on a hard disk interface for S-100 systems. It did not seem to be all that complex, with the exception of having to write an entirely new CPM type disk operating system. However, hard disk systems were going for nearly $2,000—much too much to sink into a CCII without even knowing if it would work. Even so, I began to consider the possibility.

In April 1984, a Seattle surplus electronics outlet, United Products Inc., advertised 5M byte hard disk drives for $250. I investigated and found they also had a nice controller for only $200 and a suitable power supply for $40. I had to try, so I purchased these two units.

For a week I pondered the construction of a disk operating system that would keep a map of all sectors and could randomly write to the disk as does CPM. This would eliminate the FCS problem of having to move all following files in order to delete a file. I could not stand to see the hard disk drive sitting around, so I decided to construct an interface adapter and write a short FCS compatible disk handler to see if I could get the thing working. I have never looked back.

Within two weeks, I put most of my floppy disks away and was having a great time. I discovered that FCS is not limited to 64K byte files. Since it is block structured, the actual file size limit is 64K sectors or 8.39M byte. Disk read/writes are so fast that file deletes are not much of a problem, particularly with a good file utility program. All FCS commands are operational, and any program that does not have its own disk routines will work with the HD. Other software, like Jim Helms' Data Base, requires some modification. However, the hard disk makes the data base much more effective. I regularly work with 250K to 300K byte data files without touching a floppy disk.

I have read horror stories about disk crashes, and it would take over 100 CD floppy disks to backup the hard disk. I thought about a tape drive backup system, but it is fairly expensive. I made another trip to United Products. This time they had just received in stock remanufactured 10M byte drives and were selling them for $275. Hmmm. The controller and power supply on the 5M byte drive will handle two drives, so why not use the second for backup? Done! I am now operating a 10M byte drive (actually 11.90M bytes) and using the 5M byte drive (actually 5.95M bytes) as a backup. It takes about 12 minutes to fill the backup drive. The hard disk system works very well and I do not think I would again use a CCII without a hard disk system.

Here are some of the features of the hard disk system:

1. The FCS HD handler is located in the open ROM space (4000-5FFFH), and only occupies 805 bytes. You will need to have either a single rom board or Freepost's bank select ROM board. The FCS system ROM also requires slight modification.

2. The disk drive is logically configured as eight 1.49M byte devices, HD0: to HD7: (four for a 5M byte drive).

3. Each device directory defaults to a 32 sector size (the maximum allowable in FCS); however, there is room for 191 files on each logical device.

4. There is no limit to file size since FCS can access up to 64K block (8.39M byte) files.

5. All FCS commands are operational, including copying between CD and HD devices.

6. The system is fast. As a benchmark, a 32K byte write on the HD takes 2.0 seconds compared to 30 seconds for the CD. A 32K byte read is 1.5 seconds compared to 13 seconds. When you load screens, they 'pop' right up. For disk intensive operations, such as loading LDA files, data base sorts, or Fortran compiling and linking, the speed increase is greater.

This article describes the major elements of a hard disk system consisting of the following items: a custom CCII SASI Hard Disk Interface Adapter, a hard disk controller, one 10M byte or 5M byte hard disk drive, a power supply, enclosure, and cable, and changes to the FCS operating system. I suggest you also read the series of articles by Cruce and Alexander.

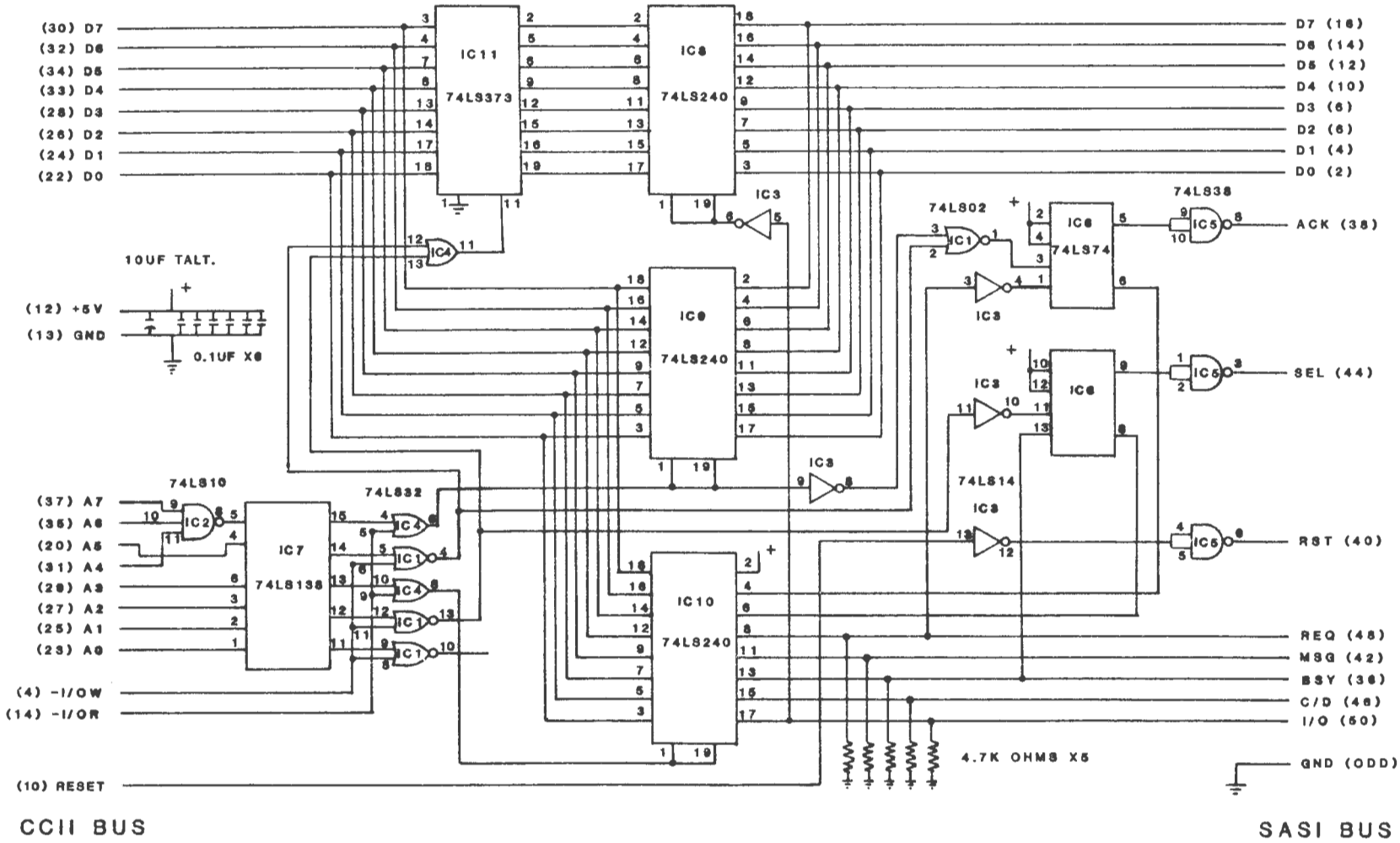
You will need to write a utility program to format and check the drives. If you use another drive as backup, you will need to write an additional utility routine. This is not too difficult, and all of the subroutines required are in the HD handler presented here. If you wish, I will send you the source for my utility routines.

As an aside, I also built an EPROM programmer which plugs into the CCII fifty pin bus and will provide the circuit diagram and software required to anyone who asks. It makes life as a CCII hacker much simpler.

THE SASI INTERFACE

The Shugart Associates Standard Interface (SASI) is an industry standard interface for parallel data and control signal transfers between a host computer and a hard disk controller. The circuit diagram for the CCII SASI interface

Figure 1. Compucolor II SASI Hard Disk Interface Adapter



board I built is shown in Figure 1. To the best of my knowledge, this interface should work with any SASI compatible hard disk controller.

The standard connector between the interface and controller is a fifty connector ribbon cable. There are two eight line groups, one for the data transfers and the other for control signals. The signals are active low and are defined as shown in Table 1. All odd numbered lines are grounded to improve noise immunity. In addition, the cable length should not exceed twenty feet.

Signal	Connector	Description
-DB0	2	Bi-directional tristate data bus.
-DB1	4	
-DB2	6	
-DB3	8	
-DB4	10	
-DB5	12	
-DB6	14	
-DB7	16	
-BSY	36	Set active by controller during each command sequence. A high level means ready for next command sequence.
-ACK	38	Set active by host in response to -REQ from controller to complete handshake.
-RES	40	Set active by host to reset controller. Must be low for at least 100 nS.
-MSG	42	Set active by controller to indicate the command sequence is complete.
-SEL	44	Set active by host to intiate a command sequence.
-C/D	46	Set active by controller when a command is on data bus. High means data.
-REQ	48	Set active by controller to intitiate a byte transfer handshake.
-I/O	50	Set active by controller to indicate input to host.
GND	1-49	All odd signal lines are connected to a common ground.

The interface board is accessed as a group of I/O ports from the CCII. Port address decode logic is located in the lower lefthand portion of the circuit diagram. Five port addresses are allowed, as listed in Table 2. I never implemented the software reset port but my hardwired reset has worked very well. If a software reset is desired, a one-shot should be added to provide the required reset pulse width.

The octal line drivers (74LS240) on the interface board invert the SASI signals so that the CCII receives normal, active high signals. Therefore, the data lines are correct and the control byte received from port 0DAH is as shown in Table 3. The controller latchs the data and control lines it is transmitting or receiving. Data transmitted by the CCII is latched by the 74LS373. The -SEL and -ACK control signals are latched by the flip-flops in the 74LS74.

TABLE 2. PORT ASSIGNMENTS

Port	Action	Function
0D8H	Read	Read data from controller
0D9H	Write	Write data to controller
0DAH	Read	Read control signals
0DBH	Write	Write select byte to controller
0DCH	Write	Send reset signal (not implemented)

Each command sequence occurs as follows:

1. The CCII selects the controller by waiting until it is idle, then writing a select byte to port 0DAH. The write sets a flip-flop in the 74LS74 to send -SEL. The controller responds by asserting -BSY. This clears the flipflop to turn off -SEL. Then controller then activates -C/D to indicate a command, but leaves -I/O deasserted to indicate output to the controller.

2. The controller requests a byte by asserting -REQ. The CCII writes a byte to port 0D9H. This write also sets another flip-flop in the 74LS74 to send -ACK to the controller. The controller reads the byte and deasserts -REQ, which also clears the flip-flop to turn off -ACK. This handshake sequence is repeated until the entire six byte command is transfered. All subsequent byte transfers use the same request/ acknowledge handshake.

3. If the command sequence involves a data transfer, the controller will deassert -C/D to indicate a data transfer and sets -I/O as appropriate. Data is then transfered using the same request/ acknowledge handshake as for the command bytes, with either a CCII read from port 0D8H or write to port 0D9H sending an -ACK to the controller.

4. At the end of a command sequence, the controller sends a completion byte which indicates if any errors occured. The controller then asserts -MSG and a message byte (0)is sent to the CCII. All control signals are deasserted and the controller returns to an idle state.

The circuit shown in Figure 1 is comprised of relatively inexpensive parts and is easy to build. As the CCII fifty pin bus is not buffered, it is important that the interface board be plugged directly into the CCII and that the CCII data and address line lengths are kept to a minimum. The circuit should be constructed on a board with a ground plane passing beneath each integrated chip, and ground leads should be short. The power decoupling capacitors should be placed at regular intervals and should also have short leads. Table 4 lists the integrated circuits and indicates their power and ground connections.

Once you have completed construction of the SASI interface, make sure that the power lines (or any others) are not shorted. Connect the interface to your CCII and check out all data lines by writing bytes out port 0D9H and reading port 0D8H. If you do not read the same value, you probably have crossed data lines somewhere. To check the control lines, write a byte to port 0D9H and you should read 05FH on port 0DAH.

Table 3. CONTROL BYTE

Bit	Signal	Level	Function
0	I/O	0	CCII => Controller
		1	Controller => CCII
1	C/D	0	Data on bus
		1	Command on bus
2	BSY	0	Controller idle
		1	Controller in command sequence
3	MSG	0	Normal condition
		1	Transfer complete, status byte sent
4	REQ	0	No request from controller
		1	Controller ready to send/receive
5	SX	1	Select output confirmation
6	AX	1	Acknowledge output confirmation
7	RES	X	Reserved - Forced to 0

The hard disk controller is an intelligent device which operates hard disk drives (generally two to four drives per controller). The controller is sent simple commands by the computer and performs the required disk functions. The controller is connected to the disk through another interface. Most five and one-quarter inch drives use an industry standard Seagate Technology 'ST506' interface. If you wish to know more about this interface, please refer to the articles by Cruce and Alexander in Byte. The controller handles all of the signals required by the hard disk; therefore, you only have to know how to connect the cables.

Most hard disk controllers which do not use a direct memory access interface, use the SASI interface. This includes almost all inexpensive controllers. In selecting a controller, you should be sure that it uses both SASI and ST506 interfaces and supports 128 byte sectors for compatibility with FCS.

TABLE 4. INTEGRATED CURCUITS

Number	Type	+ 5 Volts	Ground
IC1	74LS02 2-INPUT NOR GATE	14	7
IC2	74LS10 3-INPUT NAND GATE	14	7
IC3	74LS14 SCHMITT INVERTER	14	7
IC4	74LS32 2-INPUT OR GATE	14	7
IC5	74LS38 2-INPUT NAND BUFFER	14	7
IC6	74LS74 DUAL D FLIP-FLOP	14	7
IC7	74LS138 1-OF-8 DECODER	16	8
IC8	74LS240 OCTAL BUFFER	20	10
IC9	74LS240 OCTAL BUFFER	20	10
IC10	74LS240 OCTAL BUFFER	20	10
IC11	74LS373 OCTAL LATCH	20	10

ST506 AVAILABILITY.

Here is a list of sources for the ST506 Hard disk drive (and units that are compatible with the ST506) as well as other components in the hard disk system. Prices are from May 1985. [*] indicates a possible source for controllers. These sources are from COMPUTER SHOPPER. COLORCUE has not checked any of these sources nor do we necessarily recommend them. This listing is for your convenience only. Some units may be new, others used and tested, still others 'as is.' Proceed with a clear head!

HARD DISKS AND CONTROLLERS.

Advanced Computer Products, Inc. 1310 E. Edinger, Santa Ana, CA 92705. (800) 854-8230 or CA: (714) 558-8822. 6 MBytes: CHST506, $199.00; Shugart SA604, $119.00. 10 MBytes: Seagate ST419, $299.00.

* W W Component Supply, Inc. 1771 Junction Avenue, San Jose, CA 95112. (408) 295-7171. 5MBytes: Shugart SA604, $ 149.00.

Walker Electronics Company. 3521 Hacienda, Dallas, TX 75233. (214) 339-4916. ST506, $175.00.

* Steve, (513) 433-1501. ST506, with manual, $249.00. Call after 6 PM.

* Computer Products and Peripherals Unlimited. 18 Granite Street, Haverhill, MA 01830. (617) 372-8637.

* Digital Search. (803) 877-9444. Source for many drive products.

* Met-Chem International Corp. 2911 Dixwell Avenue, Hamden, CT 06518. (203) 248-3212, (800) 638-2436. Bulletin board (300/1200 baud) (203) 281-7287.

John Hanson. 1110 Pheasant Circle, Winter Springs, FL 32708. (305) 699-0124. ST506, $195.00.

(Unsigned) (916) 726-3291. ST506, $95.00.

POWER SUPPLIES

Jameco Electronics, 1355 Shoreway Road, Belmont, CA 94002. (415) 592-8097. Kepro TDK $59.95 (12 volts at 2 A)

Nicorn Electronics. 10010 Canoga Avenue, Unit B-8, Chatsworth, CA 91311. (8180 341-8833. Apple Power Supply (12 volts at 2.5A)

H. J. Knapp of Florida, Inc. 4750 96th Street, St. Petersburg, FL 33708. (813) 392-0406. $29.95. (12 volts at 3.5A)

Computer Products and Peripherals Unlimited. (See address above) Model A. $29. (12 volts at 3A)

B. G. Micro. PO Box 280298, Dallas, TX 75228. (214) 271-5546. (12 volts at 2.8A, 12 volts at 2A) Recommended. $37.50.

United Products, Inc. 1123 Valley Street, Seattle WA 98109. (206) 682-5025. TDK Model 21145. Recommended. $34.50. (12 volts at 2.8A and 2.0A)

INTEGRATED CIRCUITS

JDR Microdevices. 1224 S. Bascom Avenue, San Jose, CA 95128. (800) 538-5000.

Jameco. (See address above.)

CABLES

Altex Electronics. 10731 Golfdale, San Antonio, TX 78216. (800) 531-5369.

COMPUTER SHOPPER also regularly advertises enclosures which are suitable for a CCII hard disk installation. If you wish to subscribe to this valuable publication, write to them at PO Box 1419, Titusville, FL 32781. $15.00/year.

TABLE 5. OMTI 20L COMMAND SUMMARY

Code	Command	Explaination
00H	SENSE STATUS	Check for drive ready
01H	RECALIBRATE	Step out until Track 0
03H	REQUEST SENSE	Report detailed error codes
04H	FORMAT DRIVE	Format entire drive
05H	CHECK TRACK	Check entire track for errors
06H	FORMAT TRACK	Format a single track
07H	FORMAT BAD TRACK	Write defective bit in ID field
08H	READ DATA	Read 1 to 256 sectors
0AH	WRITE DATA	Write 1 to 256 sectors
0EH	ASSIGN ALT TRACK	Set alternate track bit in ID field
C2H	ASSIGN PARAMETERS	Assign hard disk variable parameters
E1H	WRITE ECC	To allow ECC testing
E2H	READ ID	To read ID field
E6H	REQUEST LOGOUT	Read retry and error counts
EAH	READ ECC	To allow ECC testing
ECH	READ DATA BUFFER	Read buffer only, not disk
EFH	WRITE DATA BUFFER	Write buffer only, not disk

I used an OMTI 20L controller. Its general characteristics include:

1. A 10K byte buffer to allow reading and writing of a full track at one time by the controller.

2. Controller logic allows 128, 256, or 512 byte sectors.

3. A single ID field is used for each track, allowing for thirty-eight 256 byte sectors (seventy-six 128 byte equivalents), as opposed to standard formatting which results in only 32 sectors. Therefore, the capacity is 5.95M bytes on a 5M byte drive and 11.90M bytes on a 10M byte drive.

4. Four bytes of error correcting code (ECC) is written for each 256 bytes. This allows automatic correction of up to five bits per 256 bytes during disk reads.

5. Data written to the disk can be automatically verified.

6. Up to 256 sectors can be transfered in a single read or write command.

7. Device size limited to 2 to the 15th power sectors (536M bytes for 128 byte sectors) or 65K tracks (637M bytes).

8. The controller is on a single 5.75 by 8.00 inch PC board which mounts directly on the hard disk drive. (I had to place a copper PC board ground plane between the controller and hard disk circuit boards to eliminate interference.)

Table 5 summarizes the SASI commands supported by the OMTI 20L controller. If bad sectors are found on any track, alternate tracks can be assigned during formatting; from then on, access to these tracks is transparent to the host computer. However, I have not encountered a bad track in three hard disk drives, two of which were remanufactured.

The hard disk handler code described in this article should work with minor changes, if any, on most any SASI compatible controller which supports 128 byte sectors. The handler only uses the normal Sense Status, Request Sense, Read Data, Write Data, and Assign Parameters commands. A separate utility program must be written to format and check the drive (and assign alternate tracks, if necessary).

OMTI no longer produces the 20L controller; however, there may still be some on the surplus market (United Products is sold out). OMTI's current comparable controller is the 5200 series. This controller has the same, or better, features and supports 128 byte sectors, but uses the standard thiry-two 256 byte sectors per track. However, it also supports eight inch disk drives. OMTI is now part of Scientific Microsystems, and their products are distributed by Arrow Electronics.

The industry standard controller is the Data Technology Corporation DTC-500 Series. These are carried by Active Electronics and others. However, I do not know if these controllers support 128 byte sectors.

THE HARD DISK DRIVE

The hard disk drive is comprised of several sealed metal media disks rotating at 3600 rpm. There is a read/write head for each surface (two per disk). The ST506 drive has two disks and four heads. Each surface has 153 tracks for a total of 612. The ST412 also has four heads; however, later technology allowed for 306 tracks per surface, doubling the capacity. The ST412 also has a much faster head stepping speed.

About the only requirement for hard disk selection is that it should be 'ST506 compatible,' and most five and one-quarter inch hard disk drives are. The controller must be assigned the proper parameters using the Assign Parameters command. Variable parameters you will need to know include those shown in Table 6.

Seagate Technology hard disk drives have a 16 pin (8 connection) option shunt block which must be inserted to set customer options. This includes the drive select encoding. For ST506 and ST412 drives, you should shunt pins 2-15, 4-13, and 8-9 (DS1) or 7-10 (DS2). Your drive supplier should provide you with a description of how to set the options for your drive.

THE POWER SUPPLY, ENCLOSURE, AND CABLES

Most hard disks require power supplies which can provide + 12 volts at 3 to 4 amps peak (2 amps continuous) and + 5 volts at 1 amp. The peak 12 volt current is only required while the drive is coming up to speed. I used a Kepro switching power supply which is generally available on the

TABLE 6. DRIVE PARAMETERS WITH OMTI 20L CONTROLLER

Name	ST506	ST412
Step Pulse Width	3 uS	2 uS
Step Pulse Period	3 mS	50 uS
Step Mode	normal	normal
Number of Heads	4	4
Total Number of Tracks	612	1224
Reduced Write Current Track	128	128
Sectors per Track	76	76

TABLE 7. PARTS COST FOR HARD DISK SYSTEM

SASI Interface Board and Parts	$ 50.00
All Cables and Connectors	75.00
Kepro Power Supply	35.00
Heathkit Enclosure Parts	90.00
Fan	25.00
OMTI Controller	200.00
Seagate Technology ST412 Drive	275.00
Subtotal	750.00
ST506 Backup Drive	250.00
Total	1000.00

surplus market for less than $50. As long as I only power up one drive at a time, this supply runs a ST506 drive, a ST412 drive, and the OMTI controller.

I have used Heathkit H-77 disk drive enclosures for both my two CD drives and my two HD drives. The cabinets look nice and have plenty of room for two drives and a power supply. You can order the eleven pieces required from Heathkit for about $90. [See parts list in Table 8.] You should also put a fan on your enclosure to provide plenty of ventilation. Hot components are not known for their reliability or long life.

Connection cables are a significant item in putting togeather a hard disk system. You will need the following:

1. A fifty line ribbon cable from the interface to the controller, along with edge or pin connectors.

2. A thirty-four line ribbon cable from the controller to each drive in a daisy chain manner, along with connectors.

3. An individual twenty line ribbon cable from the controller to each drive, along with connectors.

4. A four wire connection from the power supply to each drive and to the controller. The connector is an AMP 1-480424-0.

THE OPERATING SYSTEM CHANGES

All FCS routines access storage devices by setting up a series of parameters and then calling a handler for the current device type. There is a table in the FCS ROM which lists the power up default device type and number. This is followed by a jump to the handler, a device name, and the maximum device number for each supported device. To support the 'HD' drives, the disk lookup table in the FCS rom must be modified as shown in Listing 1.

The hard disk handler can be located anywhere in the 4000-5FFFH ROM space. In order to support Freepost's multiple bank ROM board, the jump to the handler in FCS goes to a patch (Listing 2) at the end of each ROM bank. This patch jumps to the handler located in Bank 0. In this manner, the hard disk can be accessed from any bank with the patch at the end.

NOTE: There is a routine called 'RESET' which is called on entry to FCS. RESET sends a turn off function to each device in the device table. If you are in a bank (including the RAM bank) that does not have the patch in place, your system will hangup. For the RAM board, you can CPU reset; ESC W to Basic Reset; set the bank by OUT 255,7; defeat calls to the hard disk handler by POKE 24542,201; and then ESC D to FCS.

My hard disk handler code is presented as Listing 3. It is for a Seagate Technology ST412 disk drive and an OMTI 20L controller. The modifications for a ST506 drive are also indicated. If you use another drive, the code will need to be appropriately modified. A good manual on the controller you select is a must. The handler occupies only 805 bytes, leaving plenty of room for your other utilities or programs.

There is nothing magic about the logical device size I selected. You can easily modify the handler and device table to support other numbers of logical devices. The number of sectors per device should be an even division of the total number of sectors on the hard disk, and must not exceed 64K sectors (FCS's limit).

The only limitation with the handler, as I have implemented it, is that you cannot read or write data directly from a RAM card occupying the 4000-5FFFH memory area, unless you load a copy of the handler in the RAM bank. I get around this by either using a CD disk or by using a simple utility program which loads the data in upper memory from the hard disk, switches to the ram bank, and then moves the data down.

CONCLUDING REMARKS

The total cost to setup my hard disk system is about $1000, and can be broken down approximately as shown in Table 7. I have recently seen ST506 drives on the surplus market for less than $175 and new Shugart 604 6.7M byte drives advertised for $139.

When you sit back and consider having eight 1.5M drives on line, very fast disk access, essentially unlimited file size, full FCS compatibility, and not having to mess with CD disk drives and limited capacity disks, it all seems well worth the effort and price. My CCII is now so powerful and so much fun to use I will probably stick with it for several more years. Good old Serial No.16 is a lot different than when it rolled off the assembly line in 1978. When I eventually move on, the entire hard disk system (beyond the interface board), which is industry standard, and can move on with me.

References.

Andrew C. Cruce and Scott A. Alexander. "Building a Hard-Disk Interface for an S-100 Bus System", a three part article in Byte: March, April, and May 1983.

Jim Thoreson. "The Winchester Odyssey, from manufacturer to user." Byte: March 1983.

"ST506 OEM Manual." July 4, 1983. Seagate Technology, 920 Disk Drive, Scotts Valley, California 95066. 408/438-6550

Scientific Microsystems (OMTI). 339 North Bernard, Mt. View, California 94043, 415/964-5700.

United Products Inc. 1123 Valley, Seattle, Washington 98109 206/682-5025.

Commentary on the **HARD DISK**

Chris Zerr
10932-156th Court
Redmond, WA 98052

Wouldn't you know it! Just as COLORCUE ends, something new appears for the CCII to be shared with all users—a hard disk system. When John Newby called me, back in May of 1984, and told me about this I fell off my chair! Amazing! So last February I splurged and bought a ST506 Hard Disk, OMTI 20L controller and a power supply. The total cost was $506 [1], including cables and wire. I began to wire the SASI interface using a Radio Shack epoxy board and point to point wiring.

It took me a week to wire, working a few hours every night. Finally the big day arrived. I plugged it in and...hmmm. I had a broken wire and one wire misplaced. I corrected these and..Bingo..it worked. But this was only the interface being tested. The next step was to connect the interface to the controller and hard disk. This is so simple that there isn't much that can go wrong, right? All that remained was connecting two jumpers, wiring a select strap, and connecting a cable between the CCII interface and the controller card.

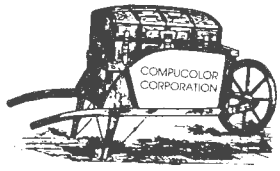
I turned on the power and typed DIR HD0: . FCS ERROR ENVE. Disk not initialized..whew! I next had to format the hard disk and initialize each directory. The formatting takes only thirty seconds or so, and initializing is nearly instant. Once completed, the experience is wonderful. To think, 6 megabytes online. Editing SRC files is a breeze. I do not have a second drive for backup so, for now, I only backup files I have changed or that are really important. As for software, I am currently using the Frepost bank board with the HD driver in Bank 0. So far I'm enjoying all of it. The CCII will keep me going for a few more years now. Should it die, I'll know that the hard disk can move on to another computer. Only the interface card, which costs about $50.00 will be lost.

I would like to give special thanks to John Newby for his time and effort in making this wonderful enhancement possible.

[1] The cost could also be as low as this for subscribers who want to investigate a hard disk for their CCII. See the list of possible sources for the components in this issue. ROM chips are available from John Newby. Joseph Norris might lay out a double sided PC card for the interface. You may place an order with him. The price is not known, but is expected to be about $40.00.

```
;       LISTING 1.  FCS ROM MODIFICATION
;
        CDHD    EQU     0211CH  ;(1AC1)
        HDHD    EQU     05FDEH  ;(5FDE)
;
        ORG     0368BH          ;(0040)
;
;       INITIAL DEVICE NAME AND NUMBER
;
        IDEV:   DB      'CD'
        IUNIT:  DB      '0'
;
;       COMPUCOLOR DISK HANDLER
;
        HDVCT:  JMP     CDHD    ;VECTOR TO HANDLER
        CDNM:   DB      'CD'    ;NAME
        CDNU:   DB      2       ;MAX. DEVICES
        CDSEC:  DB      00AH    ;SECTORS/TRACK
        CDK:    DB      050H    ;UP TO SPEED TIME
                DS      2       ;SPARE SPACE
;
;       INSERT HARD DISK HANDLER HERE
;
        HDHDL:  JMP     HDHD    ;VECTOR TO PATCH AT END OF ROM
        HDNM:   DB      'HD'    ;NAME
        HDNU:   DB      8       ;4 FOR ST506
                DS      4
;
;       ROOM FOR STILL ONE MORE
;
        OPNHD:  DB      0FFH
                DS      5
```

```
;       LISTING 2.  BANK ROM PATCH
;
        BFLAG   EQU     08042H  ;SPACE FOR CURRENT BANK
        BANK    EQU     0??H    ;THE CURRENT BANK
        HDHAND  EQU     0????H  ;WHEREVER YOU WANT IN 4000-5FFFH
;
        ORG     05FDEH
;
        HDHD:   MVI     A,BANK  ;THIS ROM BANK
                STA     BFLAG   ;SAVE IT FOR RETURN
                XRA     A       ;GOTO HANDLER BANK
                OUT     0FFH
                CALL    HDHAND  ;DO THE HANDLER OPERATION
                PUSH    PSW     ;SAVE STATUS
                LDA     BFLAG   ;GO BACK TO PROPER BANK
                OUT     0FFH
                POP     PSW     ;RESTORE STATUS
                RET
;
;       OTHER BANK ACCESS ROUTINES
;
        RRET:   POP     B       ;GET BACK BANKS
                POP     H       ;GET ORIG. CALLERS RETURN
                MOV     A,B     ;GO TO PROPER BANK
                JMP     RJMP
;
;       RCALL - USED TO CALL A ROUTINE IN ANOTHER BANK
;               HL = ROUTINE ADDRESS
;               B  = CALLING BANK
;               C  = ROUTINE BANK
;               NOTE: ROUTINE CAN NOT ALTER STACK
;
        RCALL:  PUSH    B       ;SAVE BANKS
                LXI     D,RRET  ;SET UP RETURN
                PUSH    D
                MOV     A,C     ;GET READY TO GO
;
;       THE NORMAL MINIMUM PATCH REQUIRED
;
        RJMP:   OUT     0FFH    ;GO TO BANK
                PCHL            ;SHOULD BE AT 05FFFH
```



```
LISTING 3.

COMPUCOLOR II - SEAGATE TECHNOLOGY/ OMTI 2ØL DISK HANDLER
FCS ROM VERSION 6.78 (8.79)
VERSION 1.Ø : 5/1/84 : ST5Ø6 DRIVE, 4 * 1.49M BYTES
        2.Ø : 9/1/84 : ST412 DRIVE, 8 * 1.49M BYTES, TRAILERS
        2.1 : 4/1/85 : FIXED POWER UP ERROR, INTERRUPT MASK

SYSTEM ROUTINES  6.78     (8.79) ******************************

CRLF     EQU     Ø338BH  ;(17C1) CR AND LF TO SCREEN
LBYT     EQU     Ø339BH  ;(17D1) LIST HEX BYTE TO SCREEN
OSTR     EQU     Ø33F4H  ;(182A) SEND STRING TO SCREEN
MOVDH    EQU     Ø343BH  ;(1871) MOVE B BYTES FROM HL TO DE
CMPHD    EQU     Ø344DH  ;(1883) COMPARE DE TO HL
SUBHD    EQU     Ø3459H  ;(188F) SUBTRACT DE FROM HL
PSPAC    EQU     Ø34B3H  ;(18E9) PRINT A SPACE
PCOLN    EQU     Ø34B8H  ;(18EE) PRINT A COLON
PSSTR    EQU     Ø34BDH  ;(18F3) PRINT A SPACE AND STRING
PSTR     EQU     Ø34CØH  ;(18F6) PRINT A STRING
BC2BK    EQU     Ø35A8H  ;(19DE) CONVERT BYTE COUNT TO BLOCKS
ERMSG    EQU     Ø3631H  ;(1A67) ERROR MESSAGE
HDNAM    EQU     Ø369BH  ;(ØØ5Ø) 'HD'

DATA AREAS **********************************************

AREAS OTHERWISE USED BY CDHD

SBC      EQU     Ø8Ø42H
CRC1     EQU     Ø8Ø43H
CRC2     EQU     Ø8Ø44H
BRTRY    EQU     Ø8ØEØH
RFLG     EQU     Ø8ØE1H
CRTRY    EQU     Ø8ØE2H
OSEC     EQU     Ø8ØEDH
SEC      EQU     Ø8ØEEH
TRK      EQU     Ø8ØEFH
TEMP2    EQU     Ø81F4H

AS USED BY HD HANDLER

BFLAG    EQU     Ø8Ø42H  ;CURRENT OPERATING BANK NUMBER
HALF1    EQU     Ø8Ø43H  ;NO. SECTORS TO READ/WRITE
HALF2    EQU     Ø8Ø44H  ;BALANCE OF SAME IF OVER 256
HDFLG    EQU     Ø8Ø43H  ;MORE THAN ONE ERROR FLAG
SECTØ    EQU     Ø8ØEØH  ;SECTOR ADDRESS MSB'S
SECT1    EQU     Ø8ØE1H  ;SECTOR ADDRESS
SECT2    EQU     Ø8ØE2H  ;SECTOR ADDRESS LSB'S
LBLK     EQU     Ø8ØEDH  ;LAST BLOCK BYTE COUNT
BKCNT    EQU     Ø8ØEEH  ;BLOCK COUNT FOR TRANSFER
ECODE    EQU     Ø8ØEDH  ;ERROR CODE
RSFLG    EQU     Ø81F4H  ;Ø = CPU RESET PERFORMED

THESE ARE THE SAME FOR ALL HANDLERS

TFCN     EQU     Ø8ØE5H  ;FUNCTION CODE
TDRV     EQU     Ø8ØE6H  ;DRIVE NUMBER
TBLK     EQU     Ø8ØE7H  ;BLOCK NO FOR TRANSFER
TMEM     EQU     Ø8ØE9H  ;MEMORY BUFFER POINTER
TBC      EQU     Ø8ØEBH  ;BYTE COUNT

ST412 DISK PARAMETERS    (ST5Ø6) ******************************

PLSEW    EQU     ØØ2H    ;(ØØØ3) STEP PULSE WIDTH; 1 uS EACH
PLSEP    EQU     ØØ1H    ;(ØØ3C) STEP PULSE PERIOD; 5Ø uS EACH
SMODE    EQU     ØØØH    ;(ØØØØ) STEP MODE; Ø = NORMAL
HDNU     EQU     ØØ8H    ;(ØØØ4) MAX. NUMBER OF DEVICES
NHEAD    EQU     ØØ3H    ;(ØØØ3) NUMBER OF HEADS / DRIVE -1
NCYL     EQU     Ø131H   ;(ØØ98) NUMBER OF TRACKS / DEVICE -1
RWCC     EQU     Ø8ØH    ;(ØØ8Ø) REDUCED WRITE CURRENT TRACK
DRIVE    EQU     ØØØH    ;(ØØØØ) DRIVE TYPE
NSEC     EQU     Ø4CH    ;(ØØ4C) NUMBER OF SECTORS / TRACK
MBLK     EQU     Ø2D6CH  ;(2D6C) NO. BLOCKS / LOGICAL DEVICE

PORT ASSIGNMENTS ******************************************

RXHD     EQU     ØD8H    ;READ SASI PORT DATA
TXHD     EQU     ØD9H    ;WRITE SASI PORT DATA
HSTAT    EQU     ØDAH    ;READ CONTROL DATA
SELHD    EQU     ØDBH    ;WRITE SELECT BYTE

ERROR CODES ***********************************************

OMTI 2ØL DRIVE ERRORS

NOERR    EQU     ØØH     ;NO ERRORS
NINDX    EQU     Ø1H     ;NO INDEX
NSEEK    EQU     Ø2H     ;NO SEEK COMPLETE
WRFLT    EQU     Ø3H     ;WRITE FAULT
DRVNR    EQU     Ø4H     ;DRIVE NOT READY
NTRKØ    EQU     Ø6H     ;NO TRACK ØØ FOUND

OMTI 2ØL DATA ERRORS

IDECC    EQU     1ØH     ;ECC ERROR IN ID FIELD
DATER    EQU     11H     ;UNCORRECTABLE DATA ERROR
NIDAD    EQU     12H     ;NO ADDRESS MARK FOUND IN ID
NDAAD    EQU     13H     ;NO ADDRESS MARK FOUND IN DATA
NOREC    EQU     14H     ;NO RECORD FOUND
SEKER    EQU     15H     ;SEEK ERROR
CORER    EQU     18H     ;CORRECTABLE DATA ERROR
BDTRK    EQU     19H     ;BAD TRACK FLAG SET
ALTER    EQU     1CH     ;ALTERNATE TRACK READ ERROR

OMTI 2ØL COMMAND ERRORS

IVCMD    EQU     2ØH     ;INVALID COMMAND
ILSEC    EQU     21H     ;ILLEGAL SECTOR ADDRESS
VOLOF    EQU     23H     ;VOLUME OVERFLOW

OMTI 2ØL HARDWARE ERRORS

RAMER    EQU     3ØH     ;RAM ERROR
DMAER    EQU     33H     ;DMA TIMEOUT

HD HANDLER ERRORS

IVUNT    EQU     Ø7H     ;INVALID UNIT
IVFCN    EQU     Ø8H     ;INVALID FUNCTION CODE
NOPWR    EQU     Ø9H     ;NO POWER TO OMTI 2ØL

; ***** THE HANDLER         *************************************

WHEN CALLED HL => BLOCK OF 8 BYTES, AS FOLLOWS:

        FCN: DS 1  ;FUNCTION CODE
        DRV: DS 1  ;DEVICE NUMBER
        BLK: DS 2  ;FIRST BLOCK NO. FOR TRANSFER
        BUF: DS 2  ;MEMORY POINTER FOR TRANSFER
        XBC: DS 2  ;BYTE COUNT FOR TRANSFER

FUNCTION CODES:

        - 4 = TURN OFF            NO ACTION
        - 3 = SUBTRACT USER       NO ACTION
```

```
;                - 2 = ADD USER          NO ACTION
;                - 1 = GET SIZE          SET TBLK = MAX. NO. BLOCKS
;                  Ø = READ DATA         READ DATA W/ ERROR CORRECTION
;                  1 = WRITE DATA        WRITE WITH AUTOMATIC VERIFY
;                  2 = VERIFY DATA       NO ACTION
;
        ORG     05000H          ;OR WHERE EVER YOU WISH
;
HDHAND: MVI     A,HDNU          ;MAX. NO. OF DEVICES
        CALL    HDPAR           ;GET PARAMETERS AND SETUP RETURN
        LXI     H,MBLK          ;BLOCKS PER LOGICAL DEVICE
        JP      REWRIT          ;IT IS A READ OR WRITE FUNCTION
        SHLD    TBLK            ;SAVE MAXIMUM BLOCKS
        POP     D               ;STRIP ERROR RETURN
        XRA     A               ;SET NO ERRORS
        RET                     ;ALL EXCEPT READ OR WRITE FUNCTIONS
;
;       GET TRANSFER DATA BYTES AND SETUP HANDLER RETURN ************
;
HDPAR:  POP     B               ;GET HANDLERS RETURN ADDR.
        LXI     D,HDRET         ;SET RETURN THROUGH ERROR ROUTINE
        PUSH    D
        PUSH    B               ;RESTORE RETURN ADDR.
        MOV     C,A             ;C = MAX. NO. OF DEVICES
        LXI     D,TFCN          ;POINT TO PARAMETER STORAGE
        MVI     B,008H          ;NO. BYTES IN PARAMETER BLOCK
        CALL    MOVDH           ;COPY INTO STORAGE AREA
        POP     H               ;RESTORE RETURN ADDRESS
        LDA     TDRV            ;DRIVE NO. ASKED FOR
        MVI     E,IVUNT         ;SET INVALID UNIT
        CMP     C               ;COMPARE TO MAX
        RNC                     ;IVU, RETURN TO ERROR
        LDA     TFCN            ;GET FUNCTION ASKED FOR
        CPI     2
        JC      OKDRV           ;READ OR WRITE
        MVI     E,IVFCN         ;SET INVALID FUNCTION
        CPI     -4              ;LOWEST POSSIBLE
        RC                      ;IVF, RETURN TO ERROR
OKDRV:  MVI     E,Ø             ;SET NO ERRORS
        ANA     A               ;TEST FUNCTION CODE
        PCHL                    ;TRICKY RETURN TO HDHAND

;       RETURN FROM PROCESSING ROUTINE  ****************************
;
HDRET:  JNZ     HDERR           ;ERRORS !!!!
        LXI     H,Ø             ;SUCCESSFUL TRANSFER
        SHLD    TBC             ;NO BYTES LEFT
        LHLD    BKCNT           ;GET NEXT BLOCK AFTER TRANSFER
        SHLD    TBLK            ;AND POINT AHEAD FOR NEXT PASS
        XRA     A               ;SET NO ERRORS
        RET
;
;       ERROR REPORTING ROUTINE         ****************************
;
HDERR:  XRA     A
        STA     HDFLG           ;SET NO ADDITIONAL ERRORS
        STA     ECODE           ;SET NON VALID SECTOR ADDRESS
        MOV     A,E             ;CHECK FOR IVU
        CPI     IVUNT
        JZ      HDREX
        CPI     IVFCN           ;CHECK FOR IVF
        JZ      HDREX
;
;       GET ERROR CODE FROM CONTROLLER
;
ELOOP:  MVI     A,003H          ;REQUEST SENSE
        LXI     B,000H          ;ERROR CORRECTION OK,RETRY OK
        CALL    COMND           ;DO IT
        CALL    DATOK           ;GET ERROR, SECTOR VALIDITY
        STA     ECODE
        CALL    DATOK           ;GET LOGICAL ADDR 2
        STA     SECTØ           ;SAVE ERROR SECTOR ADDRESS
        CALL    DATOK
        STA     SECT1
        CALL    DATOK
        STA     SECT2
        CALL    REASM           ;READ STATUS
        STA     HDFLG           ;INDICATE MORE ERRORS, IF SO
        LDA     ECODE
        ANI     03FH            ;STRIP SECTOR VALIDITY
        MOV     E,A             ;ERROR CODE IN E
;
;       PRINT ERROR CODES TO SCREEN
;
HDREX:  LXI     H,ERTBL         ;FIND ERROR CODE IN TABLE
        LXI     B,4             ;TABLE ENTRY SIZE
HDERØ:  MOV     A,M
        ANA     A
        JZ      HDER1           ;USE UNKNOWN ?
        CMP     E
        JZ      HDER1           ;GOOD ERROR CODE
        DAD     B
        JMP     HDERØ           ;TRY NEXT CODE
HDER1:  INX     H               ;POINT TO ERROR MSG
        PUSH    H               ;SAVE POINTER
        LXI     H,ERMSG         ;SEND LEAD MESSAGE
        CALL    OSTR
        POP     H               ;POINTER BACK
        MVI     B,3             ;SEND ERROR MESSAGE
        CALL    PSTR
        LXI     H,HDNAM         ;SEND NAME
        MVI     B,2
        CALL    PSSTR
        LXI     H,TFCN          ;FUNCTION CODE
        MOV     B,M
        INX     H
        MOV     A,M             ;UNIT NO.
        CALL    LBYT            ;LIST UNIT
        CALL    PCOLN           ;COLON
        MOV     A,B
        CALL    LBYT            ;LIST FUNCTION
        LDA     ECODE
        ANI     080H            ;VALID ADDRESS ?
        JZ      NOSEC           ;NO
        CALL    PSPAC           ;SPACE
        LDA     SECTØ
        CALL    LBYT            ;PRINT SECTOR ADDRESS
        LDA     SECT1
        CALL    LBYT
        LDA     SECT2
        CALL    LBYT
NOSEC:  CALL    CRLF
        LDA     HDFLG           ;MORE ERRORS ?
        ANA     A
        JNZ     ELOOP           ;YES
        STC                     ;INDICATE ERROR
        RET                     ;RETURN TO CALLER
;
DATOK:  IN      HSTAT           ;REQ, BSY, I/O MEANS READ DATA
        CPI     015H
        JNZ     DATOK
        IN      RXHD
        RET
;
;       ERROR CODE TABLE
;
```

```
ERTBL:  DB      NSEEK,'ISK'     ;NO SEEK COMPLETE
        DB      WRFLT,'WFT'     ;WRITE FAULT
        DB      DRVNR,'DNR'     ;DRIVE NOT READY
        DB      IVUNT,'IVU'     ;INVALID UNIT
        DB      IVFCN,'IVF'     ;INVALID FUNCTION
        DB      NOPWR,'PWR'     ;NO POWER TO OMTI 20L
        DB      IDECC,'IDF'     ;ECC ERROR IN ID FIELD
        DB      DATER,'DAT'     ;UNCORRECTABLE DATA ERROR
        DB      NIDAD,'IDA'     ;NO ADDRESS MARK IN ID
        DB      NDAAD,'DAA'     ;NO ADDRESS MARK IN DATA
        DB      NOREC,'NRF'     ;NO RECORD FOUND
        DB      SEKER,'SKF'     ;SEEK FAILURE
        DB      CORER,'COR'     ;CORRECTABLE DATA ERROR
        DB      BDTRK,'BTK'     ;BAD TRACK FLAG SET
        DB      ALTER,'ALT'     ;ALTERNATE TRACK READ ERROR
        DB      ILSEC,'ILS'     ;ILLEGAL SECTOR ADDRESS
        DB      VOLOF,'VOF'     ;VOLUME OVERFLOW
        DB      0,    'UNK'     ;ALLOW OTHERS ?

;       READ AND WRITE ROUTINES         ******************************
;
;       SETUP FOR READ/ WRITE
;
REWRIT: IN      HSTAT           ;SEE IF POWERED UP
        ANI     01FH            ;CONTROLL INPUTS
        CPI     01FH            ;ALL HIGH ??
        MVI     E,NOPWR
        JZ      PRERR           ;YES, NO POWER TO CONTROLLER
        XRA     A
        STA     SECT0           ;CLEAR DISK SECTOR ADDRESS
        STA     SECT1
        STA     SECT2
        STA     HALF2           ;SET NO MORE THAN 256 SECTORS
        CALL    IPARMS          ;SET PARAMETERS IF CPU RESET
        LHLD    TBC             ;GET NO. OF BYTES TO TRANSFER
        MOV     A,H
        ORA     L
        RZ                      ;NO BYTES, IGNORE BUT NO ERROR
        CALL    BC2BK           ;CONVERT HL = NO. BLOCKS
        MOV     A,C
        STA     LBLK            ;SAVE LAST BLOCK BYTE COUNT
        SHLD    BKCNT           ;SAVE TOTAL BLOCKS
        MOV     A,L             ;FIGURE BLOCKS TO TRANSFER
        STA     HALF1           ;LOWER 32K BYTES
        LXI     D,00100H        ;DE = 256
        CALL    CMPHD
        JZ      PROCS           ;256 BLOCKS EXACTLY
        JC      PROCS           ;LESS THAN 256 BLOCKS, OK
        CALL    SUBHD           ;SUBTRACT 256 BLOCKS
        MOV     A,L
        STA     HALF2           ;REMAINDER FOR SECOND PASS
PROCS:  LHLD    BKCNT           ;GET TOTAL BLOCKS TO READ
        XCHG
        LHLD    TBLK            ;GET BLOCK TO START
        PUSH    H               ;SAVE START BLOCK
        DAD     D               ;ADD BLOCKS TO READ
        SHLD    BKCNT           ;NEXT BLOCK AFTER TRANSFER
        LXI     D,MBLK          ;MAXIMUM ON DRIVE
        CALL    CMPHD           ;MORE THAN MAXIMUM ?
        JC      PROCX           ;OK !!!
        MVI     E,023H          ;VOLUME OVERFLOW
        POP     H               ;STRIP START BLOCK
PRERR:  POP     H               ;STRIP NORMAL ERROR RETURN
        XRA     A
        STA     HDFLG           ;INDICATE THIS ERROR ONLY
        STA     ECODE           ;INDICATE INVALID SECTOR ADDRESS
        JMP     HDREX           ;SPECIAL ERROR EXIT

;       ADJUST SECTOR FOR RIGHT LOGICAL DEVICE
;
PROCX:  POP     H               ;RECOVER START BLOCK
        LDA     IDRV            ;GET DRIVE NUMBER
        ANA     A               ;CHECK FOR UNIT ZERO
        JZ      PROC2           ;HL = CORRECT SECTOR ADDRESS
PROC0:  DAD     D               ;ADD MBLK TO NEXT UNIT
        JNC     PROC1           ;NOT OVER 64K SECTORS
        PUSH    H
        LXI     H,SECT0         ;MSB'S OF DISK SECTORS
        INR     M
        POP     H
PROC1:  DCR     A
        JNZ     PROC0
PROC2:  MOV     A,H             ;SAVE SECTOR ADDRESS
        STA     SECT1
        MOV     A,L
        STA     SECT2
        LDA     HALF2           ;TWO PASSES REQUIRED ?
        ANA     A
        JNZ     PROC4           ;MORE THAN 32K, DO TWO PASSES
;
;       READ/ WRITE LAST PASS
;
PROC3:  LDA     LBLK            ;LAST BLOCK BYTE COUNT
        CPI     080H
        JZ      PROC4           ;DO A FULL BLOCK
        LDA     HALF1           ;SAVE ONE FOR SPECIAL TRAILING BYTES
        DCR     A
        STA     HALF1
        JZ      DOLAST          ;ONLY ONE BLOCK TO DO
PROC4:  LHLD    TMEM            ;GET MEMORY POINTER
        LDA     HALF1           ;BLOCKS TO READ/ WRITE
        MOV     C,A
        LDA     TFCN            ;GET FUNCTION CODE
        ANA     A
        JZ      RCODE           ;DO A READ
;
;       DO A WRITE WITH AUTOMATIC VERIFY
;
WCODE:  MVI     B,010H          ;ENABLE VERIFY
        MVI     A,00AH          ;WRITE CODE
        CALL    COMND           ;SEND COMMAND
        CALL    WRITX           ;WRITE THE BLOCKS OUT
        RNZ                     ;ERROR, RETURN <NZ>
        JMP     TNEXT           ;SEE IF SECOND PART
;
;       DO A READ WITH ECC AND AUTOMATIC RETRY
;
RCODE:  MVI     B,00H           ;ECC AND RETRY
        MVI     A,008H          ;READ CODE
        CALL    COMND           ;SEND COMMAND
        CALL    READX           ;READ THE BLOCKS
        RNZ                     ;ERROR

;       SETUP FOR SECOND PASS IF NECESSARY
;
TNEXT:  SHLD    TMEM            ;HL => NEXT MEM LOCATION
        LXI     H,HALF2         ;SEE IF ANY MORE BLOCKS
        MOV     A,M
        ANA     A
        JZ      DOLAST          ;WAS LESS THAN 32K
        STA     HALF1           ;SAVE NO. OF REAMINING
        MVI     M,0             ;NO MORE AFTER THIS
        LXI     H,SECT1         ;INCREMENT SECTOR POINTER
        INR     M               ;BY 256 SECTORS
        RNZ                     ;NO CARRY OUT
```

```
        DCX     H               ;POINT TO SECTØ
        INR     M               ;AND INCREMENT
        JMP     PROC3           ;RETURN TO READ/WRITE
;
;       READ/ WRITE TRAILING BYTES IN LAST BLOCK
;
DOLAST: LDA     LBLK            ;WAS LAST BLOCK FULL ?
        CPI     Ø80H
        MVI     E,Ø
        RZ                      ;DONE, NO ERRORS
        MOV     E,A             ;SAVE NO BYTES TO READ/WRITE
        LDA     HALF1           ;GET NO BLOCKS MOVED
        LXI     H,SECT2
        ADD     M               ;ADD TO SECTOR COUNT
        MOV     M,A
        JNC     DOL1
        DCX     H               ;INCREASE NEXT BITS
        INR     M
        JNZ     DOL1            ;DID NOT ROLL PAST ZERO
        DCX     H
        INR     M
DOL1:   LHLD    TMEM            ;GET BACK MEMORY POINTER
        MVI     C,1             ;ONLY ONE SECTOR TO READ/WRITE
        LDA     TFCN            ;READ OR WRITE ?
        ANA     A
        JZ      READL           ;READ
;       WRITE LAST BLOCK IN A FILE WITH TRAILING ZEROS
;
WRITL:  MVI     B,Ø10H          ;ENABLE VERIFY
        MVI     A,ØØAH          ;WRITE CODE
        CALL    COMND           ;SEND COMMAND
WRLØ:   IN      HSTAT           ;REQ, BSY MEANS WRITE DATA
        CPI     Ø14H
        JNZ     WRLØ
        MOV     A,M             ;WRITE DATA
        INX     H
        OUT     TXHD
        DCR     E               ;COUNTER
        JNZ     WRLØ
WRL1:   IN      HSTAT           ;REQ, BSY, C/D, I/O MEANS DONE
        CPI     Ø17H
        JZ      REASØ           ;READ STATUS AND RETURN
WRL2:   IN      HSTAT           ;REQ, BSY MEANS WRITE DATA
        CPI     Ø14H
        JNZ     WRL1
        XRA     A
        OUT     TXHD            ;WRITE A ZERO
        JMP     WRL2
;
;       READ LAST BLOCK OF FILE IGNORING TRAILING BYTES
;
READL:  MVI     B,ØØØH          ;ECC AND RETRY
        MVI     A,ØØ8H          ;READ CODE
        CALL    COMND           ;SEND COMMAND
RELØ:   IN      HSTAT           ;REQ, BSY, I/O MEANS READ
        CPI     Ø15H
        JNZ     RELØ
        IN      RXHD            ;GET BYTE
        MOV     M,A             ;PUT IN MEMORY
        INX     H
        DCR     E               ;COUNTER
        JNZ     RELØ
REL1:   IN      HSTAT           ;REQ, BSY, C/D, I/O MEANS DONE
        CPI     Ø17H
        JZ      REASØ           ;READ STATUS AND RETURN
REL2:   IN      HSTAT           ;REQ, BSY, I/O MEANS READ DATA
        CPI     Ø15H
        JNZ     REL1
        IN      RXHD            ;READ AND IGNORE
        JMP     REL2


;       INITIALIZE CONTROLLER   ****************************************
;
IPARMS: LXI     H,RSFLG         ;HAS CPU RESET OCCURED ?
        XRA     A
        CMP     M
        RNZ                     ;NO
        INR     M               ;RESET DONE
        LXI     B,ØØØH
        MVI     A,ØC2H          ;ASSIGN PARAMETERS CODE
        CALL    COMND
        LXI     H,DTBL          ;PARAMETER LIST
        JMP     WRITX           ;WRITE THE DATA
;
;       DISK PARAMETER TABLE
;
DTBL:   DB      PLSEW           ;PULSE WIDTH
        DB      PLSEP           ;PULSE PERIOD
        DB      SMODE           ;STEP MODE
        DB      NHEAD           ;NUMBER OF R/W HEADS
        DB      NCYL SHR 8      ;MAX. TRACK ADDRESS HIGH
        DB      NCYL AND ØFFH   ;MAX. TRACK ADDRESS LOW
        DB      RWCC            ;REDUCED WRITE CURRENT TRACK
        DB      DRIVE           ;DRIVE TYPE
        DB      NSEC            ;SECTORS/ TRACK
        DB      ØØØH            ;RESERVED
;
;       SEND COMMAND TO CONTROLLER      ********************************
;
;       A     =  COMMAND CODE
;       SECTØ =  MSB'S OF SECTOR ADDRESS
;       SECT1 =          SECTOR ADDRESS
;       SECT2 =  LSB'S OF SECTOR ADDRESS
;       B     =  THE CONTROL FIELD
;       C     =  NUMBER OF SECTORS
;
COMND:  DI                      ;DISABLE INTERRUPTS
        PUSH    PSW             ;SAVE COMMAND
COMNØ:  IN      HSTAT           ;CHECK TO SEE IF BUSY OFF
        ANI     ØØ4H
        JNZ     COMNØ
        DCR     A               ;SELECT CONTROLLER ØFFH
        OUT     SELHD
        POP     PSW
        CALL    REQOK           ;SEND THE COMMAND
        LDA     SECTØ           ;MSB'S OF ADDRESS
        CALL    REQOK
        LDA     SECT1           ;HIGH ADDRESS
        CALL    REQOK
        LDA     SECT2           ;LSB'S OF ADDRESS
        CALL    REQOK
        MOV     A,C             ;NO. SECTORS
        CALL    REQOK
        MOV     A,B             ;CONTROL FIELD
REQOK:  PUSH    PSW             ;SAVE THE BYTE
REQØØ:  IN      HSTAT           ;REQ, BSY, C/D MEANS WRITE COMMAND
        CPI     Ø16H
        JNZ     REQØØ
        POP     PSW
        OUT     TXHD
        RET
```

```
;       READ DATA                       ******************************
;
;       HL => MEMORY LOCATION FOR DATA
;
READX:  IN      HSTAT           ;REQ, BSY, C/D, I/O MEANS DONE
READØ:  CPI     Ø17H
        JZ      REASØ           ;GO GET STATUS AND MESSAGE
READ1:  IN      HSTAT           ;REQ, BSY, I/O MEANS READ DATA
        CPI     Ø15H
        JNZ     READØ
        IN      RXHD
        MOV     M,A
        INX     H
        JMP     READ1
;
;       WRITE DATA                      ******************************
;
;       HL => MEMORY LOCATION WITH DATA
;
WRITX:  IN      HSTAT           ;REQ, BSY, C/D, I/O MEANS DONE
WRITØ:  CPI     Ø17H
        JZ      REASØ           ;GO GET STATUS AND MESSAGE
WRIT1:  IN      HSTAT           ;REQ, BSY MEANS WRITE DATA
        CPI     Ø14H
        JNZ     WRITØ
        MOV     A,M             ;WRITE DATA
        INX     H
        OUT     TXHD
        JMP     WRIT1
;
;       READ STATUS AND MESSAGE         ******************************
;
REASM:  IN      HSTAT           ;REQ, BSY, C/D, I/O MEANS DONE
        CPI     Ø17H
        JNZ     REASM
REASØ:  IN      RXHD            ;GET THE STATUS BYTE
        MOV     E,A             ;SAVE IN E
REAS1:  IN      HSTAT           ;REQ, MSG, BSY, C/D I/O MEANS READ
        CPI     Ø1FH            ; THE MESSAGE BYTE
        JNZ     REAS1
        IN      RXHD            ;GET THE MESSAGE BYTE AND IGNORE
        MOV     A,E             ;TEST STATUS
        ANA     A
        EI                      ;ENABLE INTERRUPTS
        RET
```

'Tom Teaser' (Solution)

Tom Napier

Here was the puzzle; without using an MVI instruction or making any preconditions, load 06H into the A register using only two bytes.

This odd puzzle arose from my observation that though I have always used SUB A to clear the A register, nearly all the published programs I have seen use XRA A. At first glance the effect of the two instructions is exactly the same, but there is a subtle difference. Both instructions clear the Carry and Sign flags, and set the Parity and Zero flags. The difference lies in the Auxiliary Carry flag. XRA A clears it, but SUB A sets it. This is not a dramatic difference unless a DAA instruction follows. The sequence XRA A : DAA sets A = 0 and sets the zero flag. The sequence SUB A : DAA sets A = 6 and clears the zero flag. The prior content of the A register is immaterial.

This little exercise shows that the 8080 instruction set can still have some surprises. This particular one has been hidden from me in ten years of programming experience.

ASTRO

Wallace Rust
523 Britton Road
Greece, NY 14616

The ASTRO program, presented here, calculates the celestial positions of the sun, moon, and all eight planets. The sun and planets are located to within a few minutes of arc, and the moon within a degree or so. Writing such a computer program presents a number of interesting challenges, and ASTRO meets these with surprising results.

Most of the necessary constants and equations have been obtained from the excellent book by Peter Duffett-Smith, "Practical Astronomy with Your Calculator." (See full reference in program listing, lines 130-150.) This book presumes a scientific calculator or a computer with double precision arithmetic, and so a principle challenge has been to maintain accuracy while using Compucolor's single-precision BASIC.

Julian date is generally used by astronomers to refer to the time of events because it neatly avoids the peculiarities of the civil calendar. The Julian date is the number of days that have elapsed since January 1, 4713 B.C. If we want a precision of a few seconds per day, we will have Julian dates such as 2445538,1234 to work with. Therefore, in ASTRO we use an epoch (reference) of A.D. Jan 0,0 1980 instead, and count days forward or backward from that. Another way of preserving precision is to work in radians as much as possible, and convert radians to time or degrees only for the output.

Let us take a walk through the features of the program. At line 310 we dimension arrays for holding the orbital elements of the planets (PP), sun (SP), moon (MP), names (R$), and the calculated results (R). Lines 340-397 explain the subscripts.

Constants are defined in lines 400-450. They are entered to eight-digit precision because the Compucolor works to that precision internally. In lines 500-950 we load the arrays.

At lines 2000-2310 we input the observing site. You can add your own location to the program here, or you can type it in when the program runs by choosing "1" at the line 2200 prompt, which causes a branch to the input routine at lines 2400-2500. The date and time of observation are entered at lines 2505-2630.

ASTRO begins its calculations at line 2640, and they take about thirty seconds. Great care must be taken in calculating arctangents. The routine at lines 4100-4150 was written to do this correctly for all quadrants. You may have to modify the routine at lines 5000-5030 to suit your own printer; the routine in the listing is for the Radio Shack DMP-110.

Sample Printout of "ASTRO"

```
SKY POSITIONS FOR 1984 WED JUN  6

LOCAL STANDARD TIME:  22 HOURS, 0 MINUTES

FOR LAT.  40.72    LONG. WEST 74.02    ELEV.  10 METRES    (New York City)

DAYS SINCE JAN 0 THIS YEAR:  158

TIMES IN HOURS:  LMT = 22          GMT = 3
                 LST = 15.0463     GST = 19.9809
```

BODY	ECLIPTIC LONG. (DEG)	ECLIPTIC LAT. (DEG)	RIGHT ASCENSION* (HOURS)	DECLI-NATION* (DEG)
SUN	76.3728	0	5.01324	22.7443
MOON	169.316	5.00975	11.4359	8.25356
MERCURY	59.1972	-1.56545	3.82274	18.4519
VENUS	73.8518	-.150447	4.8336	22.3159
MARS	223.043	-1.31751	14.6783	-17.0109
JUPITER	280.602	.117435	18.7681	-22.9014
SATURN	221.261	2.53943	14.6425	-12.7943
URANUS	251.955	.0240665	16.6969	-22.2016
NEPTUNE	269.816	1.23014	17.9867	-22.2116
PLUTO	211.28	17.1477	14.3354	4.19362

BODY	HOUR ANGLE* (DEG WEST)	AZIMUTH* (DEG)	ALTITUDE* (DEG)	PHASE (0 TO 1)
SUN	150.496	330.918	-20.8615	1
MOON	54.1552	251.446	32.2	.525819
MERCURY	168.353	347.245	-29.8454	.764606
VENUS	153.19	333.211	-22.2139	.999087
MARS	5.51894	186.229	32.0417	.964916
JUPITER	304.173	129.689	7.94875	.998402
SATURN	6.05631	187.325	36.1923	.99913
URANUS	335.24	155.087	22.9982	.999996
NEPTUNE	315.893	138.181	14.9048	.999985
PLUTO	10.6632	197.538	52.2351	.999842

```
* MOON POSITIONS ARE CORRECTED FOR PARALLAX.

THE ABOVE ANGLES IN RADIANS:
```

1.33296	0	1.31246	.396963	2.62664	5.77561	-.364102
2.95312	.0874366	2.99392	.144052	.945186	4.38856	.561996
1.03319	-.0273222	1.00079	.322047	2.93831	6.06057	-.5209
1.28896	-2.6258E-03	1.26543	.389486	2.67367	5.81563	-.387705
3.89284	-.022995	3.84278	-.296897	.0963237	3.2503	.559233
4.89744	2.04963E-03	4.91348	-.399704	5.30881	2.26349	.138732
3.86174	.0443214	3.8334	-.223302	.105703	3.26943	.631675
4.39745	4.2004E-04	4.37124	-.387491	5.85105	2.70678	.401395
4.70917	.02147	4.70892	-.387666	5.51337	2.4117	.260138
3.68753	.299283	3.753	.0731926	.186109	3.44768	.911674

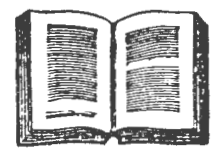
Calculation routines are headed by remarks and by references in brackets to the Duffett-Smith book. In the routines at lines 16500-16960 we take into account the parallax caused by the nearness of the moon. (The only factor not taken into account in this program is the precession of the earth.) At lines 17140-17300 we calculate look-angles for the chosen observing site.

At line 20000 we output the results to the screen as three pages of tables. At the user's option (line 20760), results can also be directed to the printer. The routine at line 60000 permits a printed listing by typing "GOTO 60000" in 'immediate' mode.

The sample output for a particular time and site will allow you to debug your program. If any reader wants a listing of the 107 variables and their meaning, send me a 3 by 9 inch SASE. Ten dollars (US) will purchase both the list and a diskette in CCII v6.78 format. This program uses about 12K of RAM.

```
100 REM  "ASTRO", CALC & PRINT ASTRONOMICAL POSITIONS.
105 REM  WALLACE R. RUST, 523 BRITTON ROAD, GREECE, NY 14616.
110 REM  VERSION: 9 JAN 1985
125 REM
130 REM  REF: "PRACTICAL ASTRONOMY WITH YOUR CALCULATOR",
140 REM  2ND EDITION, BY PETER DUFFETT-SMITH,
145 REM  CAMBRIDGE UNIVERSITY PRESS, (C) 1981.
150 REM  SECTION REFERENCES IN [ ].
160 REM
170 REM  LAMBDA, BETA = ECLIPTIC LONGITUDE, LATITUDE
171 REM  ALPHA, DELTA = RIGHT ASCENSION, DECLINATION
172 REM  EPOCH JAN 0.0 1980
173 REM
180 REM  CHANGE LINES 5000-5110 TO SUIT YOUR OWN PRINTER!
185 REM  ENTER YOUR FAVORITE STATIONS AT LINES 2000-2390!
190 REM
200 REM  !--- TITLE PAGE
202 CLEAR 200
205 PLOT 6,3,12,14
210 PRINT TAB( 26);:PLOT 6,38:PRINT " > ASTRO < "
220 PLOT 6,3,15:PRINT
230 PRINT "THIS PROGRAM CALCULATES POSITIONS OF THE SUN, MOON, AND PLANETS."
240 PLOT 6,5:PRINT "TURN ON THE PRINTER!"
300 REM  !--- HOUSEKEEPING
310 DIM PP(8,6),SP(5),MP(7),R$(9),R(9,7)
330 REM
340 REM  PP(I,J):
350 REM    I=0 EARTH     J=0 PERIOD
352 REM      1 MERCURY     1 LONG. AT EPOCH
354 REM      2 VENUS       2 LONG. AT PERIH.
356 REM      3 MARS        3 ECCENTRICITY
358 REM      4 JUPITER     4 SEMI-MAJ AXIS (AU)
360 REM      5 SATURN      5 INCLINATION
362 REM      6 URANUS      6 LONG. ASCEN. NODE
364 REM      7 NEPTUNE
366 REM      8 PLUTO
368 REM
380 REM  R(I,J):
382 REM    I=0 SUN       J=0 LAMBDA
384 REM      1 MOON        1 BETA
386 REM      2 MERCURY     2 ALPHA
388 REM      3 VENUS       3 DELTA
390 REM      4 MARS        4 HOUR ANGLE
392 REM      5 JUPITER     5 AZIMUTH
394 REM      6 SATURN      6 ALTITUDE
395 REM      7 URANUS      7 PHASE
396 REM      8 NEPTUNE
397 REM      9 PLUTO
398 REM
400 PI= 3.1415926:P2= 6.2831853:DR= 57.29578:HR= 3.8197186
410 KA= 360:KB= 365.2422:KC= 13.1763966:KD= .1114041
420 KE= .0529539:KF= 1.2739:KG= .1858:KH= .37
430 KI= 6.2886:KJ= .214:KK= .6583:KL= .16
440 KM= .001:KN= .065709822:KQ= 1.002738
450 KR= .996647:KS= 6378140:KT= 6378.16:REM  [#35,36]
500 REM  !--- SUN ELEMENTS [P. 82;44]
510 RESTORE 520
520 DATA 278.83354,282.596403,.016718,1.495985E8,.533128
521 DATA 23.441884
```

```
520 DATA 278.83354,282.596403,.016718,1.495985E8,.533128
521 DATA 23.441884
530 FOR J= 0TO 5:READ SP(J):NEXT
620 DATA 64.975464,349.383063,151.950429,5.145396,.0549,.5181,384401,.9507
630 FOR J= 0TO 7:READ MP(J):NEXT
700 REM  X--- EARTH, MERC, VEN, MARS-PLUTO ELEMENTS [P. 100]
705 RESTORE 710
710 DATA 1.00004,98.83354,102.5964,.016718,1,0,0
720 DATA .24085,231.2973,77.144213,.2056306,.3870986,7.0043579,48.094173
730 DATA .61521,355.73352,131.28958,.0067826,.7233316,3.394435,76.499752
750 DATA 1.88089,126.30783,335.690816,.0933865,1.5236883,1.8498011,49.4032
760 DATA 11.86224,146.966365,14.009549,.0484658,5.202561,1.3041819,100.25202
770 DATA 29.45771,165.32224,92.665397,.0556155,9.554747,2.4893741,113.48883
775 DATA 84.01247,228.07085,172.73633,.0463232,19.21814,.7729895,73.876864
780 DATA 164.79558,260.3579,47.867215,.0090021,30.10957,1.7716017,131.56065
785 DATA 250.9,209.439,222.972,.25387,39.78459,17.137,109.941
800 FOR I= 0TO 8:FOR J= 0TO 6:READ PP(I,J):NEXT :NEXT
900 REM  X--- NAMES
910 RESTORE 920
920 DATA "SUN","MOON","MERCURY","VENUS","MARS","JUPITER"
921 DATA "SATURN","URANUS","NEPTUNE","PLUTO"
930 FOR J= 0TO 9:READ R$(J):NEXT
940 W$= "SUNMONTUEWEDTHUFRISAT"
950 M$= "JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC"
2000 REM  X--- INPUT SITE
2010 PLOT 6,2:PRINT :PRINT "AVAILABLE SITES:":PLOT 6,6
2020 PRINT " 1. -USER'S CHOICE-"
2030 PRINT " 2. GREENWICH (LONDON)"
2040 PRINT " 3. NEW YORK, NY"
2050 PRINT " 4. MIAMI, FL"
2060 PRINT " 5. ERIE, PA"
2070 PRINT " 6. GREECE, NY (RUST'S HOME)"
2080 PRINT " 7. SAN DIEGO, CA"
2090 REM  ADD  #8 HERE
2100 REM  ADD  #9 HERE
2110 REM  ADD #10 HERE
2120 REM
2200 PRINT :PLOT 6,1:INPUT "ENTER YOUR CHOICE (1 TO 6): ";Q
2210 IF Q< 1OR Q> 10THEN 2200
2220 ON QGOTO 2400,2302,2303,2304,2305,2306,2307,2308,2309,2310
2230 REM  SY=LATITUDE; SX=LONGITUDE; SE=ELEV IN METRES; TZ=TIME ZONES WESTWARD
2302 SY= 51.5:SX= 0:SE= 10:TZ= 0:GOTO 2500
2303 SY= 40.72:SX= 74.02:SE= 10:TZ= 5:GOTO 2500
2304 SY= 25.77:SX= 80.2:SE= 5:TZ= 5:GOTO 2500
2305 SY= 42.13:SX= 80.06:SE= 200:TZ= 5:GOTO 2500
2306 SY= 43.23625:SX= 77.63894:SE= 113:TZ= 5:GOTO 2500
2307 SY= 32.76:SX= 117.22:SE= 10:TZ= 8:GOTO 2500
2308 REM  ADD  #8 HERE
2309 REM  ADD  #9 HERE
2310 REM  ADD #10 HERE
2390 REM
2400 PRINT :PLOT 6,3
2410 INPUT "ENTER LATITUDE IN DEGREES: ";SY
2420 IF SY< - 90OR SY> 90THEN 2410
2430 INPUT "ENTER LONGITUDE IN DEGREES WEST OF GREENWICH: ";SX
2440 IF SX< 0OR SX> 360THEN 2430
2450 INPUT "ENTER ELEVATION IN METRES ABOVE SEA LEVEL: ";SE
2470 INPUT "ENTER TIME ZONE NUMBER (0 TO 23): ";TZ
2480 TZ= INT (TZ):IF TZ< 0OR TZ> 23THEN 2470
2500 LA= SY/ DR
2505 REM  X--- INPUT DATE & TIME
2510 PLOT 6,5:PRINT :PRINT "ENTER DATE OF INTEREST:":PLOT 6,3
2520 INPUT "YEAR (1583 TO 2400):  ";TY
2530 TY= INT (TY):IF TY< 1583OR TY> 2400THEN 2520
2540 INPUT "MONTH (1 TO 12):      ";TM
2550 TM= INT (TM):IF TM< 1OR TM> 12THEN 2540
2560 INPUT "DATE (1 TO 31):       ";TD
2570 TD= INT (TD):IF TD< 1OR TD> 31THEN 2560
2580 PLOT 6,5:PRINT "ENTER LOCAL STANDARD TIME OF INTEREST:":PLOT 6,3
2590 INPUT "HOUR (0 TO 23):       ";TH
2600 TH= INT (TH):IF TH< 0OR TH> 23THEN 2590
2610 INPUT "MINUTE (0 TO 60.0):   ";TN
2620 IF TN< 0OR TN> 60THEN 2610
2630 LM= TH+ TN/ 60
2640 PRINT "XWAIT..."
```



```
2700 REM  ¥--- CALC DAYS SINCE JAN 0.0 1980
2701 REM  [COMPUCOLOR 'BIORHYTHMS']
2710 D1= TH/ 24+ TN/ 1440
2720 M9= (- 1)* INT (((14- TM)/ 12)+ KM)
2730 J1= TD- 2447095+ INT ((1461* (TY+ 4800+ M9)/ 4)+ KM)
2740 J2= J1+ INT ((367* (TM- 2- 12* M9)/ 12)+ KM)
2750 J1= J2- INT ((3* (TY+ 4900+ M9)/ 400)+ KM)
2760 WD= J1- 7* INT ((J1/ 7)+ KM)+ 1:WD= INT (WD+ KM)
2770 DE= J1- 29219+ D1
2800 REM  ¥--- CALC WHOLE DAYS SINCE JAN 0.0 THIS YEAR
2801 REM  [W. RUST]
2810 D$= "000031059090120151181212243273304334"
2820 DJ= VAL (MID$ (D$,3* TM- 2,3))+ TD
2830 LY= 0:IF TY/ 4= INT (TY/ 4)THEN LY= 1
2840 IF TY/ 400= INT (TY/ 400)THEN LY= 0
2850 IF TM> 2THEN DJ= DJ+ LY
3000 REM  ¥--- LMT TO GMT [#9]
3010 GM= LM+ TZ:IF GM> = 24THEN GM= GM- 24
3030 REM  ¥--- CALC GST AT JAN 0.0 THIS YEAR [#4,12]
3035 Y= TY- 1:A= INT (Y/ 100):B= 2- A+ INT (A/ 4)
3040 C= INT (365.25* Y):S= 8+ C- 693597.5:T= S/ 36525
3045 R= 6.6460656+ 2400.051262* T+ .00002581* T* T
3047 B= 24* TY- 45576- R
3050 REM  ¥--- GMT TO GST [#12]
3070 T0= KN* DJ- 8
3080 X= T0+ KQ* GM:GOSUB 4050:GS= X
3100 REM  ¥--- GST TO LST [#14]
3110 LS= GS- SX/ 15:IF LS< 0THEN LS= LS+ 24
3999 GOTO 10000
4000 REM  ¥--- SUBR X=MOD(X,360)
4010 IF X< 0THEN X= X+ KA:GOTO 4010
4020 IF X> = KATHEN X= X- KA:GOTO 4020
4030 RETURN
4050 REM  ¥--- SUBR X=MOD(X,24)
4060 IF X< 0THEN X= X+ 24:GOTO 4060
4070 IF X> = 24THEN X= X- 24:GOTO 4070
4080 RETURN
4100 REM  ¥--- SUBR A=ATN(Y/X) RADIANS
4110 IF X< > 0THEN 4140
4120 A= PI/ 2:IF Y< 0THEN A= 3* PI/ 2
4130 RETURN
4140 A= ATN (Y/ X):IF X< 0THEN A= A+ PI:RETURN
4150 IF Y< 0THEN A= A+ P2
4160 RETURN
4200 REM  ¥--- SUBR X=MOD(X,2*PI)
4210 IF X< 0THEN X= X+ P2:GOTO 4210
4220 IF X> = P2THEN X= X- P2:GOTO 4220
4230 RETURN
4300 REM  ¥--- SUBR OUTPUT PAGING
4310 IF Q$= "P"THEN RETURN
4320 INPUT "¥PRESS ¥<RETURN> ¥FOR MORE...";A$:PLOT 6,3:RETURN
5000 REM  ¥--- SUBR SELECT PRINTER
5010 POKE 33289,80:PLOT 15,27,18,4,27,13:OUT 8,4
5020 PLOT 30,19,27,28,27,19:REM  PRINTER FONT
5030 RETURN
5100 REM  ¥--- SUBR SELECT CRT
5110 OUT 8,255:POKE 33265,0:POKE 33289,64:RETURN
9999 REM
10000 REM  ¥--- SUN CALC [#42]
10010 X= KA* DE/ KB:GOSUB 4000:N= X
10020 X= N+ SP(0)- SP(1):GOSUB 4000:SM= X/ DR
10030 E= KA* SP(2)* SIN (SM)/ PI
10040 X= N+ E+ SP(0):GOSUB 4000:SL= X/ DR
10050 R(0,0)= SL:R(0,1)= 0:REM  SUN LAMBDA, BETA (RAD)
10060 R(0,7)= 1
11000 REM  ¥--- MOON CALC [#61]
11010 X= KC* DE+ MP(0):GOSUB 4000:L= X
11020 X= L- KD* DE- MP(1):GOSUB 4000:MM= X/ DR
11030 X= MP(2)- KE* DE:GOSUB 4000:N= X
11040 C= L/ DR- SL:EV= KF* SIN (2* C- MM)
11050 AE= KG* SIN (SM):A3= KH* SIN (SM)
11060 X= MM+ (EV- AE- A3)/ DR:MM= X:REM  CORRECTED ANOMALY
11070 EC= KI* SIN (MM)
11080 A4= KJ* SIN (2* MM)
11090 L= L+ EV+ EC- AE+ A4
```

```
11095 D= L/ DR- SL
11100 V= KK* SIN (2* D)
11110 L= L+ V
11120 N= N- KL* SIN (SM)
11125 J1= (L- N)/ DR
11130 Y= SIN (J1)* COS (MP(3)/ DR)
11140 X= COS (J1)
11150 GOSUB 4100:X= A+ N/ DR:GOSUB 4200:R(1,0)= X:REM  MOON LAMBDA (RAD)
11160 J2= SIN (J1)* SIN (MP(3)/ DR)
11170 R(1,1)= ATN (J2/ SQR (1- J2* J2)):REM  MOON BETA (RAD)
11180 REM  [#63]
11190 R(1,7)= (1- COS (D))/ 2:REM  MOON PHASE
12000 REM  %--- EARTH CALC [#50]
12010 X= KA* DE/ KB/ PP(0,0):GOSUB 4000:NE= X
12020 ME= NE+ PP(0,1)- PP(0,2)
12030 X= NE+ KA* PP(0,3)* SIN (ME/ DR)/ PI+ PP(0,1):GOSUB 4000:LE= X
12040 VE= LE- PP(0,2)
12050 RE= (1- PP(0,3)* PP(0,3))/ (1+ PP(0,3)* COS (VE/ DR))
13000 REM  %--- BEGIN MERCURY THRU PLUTO CALC [#50]
13005 FOR J= 1TO 8
13010 X= KA* DE/ KB/ PP(J,0):GOSUB 4000:NP= X
13020 MP= NP+ PP(J,1)- PP(J,2)
13030 X= NP+ KA* PP(J,3)* SIN (MP/ DR)/ PI+ PP(J,1):GOSUB 4000:LP= X
13040 VP= LP- PP(J,2)
13050 RP= PP(J,4)* (1- PP(J,3)* PP(J,3))/ (1+ PP(J,3)* COS (VP/ DR))
13060 REM
13065 J9= (LP- PP(J,6))/ DR
13070 J1= SIN (J9)* SIN (PP(J,5)/ DR)
13080 PS= ATN (J1/ SQR (1- J1* J1))
13090 Y= SIN (J9)* COS (PP(J,5)/ DR)
13100 X= COS (J9)
13110 GOSUB 4100:LP= A* DR+ PP(J,6)
13120 RP= RP* COS (PS)
13200 IF J> 2THEN 15000
14000 REM  %--- CONTINUE MERC & VENUS CALC [#50]
14010 J9= (LE- LP)/ DR
14020 Y= RP* SIN (J9):X= RE- RP* COS (J9)
14030 GOSUB 4100
14040 X= 180+ LE+ A* DR:GOSUB 4000:LD= X/ DR
14041 R(J+ 1,0)= LD:REM  PLANET LAMBDA (RAD)
14045 D= LD- LP/ DR
14050 Y= RP* TAN (PS)* SIN (D)
14060 X= RE* SIN (- J9)
14070 A= ATN (Y/ X):R(J+ 1,1)= A:REM  PLANET BETA (RAD)
14080 REM  [#54]
14090 R(J+ 1,7)= (1+ COS (D))/ 2:REM  PLANET PHASE
14100 NEXT J
15000 REM  %--- CONTINUE MARS-TO-PLUTO CALC [#50]
15010 J9= (LP- LE)/ DR
15020 Y= RE* SIN (J9):X= RP- RE* COS (J9)
15030 GOSUB 4100:X= A* DR+ LP:GOSUB 4000
15035 LD= X/ DR:R(J+ 1,0)= LD:REM  PLANET LAMBDA (RAD)
15040 D= LD- LP/ DR
15050 Y= RP* TAN (PS)* SIN (D):X= RE* SIN (J9)
15060 A= ATN (Y/ X):R(J+ 1,1)= A:REM  PLANET BETA (RAD)
15065 REM  [#54]
15070 R(J+ 1,7)= (1+ COS (D))/ 2:REM  PLANET PHASE
15080 NEXT J
15100 REM  WE NOW HAVE LAMBDA & BETA FOR ALL BODIES
16000 REM  %--- CONVERT (LAMBDA,BETA) TO (ALPHA,DELTA) [#27]
16090 EP= SP(5)/ DR
16100 FOR I= 0TO 9
16110 J1= SIN (R(I,1))* COS (EP)+ COS (R(I,1))* SIN (EP)* SIN (R(I,0))
16120 R(I,3)= ATN (J1/ SQR (1- J1* J1)):REM  DELTA (RAD)
16125 REM
16130 Y=.SIN (R(I,0))* COS (EP)- TAN (R(I,1))* SIN (EP)
16140 X= COS (R(I,0))
16150 GOSUB 4100:R(I,2)= A:REM  ALPHA (RAD)
16160 NEXT I
16300 REM  %--- CALC HOUR ANGLE [#24]
16320 FOR I= 0TO 9
16330 X= LS- R(I,2)* HR:GOSUB 4050:HA= X/ HR
16340 R(I,4)= HA:REM  HOUR ANGLE (RAD)
16350 NEXT I
16500 REM  %--- MOON PARALLAX [#65]
```

```
16510 Y= 1- MP(4)* MP(4)
16511 X= 1+ MP(4)* COS (MM+ EC/ DR):PA= Y/ X
16520 PK= PA* MP(6):PH= MP(7)/ PA/ DR
16800 REM  ¥--- MOON PARALLAX [#35]
16810 U= ATN (KR* TAN (LA)):H= SZ/ KS
16820 PB= KR* SIN (U)+ H* SIN (LA):PC= COS (U)+ H* COS (LA)
16900 REM  ¥--- MOON PARALLAX [#36]
16910 HA= R(1,4):R= 1/ SIN (PH)
16920 Y= PC* SIN (HA):X= R* COS (R(1,3))- PC* COS (HA)
16930 D= ATN (Y/ X):H1= HA+ D
16935 R(1,4)= H1:REM  CORR. MOON HA
16940 R(1,2)= R(1,2)- D:REM  CORR. MOON ALPHA
16950 Y= R* SIN (R(1,3))- PB:X= R* COS (R(1,3))* COS (HA)- PC
16960 R(1,3)= ATN (COS (H1)* Y/ X):REM  CORR. MOON DELTA
17140 REM  ¥--- CONVERT (ALPHA,DELTA) TO (AZ,ALT) [#25]
17150 FOR I= 0TO 9
17155 HA= R(I,4)
17160 J1= SIN (R(I,3))* SIN (LA)+ COS (R(I,3))* COS (LA)* COS (HA)
17170 AL= ATN (J1/ SQR (1- J1* J1)):R(I,6)= AL:REM  ALTITUDE (RAD)
17180 REM
17190 Y= SIN (R(I,3))- SIN (LA)* J1
17200 X= COS (LA)* COS (AL):J2= Y/ X
17210 A= ATN (SQR (1- J2* J2)/ J2)
17220 IF J2< 0THEN A= A+ PI
17221 IF HA< PITHEN A= P2- A
17230 R(I,5)= A:REM  AZIMUTH (RAD)
17300 NEXT I
20000 REM  ¥--- PRINT RESULTS
20010 PLOT 12
20020 Q$= "S":REM  RESULTS TO SCREEN FIRST
20030 IF Q$= "P"THEN GOSUB 5000
20080 W1$= MID$ (W$,WD* 3- 2,3)
20090 M1$= MID$ (M$,TM* 3- 2,3)
20100 PRINT "SKY POSITIONS FOR";TY;" ";W1$;" ";M1$;" ";TD
20110 PRINT
20120 PRINT "LOCAL STANDARD TIME: ";TH;" HOURS,";TN;" MINUTES"
20130 PRINT
20140 PRINT "FOR LAT. ";SY;"    LONG. WEST";SX;"    ELEV. ";SE;" METRES"
20150 PRINT
20160 PRINT "DAYS SINCE JAN 0 THIS YEAR: ";DJ
20170 PRINT
20180 PRINT "TIMES IN HOURS: LMT =";LM;TAB( 36);"GMT =";GM
20190 PRINT "                LST =";LS;TAB( 36);"GST =";GS
20200 PRINT :PRINT
20210 PRINT "           ECLIPTIC    ECLIPTIC    RIGHT       DECLI-"
20220 PRINT "           LONG.       LAT.        ASCENSION*  NATION*"
20230 PRINT "BODY       (DEG)       (DEG)       (HOURS)     (DEG)"
20240 PRINT
20300 FOR I= 0TO 9
20310 PRINT R$(I);TAB( 12);R(I,0)* DR;TAB( 24);R(I,1)* DR;
20320 PRINT TAB( 36);R(I,2)* HR;TAB( 48);R(I,3)* DR
20350 NEXT I
20500 PRINT :PRINT
20510 GOSUB 4300
20520 PRINT "           HOUR ANGLE* AZIMUTH*    ALTITUDE*   PHASE"
20530 PRINT "BODY       (DEG WEST)  (DEG)       (DEG)       (0 TO 1)"
20540 PRINT
20600 FOR I= 0TO 9
20610 PRINT R$(I);TAB( 12);R(I,4)* DR;TAB( 24);R(I,5)* DR;
20620 PRINT TAB( 36);R(I,6)* DR;TAB( 48);R(I,7)
20650 NEXT I
20660 PRINT :PRINT "* MOON POSITIONS ARE CORRECTED FOR PARALLAX.":PRINT
20670 GOSUB 4300
20700 PRINT "THE ABOVE ANGLES IN RADIANS:":PRINT
20710 FOR I= 0TO 9
20720 FOR J= 0TO 6
20730 PRINT TAB( J* 11.5);R(I,J);
20740 NEXT J:PRINT :NEXT I:PRINT :PRINT
20750 IF Q$= "P"THEN 30000
20760 INPUT "¥DO YOU WANT ABOVE RESULTS SENT TO PRINTER? (¥Y ¥OR ¥N¥) ";A$
20770 IF A$= "Y"THEN Q$= "P":GOTO 20030
30000 GOSUB 5100:GOTO 2000
30010 END
60000 REM  ¥--- LIST ON PRINTER
60010 GOSUB 5000:LIST :PRINT :GOSUB 5100:END
```

Wator

A spectacular assembly language presentation.

Tom Napier
12 Birch Street
Monsey, NY 10952

This program is an assembly language implementation of a program suggested by A. K. Dewdney in the December, 1984 issue of "The Scientific American" (pp 14). The toroidal planet, WA-TOR, is populated by sharks and fish. By establishing their initial numbers, their breeding times, and the nourishment required by the sharks, an ecological universe, having its own special rhythm, is created. The computer program plots before you, on the CRT, the evolution of this universe in terms of the populations of its two species of inhabitants.

The assembly language code is commented sparingly and will not take long to type. Assembly has been ORGed at 8200H. The graphics character set is required. At run time, the following screen display will be presented, and you must set up the initial parameters for the ecology of WA-TOR. The numbers in parentheses are a suggested starting point.

```
WELCOME TO THE WATERY WORLD OF WA-TOR.

YOU MAY SELECT ITS POPULATION.

ENTER NUMBER OF FISHES ? (500)
ENTER NUMBER OF SHARKS ? (20)
ENTER FISH BREEDING TIME (1-7) ? (3)
ENTER SHARK BREEDING TIME (1-7) ? (6)
ENTER TIME TO STARVE A SHARK (1-7) ? (4)
```

Now the display begins. Upper left numbers show the number of fish, followed by the number of sharks. Fish are in cyan, sharks in red, all on a blue background. The screen redraws itself about once each second.

The suggested starting parameters appear at first to quickly diminish the fish population, but, within the first minute, you will observe that Nature has a way of striking a balance. In order to appreciate the operation and theory behind the concept demonstrated in this program, you should read the article by Mr. Dewdney, who presents a lucid and informative description of its performance. You will also share my admiration for the rendition that Tom Napier has presented. [ED] □

```
;       SHARKS AND FISHES PROGRAM  22/12/84

;       COPYRIGHT (C) 1984   T. M. NAPIER

;       AN IMPLEMENTATION ON THE COMPUCOLOR II OF THE
;       PROGRAM 'WA-TOR' DESCRIBED BY A. K. DEWDNEY
;       IN THE DECEMBER 1984 SCIENTIFIC AMERICAN.

;       DISPLAY SIZE 64 BY 32

;       2K STORAGE, ONE BYTE PER LOCATION
;       BYTE CONTAINS  M F SSS AAA

;               M = 1 IF OBJECT ALREADY MOVED
;               F = 0 IF FISH, 1 IF SHARK
;               SSS = TIME SINCE SHARK LAST ATE
;               AAA = TIME SINCE LAST BREEDING

;               IF FISH, SSS IS NOT NEEDED SO SSS := 100
;               THUS A ZERO BYTE EQUALS AN EMPTY SPACE

;               INITIAL FISH = 2XH, X = RANDOM AGE
;               INITIAL SHARK = 4XH

ORG     8200H

WTOR:   LXI     SP,STK
        MVI     A,195
        STA     UINP
        LXI     H,INP
        SHLD    UINP+1
        MVI     A,31
        STA     KBFL
        CALL    STR
        DB      6,2,15,12,10,10,0
        CALL    STR
        DB      'WELCOME TO THE WATERY WORLD OF WA-TOR.',13,10,10,0
        CALL    STR
        DB      'YOU MAY SELECT ITS POPULATION.',13,10,10,0
        CALL    STR
        DB      'ENTER NUMBER OF FISHES ',0
        CALL    NUMB
        SHLD    NOF
        CALL    STR
        DB      'ENTER NUMBER OF SHARKS ',0
        CALL    NUMB
        SHLD    NOS
        CALL    STR
        DB      'ENTER FISH BREEDING TIME (1-7) ',0
        CALL    NUMB
        STA     FBA
        CALL    STR
        DB      'ENTER SHARK BREEDING TIME (1-7) ',0
        CALL    NUMB
        STA     SBA
        CALL    STR
        DB      'ENTER TIME TO STARVE A SHARK (1-7) ',0
        CALL    NUMB
        ADD     A
        ADD     A
        ADD     A               ;MULTIPLY BY 8
        STA     SSA
        CALL    STR
        DB      6,34,12,0
        SUB     A
        STA     KBFL
        CALL    SETU
WTR1:   CALL    DISP
        CALL    SCOR
        CALL    FSCA
        CALL    SSCA
        LDA     RDY
        CPI     50H
        JNZ     WTR1
        LXI     SP,8042H
        LXI     H,3BH
        PUSH    H
        LXI     H,KBFL
        JMP     ESCD
```

```
INP:    LXI     H,IBUF
        MOV     A,E
        CPI     13
        JZ      ICR
        CPI     26
        JZ      IBS
        MOV     A,M
        CPI     4
        RNC
        INR     A
        MOV     M,A
        ADD     L
        MOV     L,A
        MOV     M,E
        MOV     A,E
        CALL    TTO
        RET

ICR:    MOV     A,M
        MVI     M,5
        INR     A
        ADD     L
        MOV     L,A
        MOV     M,E
        CALL    CRLF
        RET

IBS:    MOV     A,M
        DCR     A
        RM
        MOV     M,A
        MVI     A,26
        CALL    TTO
        MVI     A,' '
        CALL    TTO
        MVI     A,26
        CALL    TTO
        RET

NUMB:   LXI     H,IBUF
        MVI     M,0
        XCHG
        LXI     H,0
NUM1:   LDAX    D
        CPI     5
        JNZ     NUM1            ;WAIT TILL NUMBER INPUT
NUM2:   INX     D
        LDAX    D
        CPI     13
        JZ      NUM3
        MOV     B,H
        MOV     C,L
        DAD     H
        DAD     H
        DAD     B
        DAD     H               ;HL = 10*HL
        SUI     30H             ;DIGIT TO NUMBER
        CALL    ADHL
        JMP     NUM2

NUM3:   MOV     A,L
        ANI     7
        RET

SETU:   LDA     TSEC
        MOV     L,A
        LDA     TMIN
        MOV     H,A
        SHLD    RANN            ;RANDOMIZE
        MVI     A,19
        STA     RCYC            ;RANDOMIZER CYCLE COUNT
        CALL    CLR             ;CLEAR SEA
        LHLD    NOF
        MOV     B,H
        MOV     C,L
        MVI     D,FBAS          ;BARE FISH
        LDA     FBA
        MOV     E,A
        CALL    FILL            ;LOAD SEA WITH FISHES
        LHLD    NOS
        MOV     B,H
        MOV     C,L
        MVI     D,SBAS          ;BARE SHARK
        LDA     SBA
        MOV     E,A
        CALL    FILL
        MVI     A,3
        STA     RCYC            ;SPEED-UP RANDOMIZER
        RET

FILL:   CALL    EMPT            ;FIND EMPTY RANDOM LOCATION
FLL1:   CALL    RAND
        ANI     7
        CMP     E
        JNC     FLL1
        ORA     D
        MOV     M,A
        DCX     B
        MOV     A,B
        ORA     C
        JNZ     FILL
        RET

CLR:    LXI     H,SEA
        LXI     B,SEAL
CLR1:   MVI     M,0
        INX     H
        DCX     B
        MOV     A,B
        ORA     C
        JNZ     CLR1
        RET

EMPT:   CALL    RAND
        MOV     L,A
        CALL    RAND
        ANI     7
        MOV     H,A
        CALL    INDX
        MOV     A,M
        ANA     A
        JNZ     EMPT
        RET

INDX:   PUSH    D
        LXI     D,SEA
        DAD     D
        POP     D
        RET

;       FOR ALL SEA, IF FISH MAKE RANDOM MOVE

FSCA:   LXI     H,SEA
        LXI     B,SEAL
FSC1:   MOV     A,M
        ANI     MASK
        CPI     FBAS
        CZ      FMOV            ;FOUND UNMOVED FISH
        INX     H
        DCX     B
        MOV     A,B
        ORA     C
        JNZ     FSC1
        RET
```

```
FMOV:   PUSH    B               ;SAVE COUNTER
        MVI     B,0             ;SEARCH CONDITION
        CALL    PICK            ;FIND MOVE THAT MEETS COND.
        JZ      FMV1            ;NONE SO DO NOTHING
        MOV     A,M             ;FETCH FISH
        INR     A               ;INCREASE AGE
        ANI     7               ;MASK AGE
        MOV     B,A
        LDA     FBA             ;BREEDING AGE
        CMP     B
        MVI     A,FBAS+MF       ;MOVED FISH
        JNZ     FMV2
        MOV     M,A             ;REJUVENATE FISH
        STAX    D               ;NEW FISH
FMV1:   POP     B
        RET

FMV2:   ORA     B               ;ADD AGE TO FISH
        STAX    D               ;RETURN TO SEA
        MVI     M,0             ;CLEAR OLD POSITION
        POP     B
        RET

DTBL:   DB      1,0,0,-1,-1,0,0,1
        DB      1,0,0,-1,-1,0,0,1
        DB      1,0,0,-1,-1,0,0,1

;       SEARCH FOUR ADJACENT LOCATIONS
;       B = SEARCH PATTERN, HL = CURRENT LOCATION
;       BC, DE LOST. HL UNCHANGED. DE = FOUND LOCATION
;       ZERO FLAG IF NO MATCH FOUND

PICK:   PUSH    H
        MOV     A,H
        SUI     SEA/256
        MOV     H,A
        MOV     E,L
        DAD     H
        DAD     H
        MOV     D,H             ;DE = UNMASKED XY
        CALL    RAND            ;A = RANDOM BYTE
        ANI     3               ;A = 0 TO 3
        ADD     A
        LXI     H,DTBL          ;DIRECTION TABLE
        CALL    ADHL            ;INDEX INTO TABLE
        MVI     C,4
PIC1:   PUSH    D               ;SAVE XY POSITION
        MOV     A,E
        ADD     M
        INX     H
        ANI     3FH
        MOV     E,A
        MOV     A,D
        ADD     M
        INX     H
        RRC
        RRC
        MOV     D,A
        ANI     0C0H
        ORA     E
        MOV     E,A
        MOV     A,D
        ANI     7
        ADI     SEA/256
        MOV     D,A             ;DE = NEW SEA ADDRESS
        LDAX    D               ;WHAT'S THERE?
        ANI     MASK            ;MASK NON-ESSENTIALS
        CMP     B
        JZ      PIC2
        POP     D               ;GET XY POSITION
        DCR     C               ;MORE DIRECTIONS?
        JNZ     PIC1
        POP     H               ;CURRENT ADDRESS
        RET                     ;NOTHING FOUND
PIC2:   POP     H               ;DISCARD XY
        POP     H               ;OLD ADDRESS
        MVI     A,1
        ANA     A               ;SET NOT ZERO
        RET

;       MOVE SHARKS

SSCA:   LXI     H,SEA
        LXI     B,SEAL
SSC1:   MOV     A,M
        ANI     0C0H
        CPI     SBAS            ;UNMOVED SHARK?
        CZ      SMOV            ;FIND SHARK MOVE
        INX     H
        DCX     B
        MOV     A,B
        ORA     C
        JNZ     SSC1
        RET

;       SELECT MOVE FOR SHARK
;       BC = CURRENT POSITION, HL = SEA ADDRESS

SMOV:   PUSH    B               ;SAVE COUNTER
        MVI     B,FBAS          ;MASK FOR FISH
        CALL    PICK
        JZ      NEAT            ;NO FISH
        MVI     C,0             ;STARVE LEVEL
        JMP     SAGE

NEAT:   MVI     B,0             ;MASK FOR SPACE
        CALL    PICK
        JZ      SMV1            ;NO SPACE
        MOV     A,M             ;FETCH SHARK
        ADI     8               ;STEP STARVE
        ANI     38H
        MOV     C,A             ;STARVE AGE
        LDA     SSA
        CMP     C
        JZ      DIE             ;STARVE
SAGE:   MOV     A,M             ;FETCH SHARK AGAIN
        INR     A               ;STEP AGE
        ANI     7
        MOV     B,A             ;AGE
        LDA     SBA             ;BREED AGE
        CMP     B
        MVI     A,SBAS+MF       ;MOVED SHARK
        JZ      SBRD            ;BREED
        ORA     B               ;AGE
        ORA     C               ;STARVE LEVEL
        STAX    D               ;LOAD NEW SHARK
DIE:    MVI     M,0             ;DELETE OLD SHARK
SMV1:   POP     B               ;RESTORE COUNTER
        RET

SBRD:   MOV     M,A             ;NEW BORN SHARK
        ORA     C               ;STARVE LEVEL
        STAX    D               ;OLD HUNGRY SHARK
        POP     B
```

```
        RET

;       DISPLAY SEA AND CONTENTS

DISP:   LXI     H,7000H
        LXI     D,SEA
DSP1:   LDAX    D
        ANI     7FH             ;CLEAR MOVED FLAG
        STAX    D
        ANI     MASK
        CPI     FBAS
        LXI     B,BLNK
        JC      DSP2
        LXI     B,SSYM
        JNZ     DSP2
        LXI     B,FSYM
DSP2:   MOV     M,C
        INX     H
        MOV     M,B
        INX     H
        INX     D
        MOV     A,H
        CPI     80H
        JNZ     DSP1
        RET

SCOR:   CALL    CENS            ;COUNT FISHES & SHARKS
        CALL    STR
        DB      8,6,3,'         ',8,0
        LHLD    NOF
        CALL    PRNT
        LHLD    NOS
        CALL    PRNT
        RET

CENS:   LXI     H,SEA
        LXI     B,0
        LXI     D,0
CEN1:   MOV     A,M
        ANI     MASK
        JZ      CEN2
        INX     B               ;FISH COUNT
        CPI     FBAS
        JZ      CEN2
        DCX     B               ;UNCOUNT FISH
        INX     D               ;SHARK COUNT
CEN2:   INX     H
        MOV     A,H
        CPI     ESEA/256
        JNZ     CEN1
        XCHG
        SHLD    NOS
        MOV     L,C
        MOV     H,B
        SHLD    NOF
        RET

;       PRINT FISH AND SHARK COUNTS

PRNT:   LXI     B,0
        MOV     E,C
        PUSH    B
        LXI     B,10
        PUSH    B
        LXI     B,100
        PUSH    B
        LXI     B,1000
PRT1:   PUSH    B
        MOV     A,B
        CMA
        MOV     B,A
        MOV     A,C
        CMA
        MOV     C,A
        INX     B
        MVI     A,-1
PRT2:   INR     A
        DAD     B
        JC      PRT2
        POP     B
        DAD     B
        ADI     '0'
        INR     E
        CPI     '0'
        JNZ     PNT3
        DCR     E
        JNZ     PNT3
        MVI     A,' '
PNT3:   CALL    TTO
        POP     B
        MOV     A,C
        ANA     A
        JNZ     PRT1
        MOV     A,L
        ADI     '0'
        CALL    TTO
        MVI     A,9
        CALL    TTO
        RET

;       RANDOM NUMBER GENERATOR

RAND:   PUSH    B
        PUSH    H
        LDA     RCYC
        MOV     C,A
        LHLD    RANN
RND1:   MOV     A,H
        DAD     H
        ANI     8A6H
        JPE     RND2
        INX     H
RND2:   DCR     C
        JNZ     RND1
        SHLD    RANN
        MOV     A,L
        POP     H
        POP     B
        RET

;       IN-LINE STRING PRINT

STR:    POP     H
        MOV     A,M
        INX     H
        PUSH    H
        ANA     A
        RZ
        CALL    TTO
        JMP     STR

ORG     B6E0H

STK:
IBUF:   DS      6
NOF:    DS      2
NOS:    DS      2
FBA:    DS      1
SBA:    DS      1
SSA:    DS      1
RANN:   DS      2
RCYC:   DS      1

ORG     B700H

SEAL    EQU     800H

SEA:    DS      SEAL
ESEA:

FBAS    EQU     20H             ;BASIC FISH
SBAS    EQU     40H             ;BASIC SHARK
MASK    EQU     60H             ;IDENTIFICATION MASK
MF      EQU     80H             ;MOVED FLAG

BLNK    EQU     2020H           ;BLANK
SSYM    EQU     2173H           ;SHARK SYMBOL
FSYM    EQU     2670H           ;FISH SYMBOL

ESCD    EQU     16FFH           ;(16FFH)
CRLF    EQU     17C1H           ;(17C1H)
TTO     EQU     17C8H           ;(17C8H)
ADHL    EQU     194EH           ;(194EH)

UINP    EQU     81C5H
KBFL    EQU     81DFH
TSEC    EQU     81BAH
TMIN    EQU     81B9H
RDY     EQU     81FFH

END     WTOR
```

NOTICE: Intelligent Computer Systems, Huntsville

I am sorry to report that the Muellers have moved back to Germany and ceased their business activities in the United States. Before leaving, they discussed with J. Norris the possibility of COLORCUE becoming their CCII software representative in the U.S. Nothing further has been heard about such an arrangement as of this issue. This means that there is no legal source for CCII software in this country. We hope to hear further from Irmgard Mueller about this and will report any change in status in CHIP. In the meantime, please contact COLORCUE about your software needs just in case we might be able to give assistance.

SEARCH

A string search program for the 8000.

Bob Mendelson
27 Somerset Place
Murray Hill, NJ 07974

This program searches for a given string starting at a given address, both parameters set by the operator. It prints the start address of the found string location followed by the full string and all the characters after the string until a non-alphanumeric character is reached. It then prints the end address of the long string.

The nice feature of the program is that it prints out a long string while requiring the operator to enter only the first few search characters. For example, names and addresses can be found by typing only enough characters to identify the required string. Even if several locations contain the same characters the string printout makes it easy to select the desired string.

Here is how it works. The keyboard flags are first cleared. The program then asks for the start address of the search. If the operator knows the approximate location it will save a bit of time, but even running through the full 65535 address space takes only a few seconds. The program now asks for the key characters of the search string. When this data is entered the search begins. The character at the search start ad?The program then goes back to the start address and proceeds to check each character to be certain each is alphanumeric (that is, having an ASCII value between 20H and 7FH.) If it is, it will be printed. If not, the alphanumeric print will stop and the end address will be printed by the EDIT2 routine.

The STR3 routine asks the operator to chose among "Continue", "Start a new string", or "End operation" options. This program has proven so valuable that it is best stored on EPROM for immediate and easy use.

```
                ;STRING SEARCH
                ;
                ;BY R. MENDELSON, MODIFIED FOR COLORCUE
                ;VERSION 3-23-85

        ;THIS ROUTINE SEARCHES FOR A GIVEN STRING STARTING
        ;AT A GIVEN ADDRESS BOTH SET BY THE OPERATOR
        ;IT PRINTS START ADDRESS OF THE FOUND STRING LOCATION
        ;FOLLOWED BY THE STRING AND ALL CHARACTERS AFTER IT,
        ;UNTIL A NON-ALPHA-NUMERIC IS REACHED.  IT THEN PRINTS
        ;THE END ADDRESS OF LONG STRING.
        ;

          ;EQUATES, COMMON
          ;----------------------
0103                CI      EQU     0103H   ;CONSOLE INPUT
0109                CO      EQU     0109H   ;CONSOLE OUTPUT
010F                LO      EQU     010FH   ;LIST OUT
9FFF                KEYBF1  EQU     09FFFH  ;KYBD READY FLAG
9FDF                KEYBF2  EQU     09FDFH  ;KEYBOARD FLAG
012A                OSTR    EQU     0012AH  ;STRING OUTPUT
0EE4                EXHL    EQU     0EE4H   ;PRINT CONTENTS OF HL ON SCREEN
0133                EXPR    EQU     0133H   ;INPUT 4 DIGIT HEX ADDDR
0006                COLOR   EQU     06H     ;CCI STATUS
0010                BLACK   EQU     16
0011                RED     EQU     17
0013                YEL     EQU     19
0016                CYAN    EQU     22
000D                CR      EQU     13
000A                LF      EQU     10
000C                EP      EQU     12
000E                BA7ON   EQU     14
000F                BA7OFF  EQU     15      ;BLINK A7 OFF
001D                FORGND  EQU     29
001E                BKGND   EQU     30
00EF                EOS     EQU     239     ;END OF STRING

          ;EQUATES, SEARCH
          ;---------------------

9E20                ADDR1   EQU     9E20H   ;START OF SEARCH ADDRESS
9E22                LEN1    EQU     9E22H   ;LENGTH OF STRING
9E24                ADDR2   EQU     9E24H   ;START OF 'SEARCH' ADDRESS
9E26                ADDR3   EQU     9E26H   ;END OF STRING ADDRESS
9E30                STR1    EQU     9E30H   ;STRING BEING SOUGHT

0000                        .IFP1
                            .PRINT  'SEARCH FOR STRING'
                            .PRINT  'VERSION 3-85'
                            .PRINT  'ENTER ROUTINE ORIGIN '
                            .INPUT  RBEGIN
                            ENDIF

0000                        ORG     RBEGIN

F000 AF           SEARCH:   XRA     A       ;ZERO
F001 32 FF 9F               STA     KEYBF1  ;KYBD FLAG 1
F004 32 DF 9F               STA     KEYBF2  ;KYBD FLAG 2

F007 CD 76 F1     SERCH:    CALL    MESS    ;PRINT MESSAGE

F00A 06 13 0E     ADDR:     DB      COLOR,YEL,BA7ON,BKGND,BLACK,FORGND,EP
     1E 10 1D
     0C
F011 53 45 41               DB      'SEARCH-',BA7OFF
     52 43 48
     2D 0F
```

```
F019 0D 0A 0A           DB      CR,LF,LF,CYAN,'START ADDRESS ) ',RED
     16 53 54
     41 52 54
     20 41 44
     44 52 45
     53 53 20
     3E 20 11
F02E EF                 DB      EOS
F02F CD 61 F1           CALL    HEXIN   ;
F032 22 20 9E           SHLD    ADDR1   ;STORE ADDRESS OF SEARCH START

F035 21 68 F1   STR:    LXI     H,MSG1

F038 CD E4 F0           CALL    INPUT

F03B 2A 20 9E   BEGIN:  LHLD    ADDR1   ;LOAD STARTING ADDRESS
F03E 11 30 9E   CONT:   LXI     D,STR1  ;LOAD STRING
F041 3A 22 9E           LDA     LEN1    ;LOAD STRING LENGTH
F044 47                 MOV     B,A     ;MOV LENGTH TO B

                ;COMPARE STRING WITH MEMORY

F045 1A         COMP:   LDAX    D       ;LOAD STRING BYTE
F046 BE                 CMP     M       ;COMPARE WITH MEMORY
F047 C2 53 F0           JNZ     NOTEQ   ;BRANCH IF NOT EQUAL
F04A 23                 INX     H       ;INCREMENT HL POINTER
F04B 05                 DCR     B       ;DECREMENT LENGTH COUNTER
F04C CA 5D F0           JZ      EQUAL   ;BRANCH IF DONE
F04F 13                 INX     D       ;INCREMENT STRING POINTER
F050 C3 45 F0           JMP     COMP    ;CONTINUE TESTING

                ;STRING DID NOT COMPARE, ADVANCE MEMORY START POINT

F053 2A 20 9E   NOTEQ:  LHLD    ADDR1   ;LOAD NEW START ADDRESS
F056 23                 INX     H       ;INCREMENT
F057 22 20 9E           SHLD    ADDR1   ;SAVE END ADDR OF FIND
F05A C3 3E F0           JMP     CONT    ;START TEST AGAIN

                ;PATTERN DID COMPARE

F05D 3A 22 9E   EQUAL:  LDA     LEN1    ;POINT TO LENGTH
F060 2B         COUNT:  DCX     H       ;REDUCE ADDR AND COUNT
F061 3D                 DCR     A
F062 C2 60 F0           JNZ     COUNT   ;ZERØ?
F065 22 24 9E           SHLD    ADDR2   ;SAVE START ADDR OF FIND
F068 CD 76 F1   STOP:   CALL    MESS
F06B 0D 0A 0A           DB      CR,LF,LF,YEL,'ADDRESS',RED,' ) '
     13 41 44
     44 52 45
     53 53 11
     20 3E 20
F07A 13 EF              DB      YEL,EOS
F07C 2A 24 9E           LHLD    ADDR2   ;RECOVER THE START ADDR TO HL REG
F07F CD E4 0E           CALL    EXHL    ;PRINT START ADDR OF STRING

F082 CD 76 F1   EDIT1:  CALL    MESS
F085 16 20 EF           DB      CYAN,' ',EOS
F088 2A 24 9E           LHLD    ADDR2   ;GET START ADDR OF FIND
F08B 7E         EDIT1A: MOV     A,M     ;GET STRING CHARACTER
F08C 23                 INX     H

                ;IS CHAR NON-ALPHA-NUMERIC?

F08D D6 20              SUI     20H     ;IS IT LESS THAN 20H?
F08F DA 9D F0           JC      EDIT2   ;YES, END STRING PRINT

F092 D6 60              SUI     60H     ;IS IT GREATER THAN 7FH?
                                        ;-60H=7FH-1FH, 20H BECOMES 1FH BECAUSE-
                                        ;---SUBTRACT IS COMPLIMENT + 1
F094 D2 9D F0           JNC     EDIT2   ;YES, END STRING PRINT
F097 CD 09 01           CALL    CO
F09A C3 8B F0           JMP     EDIT1A  ;LOOK AGAIN
F09D 22 26 9E   EDIT2:  SHLD    ADDR3   ;SAVE END ADDR OF STRING
F0A0 CD 76 F1           CALL    MESS
F0A3 13 20 54           DB      YEL,' TO',RED,' ) ',YEL,EOS
     4F 11 20
     3E 20 13
     EF
F0AD 2A 26 9E           LHLD    ADDR3   ;RECOVER END ADDR (1 BYTE AFTER STR +EF)
F0B0 2B                 DCX     H       ;ADJUST TO LAST CHARACTER ADDRESS
F0B1 2B                 DCX     H       ;
F0B2 22 26 9E           SHLD    ADDR3   ;SAVE END ADDR OF STRING
F0B5 CD E4 0E           CALL    EXHL    ;PRINT ADDR OF END OF STRING
```

```
F0B8 CD 76 F1   STR3:   CALL    MESS    ;
F0BB 0D 0A 0A           DB      CR,LF,LF,RED,'C',CYAN,'ONT, '
     11 43 16
     4F 4E 54
     2C 20
F0C6 11 4E 16           DB      RED,'N',CYAN,'EW, '
     45 57 2C
     20
F0CD 11 45 16           DB      RED,'E',CYAN,'ND > ',EOS
     4E 44 20
     3E 20 EF
F0D6 CD 03 01           CALL    CI      ;GET ANSWER
F0D9 FE 43              CPI     'C'     ;IF ANS IS Yes THEN
F0DB CA 3E F0           JZ      CONT    ;CONTINUE TO LOOK FOR SAME STRING FROM
                                        ;END OF ADDRESS OF LAST FIND
F0DE FE 4E              CPI     'N'     ;IF ANS IS New THEN
F0E0 CA 07 F0           JZ      SERCH   ;BEGIN AGAIN

                ;ANY OTHER CHARACTER WILL TERMINATE PROGRAM

F0E3 CF                 RST     1       ;TERMINATE PROGRAM

                ;INPUT ROUTINE
                ;-------------

                ;INPUT ROUTINE: ALLOWS BACKSPACE (ERASES CHARACTER),
                ;ERASE LINE AND ALSO LINE EDIT.  BACKSPACE TO THE ERROR,
                ;CORRECT AND CURSOR RIGHT TILL END OF INPUTS. CURSOR
                ;WILL GO NO FURTHER.

F0E4 E5         INPUT:  PUSH    H       ;SAVE POINTER TO STRING
F0E5 CD 2A 01           CALL    OSTR
F0E8 21 30 9E           LXI     H,STR1  ;POINT TO OUR INPUT BUFFER
F0EB AF                 XRA     A       ;ZERO 'A'
F0EC 06 20              MVI     B,20H
F0EE 77         INPUT1: MOV     M,A
F0EF 23                 INX     H
F0F0 05                 DCR     B
F0F1 C2 EE F0           JNZ     INPUT1  ;ZERO 32 PLACES IN THE BUFFER

F0F4 21 30 9E   Y1:     LXI     H,STR1  ;POINT TO THE BUFFER
F0F7 06 00              MVI     B,0     ;'B' AS A COUNTER
F0F9 CD 03 01   Y2:     CALL    CI      ;INPUT 1 HIT OF KB
F0FC FE 1A              CPI     1AH     ;IS IT A BACKSPACE ?
F0FE CA 16 F1           JZ      Y4      ;IF YES, JUMP
F101 FE 0B              CPI     0BH     ;IS IT AN ERASE LINE ?
F103 CA 2B F1           JZ      Y6      ;IF YES, JUMP
F106 FE 19              CPI     19H     ;IS IT A CURSOR RIGHT ?
F108 CA 50 F1           JZ      Y8      ;IF YES, JUMP
F10B FE 0D              CPI     0DH     ;IS IT A 'RETURN' ?
F10D CA 38 F1           JZ      Y7      ;IF YES, JUMP
                ;NONE OF THE ABOVE

F110 77                 MOV     M,A     ;PUT INPUT IN BUFFER
F111 23         Y3:     INX     H       ;INCREMENT BUFFER
F112 04                 INR     B       ;INCREMENT COUNTER
F113 C3 F9 F0           JMP     Y2      ;LOOP till ONE OF ABOVE

                ;COMES HERE IF INPUT IS A BACKSPACE

F116 2B         Y4:     DCX     H       ;DECREMENT BUFFER
F117 05                 DCR     B       ;DECREMENT COUNTER
F118 CA 2B F1           JZ      Y6      ;IF RESULT OF BACKSPACE =) THEN..
F11B FA 2B F1           JM      Y6      ;INPUTS, ERASE LINE & DO ALL OVER
F11E 3E 20              MVI     A,20H   ;ELSE,.........................
F120 CD 0F 01           CALL    LO      ;......ERASE THE CHARACTER......
F123 3E 1A      Y5:     MVI     A,1AH   ;......BACK UP THE CURSOR.......
F125 CD 0F 01           CALL    LO
F128 C3 F9 F0           JMP     Y2      ;GO GET NEXT INPUT

                ;COMES HERE IF INPUT WAS AN ERASE LINE

F12B 3E 0B      Y6:     MVI     A,0BH
F12D CD 0F 01           CALL    LO      ;ERASE THE LINE
F130 E1                 POP     H       ;GET POINTER TO STRING
F131 E5                 PUSH    H       ;SAVE IT AGAIN
F132 CD 2A 01           CALL    OSTR    ;REPRINT THE STRING
F135 C3 F4 F0           JMP     Y1      ;GO DO IT ALL OVER AGAIN
```

```
                    ;COMES HERE IF INPUT WAS A 'RETURN'
F138 AF             Y7:     XRA     A       ;TEST FOR Cr W/O INPUT
F139 B8                     CMP     B       ;IS COUNTER STILL ZERO ?
F13A CA 2B F1               JZ      Y6      ;IF YES, DO IT OVER
F13D 78             Y7A:    MOV     A,B     ;NEEDED TO COUNT STRING LENGTH
F13E 32 22 9E               STA     LEN1    ;FOR SEARCH PRG.
F141 3E EF                  MVI     A,239   ;--PUT AN 'EF' AT END OF-----
F143 77                     MOV     M,A     ;--INPUTS, MAKING A STRING OF IT
F144 3E 0D                  MVI     A,13    ;Cr
F146 CD 0F 01               CALL    LO      ;SEND
F149 3E 0A                  MVI     A,10    ;Lf
F14B CD 0F 01               CALL    LO      ;SEND
F14E E1                     POP     H       ;ELIMINATE POINTER TO STRING
F14F C9                     RET
                    ;COMES HERE IF INPUT WAS A CURSOR RIGHT (_)

F150 7E             Y8:     MOV     A,M     ;FETCH CHARACTER
F151 A7                     ANA     A       ;IS MEMORY A ZERO ?
F152 CA 23 F1               JZ      Y5      ;IF YES, IGNORE CUR.RIGHT
F155 3E 1A                  MVI     A,1AH   ;MEMORY not A ZERO
F157 CD 0F 01               CALL    LO      ;BACK CURSOR UP ONE
F15A 7E                     MOV     A,M     ;FETCH THE CHARACTER
F15B CD 0F 01               CALL    LO      ;PRINT IT
F15E C3 11 F1               JMP     Y3      ;GO GET NEXT INPUT

F161 0E 01          HEXIN:  MVI     C,1     ;COUNTS FOR FOUR BYTES OF ADDRESS
F163 CD 33 01               CALL    EXPR    ;---FETCH BYTES TO STACK, THEN---
F166 E1                     POP     H       ;--PLACE DATA IN HL REG AS START ADDR.--
F167 C9                     RET

F168 0D 0A 16       MSG1:   DB      CR,LF,CYAN,'STRING ) ',RED
     53 54 52
     49 4E 47
     20 3E 20
     11
F175 EF                     DB      EOS

                    ;CALL FOR MESSAGE PRINT

F176 E1             MESS:   POP     H       ;FETCH POINTER TO MESSAGE
F177 CD 2A 01               CALL    OSTR    ;PRINT MESSAGE
F17A E9                     PCHL            ;RETURN TO PROGRAM

F17B                REND:

F17B                        .IFP1
                            .PRINT  'PROGRAM ORIGIN = ',RBEGIN
                            .PRINT  'PROGRAM SIZE   = ',REND-RBEGIN
                            .PRINT  'PROGRAM END    = ',REND
                            ENDIF

F17B F000                   END     SEARCH
```

A BUG IN FASBAS!

Peter Hiner

I regret to advise those of you who have the latest version (v12.24) of FASBAS that I have allowed a fatal bug to creep in. This bug was not present in earlier versions, so the corrective action outlined below should be applied only to v12.24.

```
ESC W                 ; to be sure of a clean start
ESC D                 ; to enter FCS
LOAD FASBAS.PRG       ; in response to FCS
ESC E                 ; to return to BASIC
POKE 39005,4          ; to eliminate the bug
POKE 39234,53         ; optional entry to change
                      ;  display header to v12.25
ESC D                 ; back to FCS
SAVE FASBAS.PRG;25 82A0 173F
                      ; save update to disk
```

Please check each entry carefully before hitting RET, and at the end, check the disk directory to be sure you have saved FASBAS.PRG;25 with the same values for size and load address as for v12.24.

Columns	SIZE	LBC	LADR	SADR
should read:	002F	3F	82A0	82A0

Credit for discovery of this bug goes to Doug Van Putte who presented me with a BASIC program that appeared to compile satisfactorily but caused the assembler (FBASM) to crash. I discovered the reason for this to be an error in the size of output buffer allocated within FASBAS. Running on a 32K machine, FASBAS would overflow the top of RAM if a medium to large BASIC program was being compiled. On a 16K machine not even a small BASIC program could be compiled! I am sorry for any inconvenience caused. This debugged release of FASBAS will now be officially called v12.25. □

TRACE

A printing disassembler for the CCII.

Thomas Wulff
80 Bowen Road
Churchville, NY 14428

TRACE is a program that allows the user to observe the operation of a machine language program one instruction at a time. On each machine language instruction, TRACE causes the instruction to take place and prints the results of the instruction on the screen. The "results" include a printout of the status of all the 8080 registers, the stack contents, memory addresses, and the program counter. TRACE differs from other disassemblers in that the operations actually occur. Therefore, JMPs, and CALLs are properly executed. TRACE acts as though it were a microprocessor but it only simulates a microprocessor, in a sense. For example, if the code being traced places characters on the screen, they will not actually appear there, but the register and memory contents will be accurately simulated.

The operation of TRACE is more clearly seen in the accompanying printout.

TRACE requires a single disk, at least 16K RAM, and a printer connected to the MODEM port. To operate the program, a machine language auxiliary program, PR0042.PRG;1 is loaded into memory. If PR0042 is on the same disk as TRACE, it will automatically load before tracing begins. If, for some reason, PR0042 becomes altered during the run, it may be reloaded by inserting the program disk and entering 'RUN 4000' from Basic.

The program prompts the user for several replies at the start of the program. The user must, at this time, specify the following parameters:

1) the number of printer lines per page. The default is 80 lines/page, although most printer paper will require an entry of 66 lines or less. The program will generate a page command based on this number. The title, page number, and

```
PAGE  1
GAME.PRG (CHRIS ZERR)
START ADDRESS =  36864

          CMD            !SZAPC! A  ! B  ! C  ! D  ! E  ! H  ! L  !DATA SP ADR!MEM  MADR
-----------------------+-----+----+----+----+----+----+----+----+-----------+---------
36864  34 SHLD     16698!     !    !    !    !    !    !    !    !           !18508
36867 172 XRA H          !00010!  9 !    !    !    !    !    !    !           !
36868  55 STC            !00011!    !    !    !    !    !    !    !           !
36869  58 LDA       8328!00011! 69 !    !    !    !    !    !    !           !
36872  55 STC            !00011!    !    !    !    !    !    !    !           !
-----------------------+-----+----+----+----+----+----+----+----+-----------+---------
36873  48 *** UDF <<     !     !    !    !    !    !    !    !    !           !
36874  50 STA      36410!     !    !    !    !    !    !    !    !           !
36877  32 *** UDF <<     !     !    !    !    !    !    !    !    !           !
36878  80 MOV D,B        !     !    !    !    ! 66!    !    !    !           !
36879  83 MOV D,E        !     !    !    !    ! 69!    !    !    !           !
-----------------------+-----+----+----+----+----+----+----+----+-----------+---------
36880  87 MOV D,A        !     !    !    !    ! 69!    !    !    !           !
36881   0 NOP            !     !    !    !    !    !    !    !    !           !
36882  77 MOV C,L        !     !    !    ! 76!    !    !    !    !           !
36883 144 SUB B          !00110!  3 !    !    !    !    !    !    !           !
36884 202 JZ       21250!     !    !    !    !    !    !    !    !           !
-----------------------+-----+----+----+----+----+----+----+----+-----------+---------
36887  88 MOV E,B        !     !    !    !    !    ! 66!    !    !           !
36888 172 XRA H          !00010! 75 !    !    !    !    !    !    !           !
36889 189 CMP L          !10011!    !    !    !    !    !    !    !           !
36890  40 *** UDF <<     !     !    !    !    !    !    !    !    !           !
36891  83 MOV D,E        !     !    !    !    ! 66!    !    !    !           !
-----------------------+-----+----+----+----+----+----+----+----+-----------+---------
36892  76 MOV C,H        !     !    !    ! 72!    !    !    !    !           !
36893  79 MOV C,A        !     !    !    ! 75!    !    !    !    !           !
36894  41 DAD H          !10010!    !    !    !    !144!152!    !           !
36895 164 AND H          !01110!  0 !    !    !    !    !    !    !           !
36896  50 STA      13877!     !    !    !    !    !    !    !    !           !
-----------------------+-----+----+----+----+----+----+----+----+-----------+---------
36899 166 AND M          !01010!  0 !    !    !    !    !    !    !           !
36900 189 CMP L          !00001!    !    !    !    !    !    !    !           !
36901  40 *** UDF <<     !     !    !    !    !    !    !    !    !           !
36902  83 MOV D,E        !     !    !    !    ! 66!    !    !    !           !
36903  72 MOV C,B        !     !    !    ! 66!    !    !    !    !           !
-----------------------+-----+----+----+----+----+----+----+----+-----------+---------
36904  73 MOV C,C        !     !    !    ! 66!    !    !    !    !           !
36905  41 DAD H          !00001!    !    !    !    ! 33! 48!    !           !
36906  58 LDA       9284!00001!130 !    !    !    !    !    !    !           !
36909  40 *** UDF <<     !     !    !    !    !    !    !    !    !           !
36910  49 LXI SP , 10547!     !    !    !    !    !    !    !    !52705 10547!
-----------------------+-----+----+----+----+----+----+----+----+-----------+---------
```

start address are printed as a header on each page.

2) the number of characters/line for printout. The default width is 80 characters.

3) the title of the program. Any text header may be entered here, and will be printed at the top of each printout page. The header must be restricted to the number of characters/line specified above.

4) the start address of the program. This is the address at which TRACE will begin disassembly. You must be certain that the address you supply is a logical starting address of a program or interrupt. The starting address is in decimal.

5) the printout destination, (S)creen or (P)rinter.

6) the printer Baud rate. You will enter the exact rate (ie: '2400', or '300'). TRACE does not recognize the familiar 1-7 entries here.

After this preliminary information is entered, the program will 'trace' until a HLT instruction is reached or the ATTEN/BREAK key is pressed. Since TRACE uses a machine language program, the registers and other processor parameters are not destroyed when the program is re-run, provided that PR0042 has not been reloaded in the meantime.

TRACE places PR0042.PRG immediately after the Basic portion of TRACE. If you alter the Basic portion, in terms of its byte count, and get an OM error, you do not have to reassemble PR0042. At the beginning of the Basic program you will see 'A0 = 165'. This is used to set the top of Basic RAM. It is also used in placing PR0042 in the appropriate memory area. Increasing 'A0' will provide more Basic memory space and also load PR0042 to a higher address. This load includes address changes in PR0042. TRACE also identifies any undefined functions but does not operate on them.

TRACE is available from the CHIP library in the form of seven files, including all source code and the assembled versions. The following printout demonstrates a typical TRACE output for the first few lines of code from Chris Zerr's GAME.PRG from the animation article in COLORCUE, VOL VI, No 5, p 12. □

Assembly Language Programming, Part XVII:

"Pesticidal Programming" Using ***IDA***'s Monitor.

W. S. Whilly

As promised (last year!) we will continue with our exploration of IDA, this time looking at the Interpret command and a few esoteric facilities of IDA.

IDA's monitor operates on a pseudo-8080 processor; that is, the register contents are simulated on the CRT for readout purposes. The best way to explore this marvelous instructional and debugging aid is with a short test program. Listing I has an assembler printout of the program we will use. Type it on your screen editor now and assemble it.

To explore a PRG file with IDA's monitor, first RUN IDAE (or IDA4). Next, I usually fill a good portion of memory with 00H so I can tell exactly where my program begins and ends. To do this type as follows:

```
IDA>F 8200 DFFF 00
```

Now we can load the PRG file from IDA:

```
IDA>XLOA MON.PRG
```

We do not specify a loading address, so IDA will use the loading address in the directory (8400H). You may verify the presence of the program by disassembling at 8400H:

```
IDA>D 8200 15+
```

...and there it is! We will now step through the program one instruction at a time, using the 'I' command. Clear the screen with the ERASE PAGE key. Now type in this command:

```
IDA>I8400 8400
```

In this command format the first 8400 is the address at which you want to begin executing. The second 8400 is a break point. Making the two numbers the same allows an examination of all the initial conditions without executing any of the program instructions. What you should see is this line on the CRT:

```
8400   210000   LXI     H,0000H     ; ???
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8400 0000 0000 0000 00 0 0 0 0 0 C3 40D2 40DC C33E C532 3281
```

The top line of the display shows the next instruction to be interpreted, in the usual IDA disassembly format. On the next line, from left to right, the display shows the address in the program counter (8400), the BC, DE, and HL registers (all 00H), the accumulator, five flag bits, the current contents of the memory location held in HL (really the first byte of the operating system, since HL holds 0000H), the IDA stack pointer (40D2 in my computer, but your contents may differ depending on what you have done most recently with the computer and which version of IDA you are using), the current stack contents (SP + 0) plus the last three stack entries.

Each instruction, in turn, is processed by IDA by pressing the BLUE color key (or CTL T). Press this key once now. The following line will be displayed:

```
8403   39        DAD     SP              ; 9
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8403 0006 0000 0000 00 0 0 0 0 0 0 C3 40D2 4CF8 C33E C532 3281
```

The program counter has advanced to 8403 because LXI H,xxxx is a three byte instruction. You will notice a change in the stack contents (at the star in the line above) because IDA is using the stack as it runs the monitor. We are in the process of assigning our own stack so it will reflect only our own work and not that of IDA herself. Press BLUE again and you will see the effect of DAD SP:

```
8404   22009A   SHLD    9A00H           ; "@Z
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8404 0000 0000 40D2 00 0 0 0 0 0 EA 40D2 4CF8 C33E C532 3281
```

The HL registers now hold the current stack pointer address (40D2 for me). You'll notice that M has changed, because the memory pointer has changed. M does not hold a byte shown in the first stack position on the screen, as we would expect, because IDA is deceiving us a bit about the 'real' contents of HL. However, we will change all of that when our own stack is clearly defined. When we press BLUE again, we will be saving the IDA stack pointer at address 9A00H. You could leave the interpreter and use IDA's Peek provision to examine that address, but we've done a clever thing. Press BLUE:

```
8407   31009A   LXI     SP,9A00H              ; 1@Z
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8407 0000 0000 40D2 00 0 0 0 0 0 EA 40D2 4CF8 C33E C532 3281
```

You'll see the contents of address 9A00H in the stack pointer address (SP + 0), which is right where we placed it! The next instruction, LXI,SP 9A00H will establish our new stack. Press BLUE and see the following:

```
840A   010201   LXI     B,0102H     ; ???
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
840A 0000 0000 40D2 00 0 0 0 0 0 EA 9A00 40D2 0000 0000 0000
```

The stack pointer is now 9A00H, and since we cleared memory in that area, all the stack positions are 0000H except for the 9A00H we recently placed there. This makes it easy to follow stack operations. Press BLUE:

```
840D   168F     MVI     D,8FH       ; VO   143
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
840D 0102 0000 40D2 00 0 0 0 0 0 EA 9A00 40D2 0000 0000 0000
```

The BC register pair has some new visitors. Press BLUE:

```
840F   1EFF     MVI     E,FFH       ; ^   255
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
840F 0102 8F00 40D2 00 0 0 0 0 0 EA 9A00 40D2 0000 0000 0000
```

So does D. Press BLUE:

```
8411   210000   LXI     H,0000H     ; ???
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8411 0102 8FFF 40D2 00 0 0 0 0 0 EA 9A00 40D2 0000 0000 0000
```

...and E. Press BLUE:

```
8414   86       ADD     B           ; @
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8414 0102 8FFF 0000 00 0 0 0 0 0 C3 9A00 40D2 0000 0000 0000
```

..as does HL. Notice that the old memory byte is back (C3). Press BLUE:

```
8415   F5       PUSH    PSW             ; u
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8415 0102 8FFF 0000 01 0 0 0 0 0 C3 9A00 40D2 0000 0000 0000
```

The accumulator now hold the sum of 00H and 01H (A = A + B). ADD puts the sum of A + B in A. Press BLUE:

```
8416   81       ADD     C           ; A
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8416 0102 8FFF 0000 01 0 0 0 0 0 C3 99FE 0102 40D2 0000 0000
```

We have PUSHed the accumulator and the flag register onto the stack (0102). '01' is the accumulator. What is the '02'? It's bit 1 of the flag register. Even though all the flags are zero at this point, remember that the flag register is eight bits wide but only five of these bits are used for flags. There are three bits unaccounted for here, and I suspect they are being used by the 8080 for something. If someone knows 'for what' please contact me at once! We know, at least, that bit 1 is set, because that's the only way to get '02' in the flag register. [1] Press BLUE:

```
8417   F5       PUSH    PSW             ; u
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8417 0102 8FFF 0000 03 0 1 0 0 0 C3 99FE 0102 40D2 0000 0000
```

We have added C to A (A = A + C). The Parity flag has been set. This happens when the number of binary '1's in the A register, following an operation, is an even number. The binary representation of 03H is '0000 0011'. ('0' is an even number also for purposes of parity.) Now watch the stack as we interpret the next instruction. The stack pointer will decrement, our old stack contents (0102) will be moved over to the right, and the new contents put in its former place on the CRT. Press BLUE:

```
8418   320090   STA     9000H       ; 2@P
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8418 0102 8FFF 0000 03 0 1 0 0 0 C3 99FC 0306 0102 40D2 0000
```

We have added to the stack. Notice that the PSW reflects the flag changes we made. Next with STA 9000H, we will save the A register at address 9000H. You won't see anything yet in IDA except the change in the program counter. You could always Peek at 9000H to be sure it's there. Press BLUE:

```
841B   82       ADD     D           ; B
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
841B 0102 8FFF 0000 03 0 1 0 0 0 C3 99FC 0306 0102 40D2 0000
```

Now we will add D to A. Press BLUE:

```
841C   83       ADD     E           ; C
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
841C 0102 8FFF 0000 92 0 0 1 0 1 C3 99FC 0306 0102 40D2 0000
```

A = 92H. The Sign flag has been set. This happens when bit 7 of A holds a binary '1'. 92H is such a number ('1001 0010'). We will now add '0FFH' to 92H. Press BLUE:

```
841D   F5       PUSH    PSW             ; u
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
841D 0102 8FFF 0000 91 1 0 1 0 1 C3 99FC 0306 0102 40D2 0000
```

What's this?! It seems 92H + 0FFH = 91H. Is IDA poor in arithmetic? You will notice the Carry flag has been set, indicating an A register overflow. The 'real' answer should be '191F' but a single eight-bit register can hold no more than 'FFH'. So the 8080 has done what you and I do with a column overflow in addition, it has generated a 'carry.' If your program were adding large numbers, some provision would have to be made for using this overflow to ad-

just the higher portions of a large number accordingly. We are going to PUSH this high number and its flags onto the stack. Press BLUE:

```
841E   EB        XCHG              ; K
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
841E 0102 8FFF 0000 91 1 0 1 0 1 C3 99FA 9193 0306 0102 40D2
```

Next we will exchange the contents of the DE and HL register pairs. The numbers involved are not arbitrary for we are preparing to verify our STA 9000H instruction from line 13. Press BLUE:

```
841F   23        INX     H          ; #
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
841F 0102 0000 8FFF 91 1 0 1 0 1 00 99FA 9193 0306 0102 40D2
```

The next instruction will place 9000H in the HL registers. Notice that the contents under 'M' will change. Press BLUE:

```
8420   77        MOV     M,A        ; W
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8420 0102 0000 9000 91 1 0 1 0 1 03 99FA 9193 0306 0102 40D2
```

'M' shows the 03H we placed there way back in line 13. Let's copy our new A value to M. Press BLUE:

```
8421   97        SUB     A          ; W
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8421 0102 0000 9000 91 1 0 1 0 1 91 99FA 9193 0306 0102 40D2
```

Now we will subtract A from itself, which is a good way to clear the accumulator. Press BLUE:

LISTING I.

```
                ;MON.PRG: Demo program for IDA's Interpret Command.

                ;NOTE: before loading this program, clear memory
                ;      from 8200 to DFFF.   [IDA>F 8200 DFFF 00]

0000 (8400)             ORG     8400H

8400 210000     BEGIN:  LXI     H,0000H         ;Clear HL registers
8403 39                 DAD     SP              ;Double-add current
                                                ; stack pointer to HL
8404 22009A             SHLD    9A00H           ;Save stack address
                                                ; for end of program.
8407 31009A             LXI     SP,9A00H        ;New stack pointer

840A 010201             LXI     B,0102H         ;Set initial parameters
840D 168F               MVI     D,8FH           ; with both MVI and LXI
840F 1EFF               MVI     E,0FFH
8411 210000             LXI     H,0000H         ;Clear HL

8414 80         START:  ADD     B               ;A = A + B = 01H
8415 F5                 PUSH    PSW             ;Look at stack contents!
8416 81                 ADD     C               ;A = A + C = 03H
8417 F5                 PUSH    PSW
8418 320090             STA     9000H           ;Place sum in memory
841B 82                 ADD     D
841C 83                 ADD     E
841D F5                 PUSH    PSW

841E EB                 XCHG                    ;Swap HL and DE
841F 23                 INX     H               ;Increment HL to 9000H, our
                                                ; memory location
8420 77                 MOV     M,A
8421 97                 SUB     A               ;Now A = 0, note zero flag
8422 77                 MOV     M,A             ;Zero M

8423 D5                 PUSH    D               ;Tricky way to get a lot
8424 D5                 PUSH    D               ; of zeros on the stack
8425 D5                 PUSH    D
8426 F1                 POP     PSW             ;Clear A and flag register
8427 C1                 POP     B               ;Clear BC

8428 2A009A             LHLD    9A00H           ;Move contents of this
                                                ; location into HL
842B F9                 SPHL                    ;Now move it into SP
842C 210000             LXI     H,0000H         ;Clear HL


842F (8400)             END     BEGIN   ;Plus carriage return
```

```
8422   77        MOV     M,A          ; W
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8422 0102 0000 9000 00 0 1 1 1 0 91 99FA 9193 0306 0102 40D2
```

Notice that the Zero flag has been set. This happens when an operation causes the accumulator to go to 00H. We will clear the memory location by PUSHing this new value of A. Press BLUE:

```
8423   D5        PUSH    D            ; U
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8423 0102 0000 9000 00 0 1 1 1 0 00 99FA 9193 0306 0102 40D2
```

To finish up, we will clear a few more registers from the stack and restore IDA's stack. Press BLUE and watch this happen. When the first 'NOP' appears at program address 824F, you will be finished with this demonstration. Go ahead.

```
8424   D5        PUSH    D            ; U
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8424 0102 0000 9000 00 0 1 1 1 0 00 99F8 0000 9193 0306 0102
8425   D5        PUSH    D            ; U
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8425 0102 0000 9000 00 0 1 1 1 0 00 99F6 0000 0000 9193 0306
8426   F1        POP     PSW          ; q
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8426 0102 0000 9000 00 0 1 1 1 0 00 99F4 0000 0000 0000 9193
8427   C1        POP     B            ; A
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8427 0102 0000 9000 00 0 0 0 0 0 00 99F6 0000 0000 9193 0306
8428   2A009A    LHLD    9A00H        ; *@Z
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
8428 0000 0000 9000 00 0 0 0 0 0 00 99F8 0000 9193 0306 0102
842B   F9        SPHL                 ; y
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
842B 0000 0000 40D2 00 0 0 0 0 0 EA 99F8 0000 9193 0306 0102
842C   210000    LXI     H,0000H      ; !??
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
842C 0000 0000 40D2 00 0 0 0 0 0 EA 40D2 4CF8 C33E C532 3281
842F   00        NOP                  ; @
 PC  B C  D E  H L  A  C P A Z S M   SP  SP+0 SP+2 SP+4 SP+6
842F 0000 0000 0000 00 0 0 0 0 0 C3 40D2 4CFB C33E C532 3281
```

IDA will (I)nterpret in a variety of ways. In the format Iaaaa bbbb, bbbb is a check point at which there will be a register dump. This check point may be in ROM or in user memory, and may be placed anywhere in the program. If bbbb is omitted, the program will run until its end or a previously set check point is reached. Such check points (up to eight of them) may be entered, one at a time with the 'C' command (ex: C840A). IDA will print all of these screen displays to the printer if you preceed the I command with an 'L' (ex: LI8400 8400). (This is how I obtained the copy for use in this article.)

I know of no better way to study the 8080 and its operation than writing simple programs (directly into IDA using the 'O' command if you wish) and watching them perform with 'I'.

Another of IDA's special features lies in the 'U' command. 'UA8200 8600' will display all the ASCII bytes in the memory range specified, 'UB8300 867A' all the data bytes, and 'UW8477 A09A' all the data words in the byte range. You will be pleased with the usefulness of this feature. See the IDA manual for more details.

More on restoring directories.

In my last article, we laborously discussed the restoration of a damaged disk directory. I have ruined many directories in my time, including the directory to the ISC SAMPLER disk just yesterday. I was away from home and doing some relatively urgent work. It was very, very embarassing! The Good Spirit reminded me that I had a duplicate of the disk on the reverse side so repair was very easy.

I placed the good disk in the drive and loaded IDA. I then read the directory plus some file bytes into memory:

```
IDA>XREA 00 8200 200
```

Now I put the bad disk in the drive and typed:

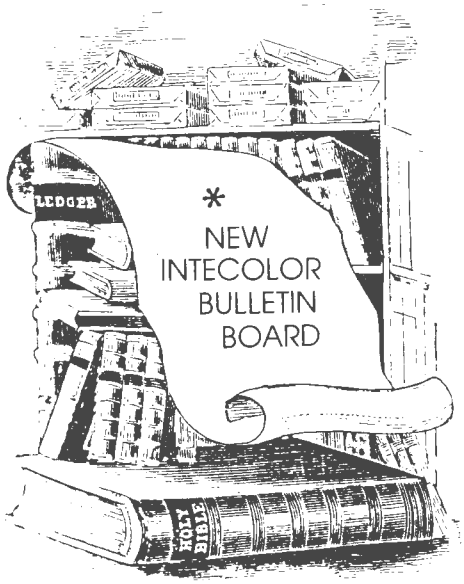
```
IDA>XWRI 00 8200 200
```

My directory was restored! This only works if the two disks are identical.

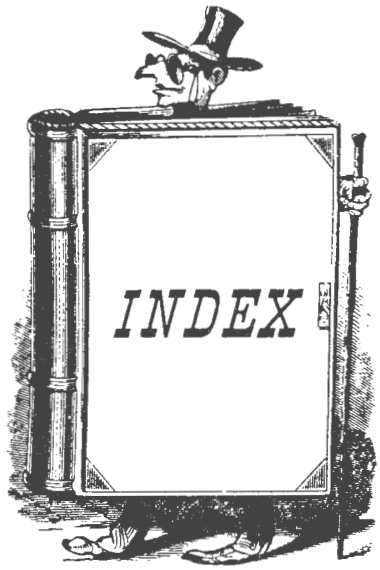
This completes our overview of IDA, the best software ever written for the CCII. We have not explored all the printing and editing facilities of IDA and I hope you will do this yourself. Hats off to Bill Greene. May he soon have IDA ready for CP/M and make some money!

[1] An interesting experiment for you. Try to determine the bit position of each of the flags by setting them in turn and examining the results of a PUSH PSW in IDA. The hex value shown on the stack will lead you to an exact determination of position for each of the flag bits. This chart shows the answers:

BIT NUMBER	0	1	2	3	4	5	6	7	
POSITION VALUE	1	2	4	8	16	32	64	128	
FLAG	C	x	P	x	A	x	Z	S	*



Intecolor Corporation has its own bulletin board operated by George Price. It is called *SPECTRUM* and the telephone number is 404-446-6931. The board operates at 300—1200 baud.



COLORCUE
CompUKolour
CUVIC
DATA CHIP

Prepared By Joseph Norris

The index in this final edition of COLORCUE covers published material from October 1978 through this issue of COLORCUE in August 1985 — eight years of user support. The following periodicals have been indexed in their respective time spans:

COLORCUE —October 1978 to August 1985, complete,
FORUM —March 1981 to the single issue of 1983, complete,
DATA CHIP —January 1979 to December 1984, complete,
CompUKolour —April 1982 to December 1984
CUVIC —January 1982 to June 1985.

It is not known if all the issues of the last two publications have been presented for indexing. If a few articles seem absent from the index I hope it will be because of this kind of omission.

Designing an index is somewhat of an art form. Realizing it, in any useful manner, is a most challenging occupation. As an intimate user of the CCII for five years I have relied heavily on my instincts to guide me in the selection of the index keys. 'Where would I look to find this?' has been my primary question as I read and catalogued the material. Unfortunately, my answer may not be yours in every case. I suggest that you scan the keywords on occasion to familiarize yourself with their pattern, and trust that this exercise will bear some fruit.

Most entries are made under more than one key. 'Printer', 'Handshake', and 'EPSON' are by their nature intertwined, for example, as are '50 pin bus' and 'interface.' All authors are listed by name with their articles following. Authors are also referenced in each entry, in parenthesis, where the authorship is known. (Some periodicals have been very careless about making authorship clear.)

Abbreviations are used as follows:

The first letter of the source for each listing refers to the publication, 'C' for COLORCUE, 'F' for FORUM, 'D' for DATA CHIP, 'K' for CompUKolour, and 'V' for CUVIC. This letter is followed in turn by the volume, number, and page, or a date of issue and page, as used by the several publications.

Several other abbreviations have been employed (somewhat inconsistently I fear) such as (A), meaning an assembly language program or routine, (B), meaning Basic code, (F), meaning Fortran, and so forth. 'Desc' means 'description.'

Page numbering is not always obvious in all periodicals. I have done the best I could under the circumstances. You may find several errors of a minor sort in this area.

The index is presented in a somewhat unusual fashion. FORUM and COLORCUE entries are grouped together in a single paragraph, as are CompUKolour and CUVIC. CHIP is in a paragraph by itself. This idiosyncracy is partially the result of a limited word processor capacity and the order of indexing, but also as a function of the likelihood of distribution of periodicals among readers in the United States and in other countries. At any rate, if you have only one of the periodicals, you may see, readily, what is available to you in your own library without reading the entire key listing.

There has been a considerable amount of reprinting and borrowing of articles amoung these publications. When this has been clear, duplicate entries for the same article will appear.

While COLORCUE will cease to exist with this issue, your access to the literature from which this index was derived will continue. If any subscriber wishes reprints of any article contained in this index, I will be pleased to provide them in XEROX form. The cost for each mailing will be $2.50 for the United States and Canada, and $5.00 in US funds for mailing outside the North American continent. Several articles may be included for the designated fee.

I send my thanks to the editors and subscribers among all the participants for their generousity and cooperation in this project. My special thanks to Wallace Rust who took valuable time to comment on an early version of the index and whose suggestions have improved it in great measure. I relieve him, however, from the burden of my errors which number, I am sure, as an early population of fishes in Tom Napier's 'watery world.'

Finally, I must add that I have been somewhat in awe of the monumental work performed by our prolific authors, whose identity becomes very apparent with the reading of the index. Their support and untiring dedication to us seems to me an extraordinary thing in this very commercial world. With their help we have been able to share in the joy and richness of the computer experience, and we have had the opportunity to sustain friendships of a high order over the years. May you find the index useful as you continue your exploration of the Compucolor II.

A

B

C

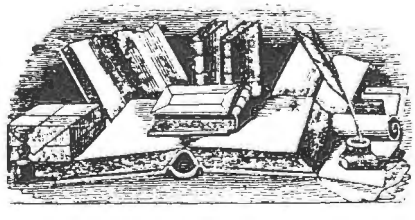
D

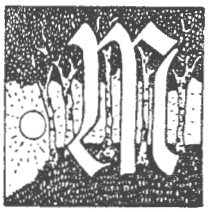
E

F

G

H





M

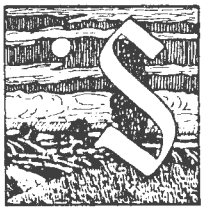
N

O

P

Q

R



S

Z

INDEX: Additions, Corrections & Notes.

COLORCUE 19 West Second Street • Moorestown, NJ 08057