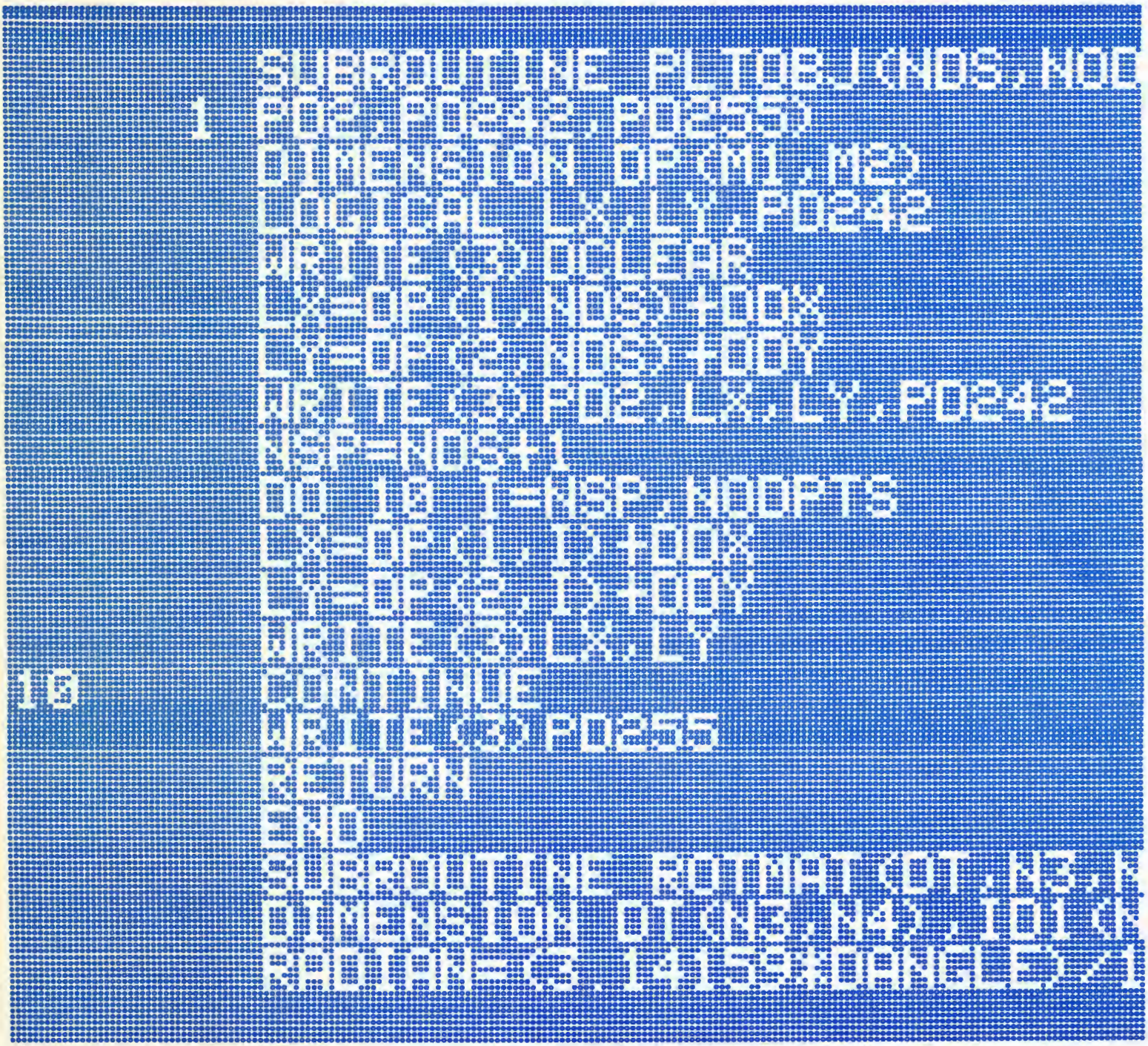
Colorcue



Aug/Sept 1982

$2.

Colorcue

A bi-monthly publication by and for Intecolor and Compucolor Users

August/September 1982
Volume 5, Number 1

Editors:
Ben Barlow
David B. Suits
Compuserve: 70045,1062

Advertisers: A good way to get in touch with potential customers is through the pages of **COLORCUE.** You will find our advertising policies attractive. Write for details.

Authors: This is a user-oriented and user-supported publication. Your articles, tips, hints, programs, etc. are required to make it go. Write or scribble your ideas down; we'll edit them and provide all artwork. Send your articles or write for information.

COLORCUE is published bi-monthly. Subscriptions are US$12/year in the U.S., Canada and Mexico, and US$24 (includes air mail postage) elsewhere. Some back issues are available. All editorial and subscription correspondence should be addressed to **COLORCUE**, 161 Brookside Dr., Rochester, NY 14618, USA. All articles in **COLORCUE** are checked for accuracy to the best of our ability, but they are NOT guaranteed to be error free.

Editors' Notes

Considering the ping pong history of **Colorcue**, I'm sure you're glad to see this issue arrive. We appologize for its lateness; what originally was a wait to determine whether or not renewals would justify continued publication turned into a wait to see just how many renewals we'd have to determine our press run, and then some experiments at reformatting failed, and it's just added up to more time than we thought. We'll try to get back on track.

Many thanks to all of you who resubscribed. Our spirits were flagging in midsummer, but renewals picked up, passed the number we felt we needed to continue, and, in fact, are still trickling in.

Late Breaking News!
(And Questions.)

Susan Sheridan, a past **Colorcue** editor many of you may remember, has given birth to a bouncing seven pound boy, her first child. Congratulations, Susan!

David B. Suits, a current **Colorcue** editor has recently returned from a stint in California, programming a game for Walt Disney Production's Epcot center in Orlando, Florida. Epcot will use ISC computers to control rides, as well as run games of the type David wrote, which visitors will be able to play as they stand in line. Look for it! It's a taxi-driving game, and the player "drives" a taxi across town by moving his finger over a touch sensitive screen.

The Source, a popular telecomputing company which until now has concentrated on the home market, is being forced by recent losses to shift its focus to business user, reports Data Communications magazine in its Sept. issue. It gives an interesting comparison of prices for the Source (23,000 subscribers) and Compuserve (28,000).

Vance Pinter is looking for information on how to connect and handshake with a Diablo 630 printer. Anyone with information, please drop Vance a note: P O Box 230, Columbus, GA 31902.

Andy Mau is interested in forming a user group in the NYC area. If you're interested in participating, write Andy at 5 Eldridge Street, Ground Floor, New York, NY 10002.

CUWEST, an active group in Australia, has told us of an excellent screen editor produced by Doug Pankhurst in another Australian user group. Called COLORTEXT, it's filled with features, like merging files, search and replace, search and delete, and more. For information, write to Doug Grant, CUWEST Librarian, 2 Brookside Ave., South Perth, Western Australia 6151.

ISC News: New products include CATS 80, a computer augmented training package, which allows a trainer to "...present material to a student in any combinations of text, color graphic displays (both static and animated) or with interactive audio and interactive video." Spectra-Text, the word proceesing package running on ISC's CP/M machines,is now available in a Spanish language version. ISC is making a big push with its Executive Presentation System, a package which can create graphic presentations for 35mm slides, overhead projector transparencies, or paper prints.

Vendor News

Jim Helms, 1121 Warbler, Kerrville, TX, 78028 (NOTE: This is his **NEW** address) has a host of programs available, and more coming. They're all written in assembly language, so they're quite fast. The list includes a Personal Database, Cross Reference Generator, General Ledger, Screen Editor, Assembler, Disk Editor, some games, and assorted other utilities. For several months now I have been using his screen editor/assembler combination which loads into my Devlin RAM board at 4000H. His programs have certainly made my assembly language programming more efficient! The assembler has some clever features (such as a printer driver and an excellent screen display during program

listing). The screen editor is top notch: it has all the functions of the ISC screen editor, plus some notable extras. You can merge files, copy blocks of text from one part of the document to another, and delete blocks. You can search and replace text. A status line at the bottom of the screen tells you how much memory you have left in RAM and on the disk, and tells you how large your file is. You can change colors and enter special characters. There are still more useful features to this editor, but I'll let you purchase your own so you can be pleasantly surprised. Write to Jim for his catalog, or contact an authorized distributor (Quality Software Associates, ICS, or Howard Rosen).

Advertising!!!

If you like to read advertisements, searching out the small but innovative companies offering products to the micro world, the Computer Shopper may be just the thing. It's a big tabloid style of publication chock full of ads for hardware, software, newsletters - everything. They also have articles and a user group bulletin board. More advertising than **Byte**!!

Mini Book Review

Several readers have written to say that while they enjoy the occasional hardware articles we publish, they wish we would run a hardware tutorial series to introduce them to electronics and digital logic.

As much as we'd like to do that, we have found a book that does a much better job than we could hope to do, and presents the material in an easy to understand, hands-on way that will serve the purpose very well.

The book is Digital Electronics - a Hands on Learning Approach by George Young, a professional electronics teacher who ran a series of articles in the first issues of Kilobaud Microcomputing. His approach is well thought out, his technique captivating, and his method is simple to follow. It won't make you an expert, but after completing the book and the experiments in it, you'll know enough to understand the articles you read here, and also those of Steve Ciarcia in BYTE. It's published in paperback by Hayden, copyright 1980.

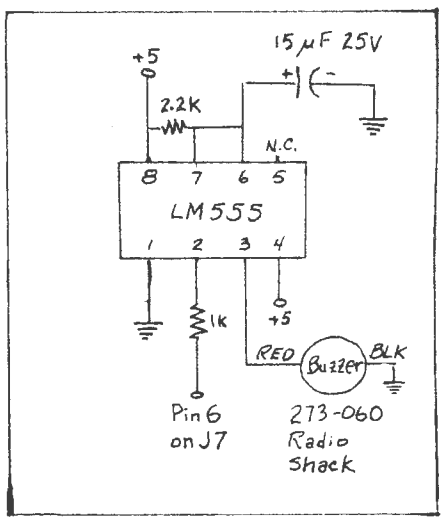
Moving?

If you're changing your address, please let both the Post Office and us know of your new address. (Tell us your old address and your new one.) We don't want you to miss a single issue of **Colorcue.**

Tech Tip

by David Zawislak
5739 N. California Ave.
Chicago, IL
60659

Another bell. I've been using this bell on my 6.78 CCII for over a year with no problems. Put the circuit on a small piece of perf board and mount it in the upper right corner of the back of the computer, where there are already some holes (for the old TV antenna). Put a 16 pin wire wrap socket in the disk connector socket, and plug the disk connector into that. The long wire wrap pins allow a length of 3-conductor ribbon cable from the perf board to connect easily. Get +5V from pin 9, GND from pin 14, and the bell trigger from pin 6. All parts are available at Radio Shack.



Reader Feedback

We have received a tremendous number of comments from renewing subscribers, and many, many requests for articles on a variety of topics. We've grouped these into major categories - we end up with nine - and present them below with some of the ideas. As you know, **Colorcue** is largely a reader-written publication; we're just the editors. But even our newest reader can write articles with this list as a guide. Our community has a wide range of interests, so whatever you write and share will help someone. Check this list, pick a topic, and write. We'll edit, so you can forget spelling, punctuation even. The list (no order):

1. Applications
   - Business applications, such as insurance or construction estimating
   - Scientific applications, esp. those of use in a high school science lab
   - Engineering, statistical, mathematical programs
   - Home record and budget keeping
   - Investment analysis, portfolio management
   - Hobby applications - HAM radio, horse racing

2. Tutorials - How do you do something, and why.
   - Assembly language - continue with this.
   - Novice level BASIC - use of primary BASIC statements, style
   - Guts - how does the computer work? Nits and grits of the chips, clocks
   - How to create source, PRG files step by step

3. Interfacing to existing devices through existing mechanisms
   - Printers - various type and idiosyncrasies
   - Modems - best type to use, whole area of communications
   - Other devices such as plotters, voice synthesizers

4. Software - utilities and useful routines as distinguished from applications
   - Languages available, benefits and use, routines of value
   - Utilities - in BASIC or any language, disk directories management, etc.
   - Subroutines - auto repeat keyboard routine, type-ahead, graphics

5. Hardware - Purchasable and buildable
   - How to use the 50-pin bus for A/D conversion, music, outside interface
   - Repair information for Compucolor owners

6. ISC Product related
   - ROM and RAM maps for software for various models
   - Availability, compatability and differences between models
   - List of hardware and software suppliers
   - List of and/or specific engineering changes

7. User Group corner
   - List of groups, what they are and do
   - User Group submitted articles (get your secretary going!)

8. Games (many requests for and against)
   - Adventure type game information
   - Puzzles and their solutions
   - Interesting BASIC games
   - Zork adaptations

9. Reviews
   - Books, games, packages, hardware, anything applicable

H O W A R D R O S E N, I N C.

P.O. Box 434
Huntingdon Valley, Pa.
19006
(215)-464-7145

* * * * BUSINESS SOFTWARE * * * *

LEDGER

Every business and home should have this program. LEDGER allows you to do a Receipt page, a Dispersment page, a Dues Collection List, a Budget, and any other form that you may have developed that uses rows & columns for numerical data storage with Titles. This easy and useful to use program allows 31 columns of data, and a 32nd column totals each row. There are 80 rows for each column and column totals. Intermediate row sub-total arithmetic is user defined. The arithmetic functions permitted are +,-,*,/,=. Saving, Loading, and Replacing data to the File Control System (FCS), Printing the Ledger sheet, and easy trial entries and changes make this a power-house. Requires 32K RAM and 117-key keyboard.

LEDGER disk includes **LEDGER, Instructions, PRINTER DRIVER, & Printer Driver Instructions.**

price 75.00

PERSONAL DATA BASE

PDB written in Assembly Language allows you to create a data base file consisting of data base records. Records are composed of a mix of literal and numerical fields as required. The records may then be used for statistical analysis, mail merge insertions for the mail merge word processor, data storage, retreival and sorting. Records may be added, changed, deleted, & searched. 32K holds 1200 records.

Personal Database II	price 85.00
Options:	
Plotting program - screen/printer	price 30.00
Distribution Analysis - Statistics	price 30.00
Encode/Decode Data/Hold Files	price 15.00
Math Option I - (+,-,*,/)	price 15.00
Math Option II - ($,+,-)	price 15.00
Form Processing	price 35.00
Left/Right Justification	price 10.00
Mail Merge Insertion	price 20.00

NOTE: PERSONAL DATA BASE and any 4 options priced at 10% discount.

EXECUTIVE WORD PROCESSOR price 299.00
MAIL MERGE WORD PROCESSOR price 349.00

FORTRAN Programming

by Howard Rosen
P.O. Box 434
Huntingdon Valley, PA 19006
(215) 464-7145

Instead of a series of lectures on the FORTRAN language, the approach will be to simulate sitting at a Compucolor II (CCII) and writing short, but executable, FORTRAN programs.

The first step will be writing the source code (the program) with the EDITOR (an ASCII editor such as the SCREEN EDITOR or the TEXT EDITOR, but not FREDI; he's strictly for BASIC). Since FORTRAN code must start in column 7 or greater, the TAB key will be pressed prior to writing any line of code except for statement labels which appear anywhere in columns 1 through 5 and are numbers. A "C" in column 1 allows that line to be a non-executable comment line and any character in column 6 is for the purpose of making that line a continuation of the previous line. Our first program will simply write a line to the screen. I'm going to use the SCREEN EDITOR for writing my program. Remember, you must have the FORTRAN disk to compile, link, and attach the library. More about that later.

Load the editor and reply to the prompt with:

```
FOR01.FOR
```

The file for a source code named FOR01.FOR has now been initiated, but nothing has been written to the disk, yet. Let us begin. Remember to TAB.

```
        WRITE(3,10000)
        DO 1000 I = 1,10
        INX = INX + I
        WRITE(3,11000)INX,DINX
1000    CONTINUE
        WRITE(3,12000)
        STOP
10000   FORMAT(' THE INTEGER AND FLOATING POINT VALUES
       1APPEAR BELOW'/10X,'INTEGER',8X,'FLOATING POINT'/)
11000    FORMAT(11X,I5,14X,F6.2)
12000   FORMAT('0   YOU HAVE SEEN THE CCII IN FORTRAN')
        END
```

The above progam represents a very simple but direct approach to starting in FORTRAN. If you feel you've made no errors, then save the program by pressing the FN1 key.

Next, the source program (FOR01.FOR which you just created) must be compiled. If you have one disk drive, remove the source disk and insert the FORTRAN compiler disk. If you have two disk drives, insert the compiler in driver CD1:

```
ONE DRIVE                TWO DRIVES

FCS>RUN F80              FCS>RUN 1:F80
Remove compiler disk     Leave disks in drives
Replace program disk
F80>FOR01                F80>FOR01
F80>control C            F80>control C

Remove program disk      Leave disks in drives
Insert compiler disk
FCS>RUN L80              FCS>RUN 1:L80
L80>FOR01                L80>FOR01
Remove program disk      Remove compiler disk
Insert Library disk      Insert library disk
L80>FORLIB/S             L80>1:FORLIB/s

(Relax. This will take several minutes.)

Remove library disk      Leave disks in drives
Insert program disk
L80>FOR01/N              L80>FOR01/N
L80>/E                   L80>/E
```

At this time the program (all linked) is being written to disk. When finished, get the disk directory and notice that FOR01.FOR, FOR01.REL and FOR01.PRG are all on disk. **FCS>**RUN FOR01 will execute your program.

There is another way to link and execute a FORTRAN program. There is a relocatable element called EQ.REL on the FORTRAN compiler/linker disk. With EQ.REL, the absoute element will use less space. Follow the steps below to experiment with EQ.REL.

```
Insert compiler disk     Insert compiler disk in DC1
FCS>RUN L80              FCS>RUN 1:L80
L80>/P:AF00              L80>/P:AF00
Insert program disk
L80>FOR01                L80>FOR01
Insert compiler disk
L80>EQ                   L80>1:EQ
Insert program disk
L80>FOR01/N/E            L80>FOR01/N/E
Insert library disk      Insert library disk in CD1
FCS>RUN LIB              FCS>RUN 1:LIB
Insert program disk
FCS>RUN FOR01            FCS>RUN FOR01
```

The PLOT command from BASIC is not available, but it can be simulated by declaring a type LOGICAL or BYTE in your FORTRAN Program.

```
BYTE RED,GREEN,YELLOW,BLUE,CR,LF,ERASE
DATA RED,GREEN,YELLOW,BLUE/17,18,19/
DATA CR,LF,ERASE/13,10,12/
```

Now write to the logical unit 3 (LUN #3), the screen, with a non-FORMATted write statement, e.g.:

```
WRITE(3)ERASE,YELLOW
```

That statement will erase the screen and prepare for the screen display to be in yellow.

Plot 3D Figures with FORTRAN 80

by Doug Van Putte
18 Cross Bow Drive
Rochester, NY 14624

FORTRAN 80 is a superior language for writing 3D graphics programs for the Compucolor or Intecolor because it handles the math operations very effectively. When Howard Rosen read my article "3D Graphics" in the Feb/Mar issue of COLORCUE he called to suggest that FORTRAN programming was an excellent way to improve the speed of the many math operations required to move objects around the screen. Thanks to Howard, I accepted the challenge of converting the primary 3D operation of rotation of an object to a structured FORTRAN program.

The first hurdle to pass was to understand the method of plotting to the screen. While BASIC has the convenient PLOT statement, screen graphics in FORTRAN 80 require the use of LOGICAL variables. After the chosen variables are defined by the LOGICAL statement, they can be assigned the required plot values identical to the familiar BASIC PLOT values. Plotting is then achieved by the unformatted WRITE(LU) AI statement, where LU stands for the device logical unit, and AI are the plot variables. For example, to plot a point, x,y, consider the little program in **Listing 1.**

Listing 1

```
        PROGRAM PLOTXY
C DEFINE VARIABLES AS LOGICAL
        LOGICAL P2,LX,LY,P255
C ASSIGN VARIABLES PLOT VALUES
        P2 = 2
        P255 = 255
        LX = 40
        LY = 50
C PLOT THE POINT X,Y ON THE SCREEN
        WRITE(3) P2,LX,LY,P255
        END
```

The value of (3) for the Logical Unit is the device number of the console. So, the unformatted WRITE is the statement that the programmer uses to convey values directly to the memory, just like BASIC's PLOT statement.

This all seems somewhat awkward, doesn't it? Keep in mind that you end up with fast, compiled PRG programs or subprograms which can solve complex problems with double precision. The subprograms can become entries in a personal library which can be linked and run with a main program at any time. In addition, a large library of math functions are at the programmer's disposal. Another strong advantage is that the source code, "as is", can probably be compiled on just about any micro which supports FORTRAN 80. This should give the program entrepreneur a broader market for his programs and the venerable Compucolor owner some comfort that his source programs will be useable on his next machine.

Now consider the concept of a 3D plotting program which draws a box and rotates it in 10 degree increments sequentially about all three axes, changing its color on each rotation. The functions in flow chart form are given in **Figure 1.**

Since I wanted to use subprograms to perform the functions in the boxes, the next learning experience was understanding how to communicate between the main program and a subprogram. Any variable that is passed to and from a subprogram must be identified in the CALL statement and in the SUBROUTINE definition statement. The main program variable names are used in the CALL statement, while dummy variable names are

used in the SUBROUTINE statement. Recognize, however, that the forms of the variable lists in both statements must be identical. The dummy names in the SUBROUTINE statement are changed by FORTRAN at execution time so things come out right. This concept allows a subprogram to be used with any main program with the proper CALL. Also, in the CALL variable list the actual dimensions of a main program array must be present following the array name in the list.

Several other hard-learned rules are related to dimensioning and to the definition of variables. The first rule is that arrays and dummy arrays must be dimensioned where they are used in both the main program and the subprogram. The last rule which was troublesome is that specially defined variables must be defined locally where they are used also. There is no "global" variable concept in FORTRAN 80. Further development of these ideas plus much more can be found in Microsoft's FORTRAN 80 manual.

The FORTRAN 80 source program which demonstrates the above features is given in **Listing 2.** After the data is loaded by the program, the four main functions are performed by the general subprograms. The equations in the subprograms are written as explicitly as possible to eliminate the necessity for DO loops which slow down the execution. If you have FORTRAN 80, type in the program with an editor, then compile, link and run the PRG program and watch the results. Next, put in your own 3D figure in array P, your own value of "NOPTS", and repeat.

The subprograms can be used in any similar program with some limitations, as follows:

1. PLTOBJ -- An object with any number of connected points can be drawn. If your object contains some figures which are not connected, you will need to CALL the subprogram for each figure.

2. INIMAT -- This routine simply initializes the 3*3 transform matrix.

3. ROTMAT -- This establishes the element values for the 3*3 rotation matrix, depending upon the axis of rotation and the rotation angle.
4. MLTMAT -- This multiplies the 3*3 rotation matrix with each object point to compute the new coordinates of the object.

Frankly, I like the prospects of using FORTRAN on my Compucolor for the advantages I have stated. But beyond that, I was schooled and experienced in FORTRAN long before BASIC and I'd forgotten how comfortable it feels. Are there any FORTRAN 80 people out there besides Howard and myself? [Yes! There's more to come next issue. -- eds.] As Howard said to me, "Try FORTRAN 80. You'll like it!"

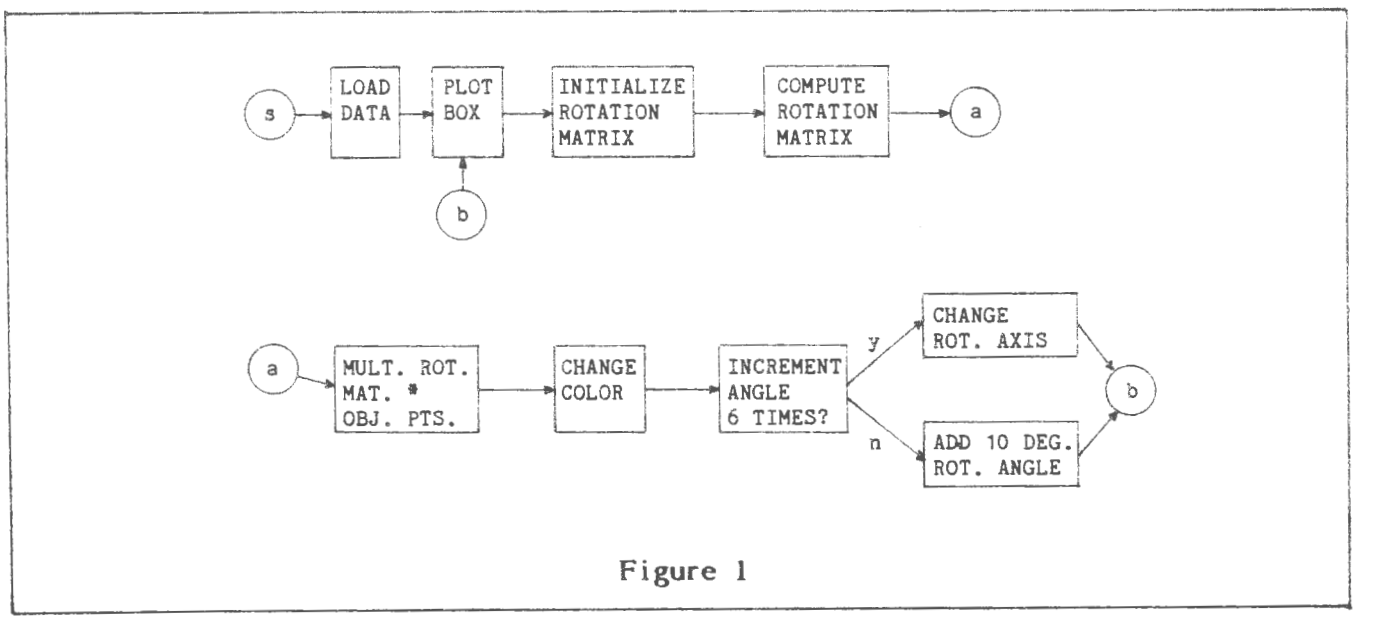


Figure 1

Listing 2

```
      PROGRAM PLOT3D
      DIMENSION P(3,18),G(3,18),T(3,3),I1(3),I2(3),
     1 DEG(3),TI(3,3)
      LOGICAL CLEAR,COLOR,P2,P242,P255
      DATA  P/0.0,0.0,0.0,30.0,0.0,0.0,30.0,30.0,0.0,
     1 30.0,30.0,30.0,30.0,30.0,0.0,0.0,30.0,0.0,
     2 0.0,0.0,0.0,0.0,0.0,30.0,0.0,30.0,30.0,
     3 30.0,30.0,30.0,0.0,30.0,30.0,0.0,30.0,0.0,
     4 0.0,0.0,0.0,0.0,0.0,30.0,30.0,0.0,30.0,
     5 30.0,30.0,30.0,30.0,0.0,30.0,30.0,0.0,0.0/
     6 I1,I2/1,1,2,2,3,3/
     7 TI/1.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0/
     8 IB,NOPTS,NS,ANGLE,DX,DY/16,18,1,10.0,60.0,60.0/
     9 P2,P242,P255,CLEAR/2,242,255,12/
      CALL PLTOBJ (NS,NOPTS,P,3,18,CLEAR,DX,DY,P2,P242,P255)
1     DO 30 IR=1,3
      DO 10 JR=1,6
      COLOR=IB+JR
      WRITE(3)COLOR
      CALL INIMAT (T,3,3,TI,3,3)
      CALL ROTMAT (T,3,3,I1,3,I2,3,IR,ANGLE)
      CALL MLTMAT (NOPTS,P,3,18,T,3,3)
      CALL PLTOBJ (NS,NOPTS,P,3,18,CLEAR,DX,DY,P2,P242,P255)
30    CONTINUE
      GO TO 1
      END
      SUBROUTINE PLTOBJ(NDS,NODPTS,DP,M1,M2,DCLEAR,DDX,DDY,
     1 PD2,PD242,PD255)
      DIMENSION DP(M1,M2)
      LOGICAL LX,LY,PD2,PD242,PD255,DCLEAR
      WRITE(3)DCLEAR
      LX=DP(1,NDS)+DDX
      LY=DP(2,NDS)+DDY
      WRITE(3)PD2,LX,LY,PD242
      NSP=NDS+1
      DO 10 I=NSP,NODPTS
      LX=DP(1,I)+DDX
      LY=DP(2,I)+DDY
      WRITE(3)LX,LY
10    CONTINUE
      WRITE(3)PD255
      RETURN
      END
      SUBROUTINE ROTMAT(DT,N3,N4,ID1,N5,ID2,N6,II,DANGLE)
      DIMENSION DT(N3,N4),ID1(N5),ID2(N6)
      RADIAN=(3.14159*DANGLE)/180.0
      K=ID1(II)
      J=ID2(II)
      DT(K,K)=COS(RADIAN)
      DT(K,J)=-SIN(RADIAN)
      DT(J,K)=SIN(RADIAN)
      DT(J,J)=COS(RADIAN)
```

```
      RETURN
      END
      SUBROUTINE INIMAT (DT,M1,M2,DTI,M3,M4)
      DIMENSION DT(M1,M2),DTI(M3,M4)
      DO 10 I=1,M1
      DO 10 J=1,M4
10    DT(I,J)=DTI(I,J)
      RETURN
      END
      SUBROUTINE MLTMAT (NODPTS,DP,M1,M2,DT,M3,M4)
      DIMENSION DP(M1,M2),DT(M3,M4)
      DO 10 I=1,NODPTS
      PX=DP(1,I)
      PY=DP(2,I)
      PZ=DP(3,I)
      DP(1,I)=PX*DT(1,1)+PY*DT(1,2)+PZ*DT(1,3)
      DP(2,I)=PX*DT(2,1)+PY*DT(2,2)+PZ*DT(2,3)
      DP(3,I)=PX*DT(3,1)+PY*DT(3,2)+PZ*DT(3,3)
10    CONTINUE
      RETURN
      END
```

A Music Tutorial Using the Compucolor II and Soundware™

by D. B. Grant
2 Brookside Avenue
South Perth 6151
Western Australia

(Reprinted by permission from the CUWEST users group newsletter.)

Music is made up of notes of varying sounds and varying lengths in varying arrangements, and that is about all that we need to know.

If you look at the sheet of music you have chosen for your first "masterpiece", you will notice that it has two sets of 5 lines. One set, correctly called the Treble Staff, is the only one in which we are interested. The other staff is called the Bass Staff.

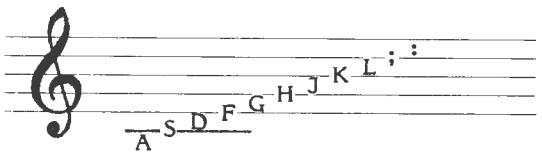


Figure 1

The large S type figure on the Treble Staff is a Clef, but more importantly you will notice that it cuts through one line four times. This line, for our purposes only, is called H, and you must fix that firmly in your mind while you are writing music for your computer. As soon as you finish with the computer you must forget "H", because there is no such note in music. (There will be other things which we will use that will apply only to us, so watch who you air your musical knowledge with.)

H is the middle key on the second line of the keyboard, and, with the addition of a "ledger line", we can cheat again and make H the middle of our musical range of notes. 'S' is on our "ledger line" and is normally called MIDDLE C in all other music.

If possible, the music you wish to compose should come within the range "A" to ":". These are our "white keys". A piano keyboard has a line of white keys plus a lesser number of black keys in groups of 2s and 3s. Our equivalents for the black keys are E R Y U I P @.

If you are able to play the piano "by ear", you could now go ahead and compose. For the rest of you...read on.

With your Soundware unit plugged into the modem port at the back of the computer, insert the Soundware disk and hit AUTO to bring up the MENU, then hit M for Music Composer. After the Music Composer display comes up on the screen, it is best to wait a few moments before starting work. Read the list of commands to make life easier later on.

The keys are all named for you, but more theory is needed at this stage.

Back to the music. We have all the white keys named on our staff, but not the black keys. The black keys are used when you see an odd looking "b" or "#" on the line or in the space. The "b" means that the original note is to be "flat", and the "#" means that the original note is to be "sharp". For instance, if on the line which is our "D" line there is also a "b", we must hit "E" instead of "D". If we had a "#" on the "D" line, we would need to hit "R". Try these three keys and note the difference in sound. If the "b" or "#" is placed at the beginning of your

piece of music right near the Treble Clef sign, then every "D" in the piece of music would be "flat" or "sharp", as would any other note on whose line or in whose space these signs were placed.

However, there are exceptions to this rule, and you'd best know about them. If a note has been "flat" or "sharp", and you then see a " ♮ " sign near one of the notes you are to play, then--and only then--that note reverts to the original note until the next BAR LINE, unless told otherwise.

What's a Bar Line? This is a vertical line from top to bottom of the staff, and there are lots of them. If a note has not been flat or sharp from the beginning of the music and you suddenly came across a flat or sharp sign, then that also remains in force only until the next bar line, unless told otherwise. This may sound a little strange, but so will the music if you don't get it right.

Now let's go on to the varying lengths part of the notes. By a very basic code a musician knows the duration of any note he is to play, and we can easily learn this code.

An outline of an egg (on its side) is called a whole note:

Hang a tail on this egg and it is halved in duration (a half note):

Fill in the hole and we have a quarter note:

Hang a flag on the tail and, yes, a one eighth note:

Another flag makes it halve again:

All these code signs have names, but who cares? We don't! Look at a piece of music and see how many types you can find. They won't all be there, thank Heaven; some peices of music use only one or two of them, and, with a bit of luck, you'll have chosen one such piece.

One last point on note duration is that a dot right near the note means that the duration of the note is increased by 50%, and you may as well know the reason for this too. The Bar Lines, as well as telling you about changes in sounds (when necessary) also tell you about the RHYTHM of the piece. Have a look at the very beginning of the music again and you will see two numbers, for example: 4/4 3/4 2/4 6/8, or just a large "C". The bottom number tells you that the basic note is a quarter note if it is a 4, or an eighth note if it is 8. The top number tells you how many such basic notes are to be between any two Bar Lines, or the equivalent of this number of basic notes. There may be 4 quarter notes, 1 whole note, 2 quarters plus 4 eighths, or any other combination, but they must add up to the same each time, between any two bar lines. The large "C" at the start is the same as 4/4. Add up the notes in a few different Bars (the space between two barlines) and you will usually get the same answer ... with a few exceptions; so sorry about this.

The exceptions are that, often in the gap between the Clef Sign and the first bar line there are not enough notes to add up to the required number, but if you look at the very last bar you will most likely find the balance... with a few exceptions! These last exceptions (hooray!) are the RESTS.

 = an eighth rest

 = a quarter rest

 = a half rest

 = a whole rest

Okay, NOW add 'em up and they will always, always be right, and you must keep them that way in your composing too. If you don't you will not get the results you would prefer. Don't forget the fact that a dot after a note increases the duration of that note by 50%.

If you are still keen to compose, then we will begin.

Hit ERASE PAGE then SPACE BAR to clear all those notes you tried out earlier,

then hit 8 followed by S. You will hear a long note. Now try all the other numbers 1 to 9, in any order, to hear the different lengths available, for any note.

Hit Right Arrow and you will hear a replay of all the notes you have entered.

Hit ERASE PAGE again if you tried out any additional notes and you will be left with only the S. Hit 8 then D F G H J K L followed by Right Arrow and you will hear a scale of long notes.

Hit L then 4 K J H G F D S S (again) 2 D F G H J K L.

Now try Right Arrow to hear the tune you have entered.

You will hear the scale in three versions; first with long notes, then with notes half as long, and then half again. You can replay these scales as many times as you wish by hitting Right Arrow after each replay. When you've digested what you have entered, hit ERASE PAGE to clear it all, except the very first note ... in this case S.

The only way you will get rid of that first note is by hitting the Space Bar. This only applies to the first note of any tune.

If you make an error while entering a tune, you delete the last note by hitting the Left Arrow, and you can keep on going back deleting notes one at a time with Left Arrow. (And the Right Arrow will NOT restore them.)

Note that when you set the length of a note on your scales, all the notes that followed were also at that length until you changed the length.

Before you start on your tune you must decide just what length you are going to make your shortest note. Glance through the piece and find the shortest note. If in a 3/4 tune (waltz) the shortest note is a 𝅗𝅥 then make it a 2 and you will find that you will be using 4 and 6 for 𝅗𝅥 and 𝅗𝅥. . If you come upon an eighth note 𝅘𝅥𝅮 then you can use 1 for that note. If you had used 4 for your shortest note then you would be in for trouble when you came to 𝅗𝅥. as it would need to be 12 long, and you can't use a length over 9 (or under 1).

Look at your first note and compare its position to Figure 1. Hit the appropriate key, followed by the length number you need. Go on to your next note and do the same. If this second note is to be the same length as the first note then you don't have to hit a length number at all; you only have to do that when you get a change in length, and then you hit the new length number AFTER the note you wish to change as this will then change the length of that note and all subsequent notes until you change it again. If you hit a wrong length number then you only need to hit the right length number to alter it; it will change only the length of the last note entered.

Hit Right Arrow often to hear how you are going. Left Arrow deletes the last note only, including rests. Oh yes. Rests. If you come to a sign denoting a rest, then you enter it at its correct length by hitting Space Bar followed by the length of the rest.

The tune is the thing that all this is about, and when you have passed your own examination then you save the tune by hitting B. You will be asked for a name for your tune (less than 7 letters), and after a pause your music will be saved. You hit C to have any music loaded from disk, then Right Arrow to hear it played.

It's a good idea to write down the names of the tunes you save, as this saves you the bother of going to the Directory to find out.

Lots of music to you.

Tech Tip

By David B. Suits

The space bar on the ISC keyboard is notoriously stiff. The stiffness comes not from the bar itself, but from the strong spring in the switch. Unsoldering the space bar switch and swapping it with another (e.g., the CPU reset switch) is an ideal and inexpensive solution.

Assembly Language Programming

by Ben Barlow

Part VII:
M80 - the Macro Assembler

The Macro Assembler can save you time writing and debugging programs, and it can save space on your disks. It's a little tougher to use than the regular Assembler, but the benefits are well worth the learning if you do any kind of serious assembly language work. This small tutorial on M80, the Macro Assembler, will explain the benefits, tell you how to use the package itself, and explain how to take advantage of the major features that M80 offers. First, let's examine the benefits.

Modularity

M80 allows you to break your program into functions (a fancy term for rational pieces), and separately code and test each function independently. L80, the Linkage Editor supplied with the M80 assembler will combine those separate functions for you into a complete program. The smaller chunks of code are easier to edit, they take less disk space, and best of all, if you define them properly, they are easily reused - without reassembly. So you save time in the edit-assemble-test phase, you save disk space, and you save time writing new programs when you can reuse existing functions.

Some Definitions

What? Wait a minute. What is a "Linkage Editor", a function - a "rational piece"? OK. A little definition may be in order. Starting with the source file, a text file of assembly language instructions constructed with an editor of some sort or another (the screen editor or its variants are good choices), an assembler (the Assembler or M80) reads the source file and translates it into object code which it places into a file. Object code is not quite machine code, but is much closer than the original source. The Assembler produces an object file of type LDA which can subsequently be LOADed and RUN by FCS; but M80 produces a **REL**ocatable object module. The REL (for short) file cannot be LOADed and RUN by FCS, because FCS does not understand its format. Being "relocatable" means that the object code in the module can be placed anywhere in memory. M80 does not generate absolute addresses, but instead produces "offsets" to a zero origin. (An origin **can** be specified, but the object module produced is not then relocatable, and many advantages are lost. Unless necessary, don't put ORG statements into your M80 source files.) The REL file must be processed by L80, the Linkage Editor before it can be used. The linkage editor takes the REL file (and possibly other previously assembled REL files) and combines them, relocates them (which means making all their address references fixed), and creates a file of type PRG, which can subsequently be RUN by FCS. Sound complicated? I'd be lying if I said it wasn't, and you wouldn't believe me anyway. The User's Manual is good for reference, but useless for training. It does say, however, that it "...is not intended to serve as instructional material, and presumes the user has substantial knowledge of assembly language programming." Some perseverance, a little experimentation, and careful reading should get you off the ground, though.

Reusability

Being able to combine previous work with

new work is probably the biggest benefit of the whole package. As an example, you could design a small routine (function) to read a joystick input and return a value. Once written and tested (probably with a small driver, or test program) the function can be used over and over simply by linking it in with the other parts of the program. You know it works, and you know how to use it. A set of screen handling routines, or the Sort routine published in a past **Colorcue** are other examples of reusable functions. Once done and working, they stay that way, and you don't need to spend time retyping or editing or assembling them. Your programming style has probably benefitted by your trying to decompose the overall problem into a set of independent functions. So much for modularity.

Macros

The second big benefit of M80 is the ability to use macros in your source code. Like the REL files which perform specific functions and are reusable, macros define functions in source language and permit their reuse, with the same benefits as modularity. Once developed, a macro can be reused, you know it will generate the right code, and you save coding and debugging time. A macro is a named set of source statements appearing at the front of your source file (or sucked in from a collection of them on a disk, called a macro library) which M80 will substitute for a call upon that macro - a reference to its name. A macro is really a sort of shorthand. Its definition by name at the front of the source says to M80, in effect, "Here's a set of assembly language instructions named ⟨whatever⟩. As you read along and come to the op-code ⟨whatever⟩, substitute these instructions." That seems easy enough. Things get a little more complicated because the macro can have arguments, and can do some limited testing of those arguments during the generation steps to modify the code produced, but we'll get into that later. (The concept of macros is not unique to assembly languages. Many high level languages derive a lot of power from their use. PL/1 and C come immediately to mind.) Macros can save a lot of repetitive coding, and can reduce errors by simplifying the instruction set.

Conditional assembly

The ability to conditionally assemble statements can be a big help to writers of general purpose programs, or to writers of fancy macros. They allow a macro to be tailored more closely to the situation in which it is called, by generating different code depending on the presence or absence, or the type of some of its arguments.

Examples

Let's look now at some examples. We'll develop them from the assembly language program on page 16 of the June/July 1982 **Colorcue.** These programs, which colored the entire screen, will be shown in listings below, so if you've wrapped fish in the June/July issue, don't despair. First, we'll develop two useful macros, to save registers at the entry to a routine, and to pop them back at its conclusion. Here is their definition:

```
ENTER   MACRO               ;begin the definition of ENTER
        PUSH    H           ;save h,l (first instr to be genn'ed)
        PUSH    D           ;second instruction
        PUSH    B           ;third
        PUSH    PSW         ;fourth
        ENDM                ;end the definition

EXIT    MACRO               ;begin the definition of EXIT
        POP     PSW
        POP     B
        POP     D
        POP     H
        RET                 ;these will be generated when EXIT is used
        ENDM                ;end the definition
```

Now, let's use the macros and see what happens. **Listing 1** shows the areas of the program where the macros are defined and used. M80 puts a + sign before each instruction generated by a macro. Note that while we code only one line (ENTER or EXIT), M80 generates four or five instructions. That's a productivity multiplier, and you won't have to remember any longer in what order you put things on the stack. **But** (there's always one of those), we don't have quite what we want. Although our ENTER and EXIT macros generate code, it's not quite the code we want. When called by FCS, as the first routine in the example is, we must return a value in the B register as an error indication. If we don't, FCS will display a red error message of some sort on the screen after RUNning our program. So, we've got to change the EXIT macro a bit to accept a return code for the B register. We'll do this with conditional assembly:

```
EXIT    MACRO   RVAL            ;begin the definition (this macro must
                                ;replace our old EXIT macro)
        POP     PSW
        POP     B
        POP     D
        POP     H               ;do the pops as before
        IFNB    <RVAL>          ;test to see if RVAL used in macro call
          MVI     B,RVAL        ;if it was, put RVAL value into B
        ENDIF                   ;if RVAL was not present in call, MVI
                                ;will not be generated.
        ENDM
```

We can use this macro in the first routine now, which is called by FCS. Simply replace the macro call we had put in as:

```
        EXIT            ;and return to FCS
```

with:

```
        EXIT    0       ;and return to FCS
```

RVAL, as coded on our macro's definition line, is an argument. When the macro is used, as we did immediately above with EXIT 0, we can include or omit this argument. If included, the macro will generate a

```
        MVI     B,RVAL
```

instruction, and replace RVAL with whatever we code on the call line. With EXIT 0, it would generate

```
        MVI     B,0
```

as you might expect. The macro can also test to see if an argument was given on the call line at all. That's what the IFNB and matching ENDIF do. IFNB stands for IF Not Blank, and means, "if an RVAL argument was specified, generate everything between here and the matching ENDIF. M80 also provides other tests (e.g., IFB - IF Blank) and we can put many arguments in our macro definition, not just one; but one illustrates the use.

Look at the generated code in **Listing 2.** You will quickly note that the POP of B in the first routine is somewhat useless, when we come along and change it two lines later. Assuming that C, which is part of the register pair BC, isn't needed, that's right. We could have tested the RVAL argument before doing the POP B, and avoided it. If we were to do that, though, ENTER would also have to be changed too, so it did not PUSH B. Then we'd have to match ENTER types and EXIT types, and we're trying to simplify, not complicate. So, at the expense of a couple of bytes of unneeded code, we've got generality. (If you're unwilling to spend the two bytes, and want to write the tightest possible code, you're probably not interested in coding and debugging speed.)

To complete the example, we'll put the macro definitions into a library, separate the program into two modules, and assemble each independently. A little absurd, given their size, but a reasonable example. Their listings are shown in **Listing 2.** (Even though the first routine, SETUP.MAC is small, it is general purpose enough to use with any programs you want to link into BASIC that are entered through the CALL vector.)

```
                ;SYSTEM EQUATES - ADDRESSES OF THINGS WE'LL NEED
                CALLVEC EQU     33282   ;ADDRESS OF VECTOR FOR CAL
                TOPMEM  EQU     32940   ;ADDRESS OF TOP OF MEMORY
                SCREEN  EQU     7000H   ;ADDRESS OF SCREEN MEMORY

                ENTER   MACRO
                        PUSH    H       ;BEGIN DEFINITION
                        PUSH    D
                        PUSH    B
                        PUSH    PSW
                        ENDM            ;END DEFINITON

                EXIT    MACRO           ;BEGIN DEFINITION
                        POP     PSW
                        POP     B
                        POP     D
                        POP     H
                        RET
                        ENDM            ;END DEFINITION

                START:  ENTER           ;MACRO: SAVE ALL REGS
+                       PUSH    H       ;BEGIN DEFINITION
+                       PUSH    D
+                       PUSH    B
+                       PUSH    PSW
                                        OF LAST BYTE
                                        MEMORY,
                        TOPMEM  ;AND PUT INTO BASIC'S POINT

                        EXIT            ;MACRO: POP REGS AND RET.
+                       POP     PSW
+                       POP     B
+                       POP     D
+                       POP     H
+                       RET

                CALL:   ENTER           ;MACRO TO SAVE REGS
+                       PUSH    H       ;BEGIN DEFINITION
+                       PUSH    D
+                       PUSH    B
+                       PUSH    PSW
                                        TOO
                        LXI             RANGE.

                        EXIT            ;MACRO: POP REGS AND EXIT
+                       POP     PSW
+                       POP     B
+                       POP     D
+                       POP     H
+                       RET
                        END     START
```

```
0000'
0000'   E5          +
0001'   D5          +
0002'   C5          +
0003'   F5          +

0004'   3E C3

0006'   32 8202
0009'   21 001C'
000C'   22 8203

000F'   21 FFFF'

0012'   22 80AC


0015'   F1          +
0016'   C1          +
0017'   D1          +
0018'   E1          +
0019'   06 00       +
001B'   C9          +

001C'
001C'   E5          +
001D'   D5          +
001E'   C5          +
001F'   F5          +

0020'   21 7001

0023'   73

0024'   23
0025'   23

0026'   7C
0027'   E6 F0
0029'   FE 70
002B'   CA 0023'


002E'   F1          +
002F'   C1          +
0030'   D1          +
0031'   E1          +
0032'   C9          +
```

```
START:  ENTER
        PUSH    H
        PUSH    D
        PUSH    B
        PUSH    PSW

        MVI     A,(JMP)
;
        STA     CALLVEC
        LXI     H,CALL
        SHLD    CALLVEC+1
;
        LXI     H,START-1
;
        SHLD    TOPMEM

        EXIT    0
        POP     PSW
        POP     B
        POP     D
        POP     H
          MVI     B,0
        RET

CALL:   ENTER
        PUSH    H
        PUSH    D
        PUSH    B
        PUSH    PSW

        LXI     H,SCREEN+1

LOOP:   MOV     M,E

        INX     H
        INX     H

        MOV     A,H
        ANI     0F0H
        CPI     070H
        JZ      LOOP

        EXIT
        POP     PSW
        POP     B
        POP     D
        POP     H
        RET

        END     START
```

Listing 1

Macro Definitions and Use in Program

The Linkage Editor

In splitting the single program into two pieces we've created a problem for L80 to solve; namely that of linking the two sections into a single whole. In SETUP, M80 no longer "knows" the address of CALL, which has been relegated to the other program, COLSCR. So we tell M80 that CALL is really external to the SETUP source file, and not to worry. The mechanism for that is the EXTRN statement, which you can see in the listing. L80 will then know, when linking the object modules for SETUP and COLSCR, that locations in SETUP that refer to CALL must be replaced with CALL's actual address. How will L80 know CALL's address? We've got to tell it. We do that with the ENTRY statement you can see in COLSCR. For correct linkage, every EXTRN in a module must be paired with an ENTRY in some other module. (ENTRY's can be "left over" without harm.)

The basics of the M80 and L80 tools are now clear. (?) Further use, reading, and experimentation will make you an expert, and you'll find your coding time decreasing, and the number of projects you can tackle increasing. In addition, interesting macros or functions will make good topics for **Colorcue.**

Using the Tools

The one problem remaining is actually using the bloody things. The ISC-supplied documentation is so poor, that once you get a source file, it's difficult to figure out how to assemble and link it. Since it would be grossly unfair to stir up your interest so far, and then leave you stranded, let's look at the nitty-gritty of using M80 and L80.

Unfortunately, we haven't space to cover source program creation with the Editors. The documentation should help you through that, although as I remember, I never did master the line Editor that came on the Assembler disk. It's best to get a copy of a screen based editor if possible.

Once the source file is built, it's time to run it into M80. If you constructed the source file with a .MAC file type, M80 will be happy if you allow it to use a default. Otherwise, you'll have to specify the file type (e.g., .SRC) each time. M80 offers several options, and we'll cover the basic ones of **L** (list on printer) and **N** (don't make REL file). You can experiment with the others on your own. Let's go through the process step by step. (The computer's output is in bold type.)

```
FCS>RUN M80                 1
M80>SETUP/N                 2
M80>COLSCR/N                3
M80>(control C)             4
```

Step 1 simply runs M80 from your default disk. (if you have two drives, you will probably want one to have your editor, M80 and L80, and the other to have the source files and REL files.) Step 2 assembles the SETUP program, and Step 3 assembles the COLSCR program. For this first pass, we just want to check for errors, so we didn't specify list option, and we did specify the "no REL file" option. Errors, if there were any, would be listed on the screen. Step 4 quits M80. If your assemblies went cleanly, omit Step 4, and go on to Step 2 below. After editing the source file to remove the errors, let's go through the steps again and obtain both a listing and a REL file. (The printed output is wider than 80 columns, so set your printer up accordingly, or suffer the overlap if you can't.)

```
FCS>RUN M80                 1
M80>SETUP/L                 2
M80>COLSCR/L                3
M80>(control C)             4
```

Now look at your disk directory. You should see SETUP.REL and COLSCR.REL there. Your printer should have pages of printed listings. On to the Linkage Editor.

L80 has even more options than M80 did. Again, we will select only a few to illustrate the process and produce a working product, and let you extend beyond by yourselves. The sequence of commands to produce a .PRG file loaded at 9000H (hex) are shown:

```
FCS>L80                     1
L80>/P:9000                 2
L80>SETUP,COLSCR/M          3
L80>TEST/N                  4
L80>/E                      5
```

```
C               INCLUDE MACS    ;PULL IN MACRO DEFS AND EQUATE
C       ;MACRO LIBRARY AND SOME HANDY EQUATES.
C
C       ;SYSTEM EQUATES - ADDRESSES OF THINGS WE'LL NEED
C
C       CALLVEC  EQU     33282   ;ADDRESS OF VECTOR FOR CALL
C       TOPMEM   EQU     32940   ;ADDRESS OF TOP OF MEMORY PO
C       SCREEN   EQU     7000H   ;ADDRESS OF SCREEN MEMORY
C
C       ENTER    MACRO
C                PUSH    H       ;BEGIN DEFINITION
C                PUSH    D
C                PUSH    B
C                PUSH    PSW
C                ENDM            ;END DEFINITON
C
C       EXIT     MACRO   RVAL    ;BEGIN DEFINITION
C                POP     PSW
C                POP     B
C                POP     D
C                POP     H
C                IFNB    <RVAL>
C                  MVI     B,RVAL
C                ENDIF
C                RET
C                ENDM            ;END DEFINITION

                 EXTRN   CALL    ;CALL IS DEFINED OUTSIDE THIS

         START:  ENTER           ;MACRO: SAVE ALL REGS
          +      PUSH    H       ;BEGIN DEFINITION
          +      PUSH    D
          +      PUSH    B
          +      PUSH    PSW

                 MVI     A,(JMP)         ;GET JUMP OP CODE (O
                 ;                        FOR REGULAR ASSEMBLER)
                 STA     CALLVEC         ;PUT IT INTO CALL VE
                 LXI     H,CALL  ;GET ADDRESS OF CALLABLE SUBR
                 SHLD    CALLVEC+1  ;AND PUT IT INTO VECTOR, G
                 ;                       JMP     CALL

                 LXI     H,START-1 ;GET ADDRESS OF LAST BYTE O
                 ;                       ;MEMORY,
                 SHLD    TOPMEM  ;AND PUT INTO BASIC'S POINTE

                 EXIT    0       ;MACRO: POP REGS AND RET.
          +      POP     PSW
          +      POP     B
          +      POP     D
          +      POP     H
          +        MVI     B,0
          +      RET

                 END     START
```

SETUP Program

```
C               INCLUDE         MACS    ;PULL IN MACRO DEFS AN
C       ;MACRO LIBRARY AND SOME HANDY EQUATES.
C
C       ;SYSTEM EQUATES - ADDRESSES OF THINGS WE'LL NEED
C
C       CALLVEC  EQU     33282   ;ADDRESS OF VECTOR FOR CALL
C       TOPMEM   EQU     32940   ;ADDRESS OF TOP OF MEMORY PO
C       SCREEN   EQU     7000H   ;ADDRESS OF SCREEN MEMORY
C
C       ENTER    MACRO
C                PUSH    H       ;BEGIN DEFINITION
C                PUSH    D
C                PUSH    B
C                PUSH    PSW
C                ENDM            ;END DEFINITON
C
C       EXIT     MACRO   RVAL    ;BEGIN DEFINITION
C                POP     PSW
C                POP     B
C                POP     D
C                POP     H
C                IFNB    <RVAL>
C                  MVI     B,RVAL
C                ENDIF
C                RET
C                ENDM            ;END DEFINITION

                 ENTRY   CALL    ;MAKE NAME KNOWN OUTSIDE

         CALL:   ENTER           ;MACRO TO SAVE REGS
          +      PUSH    H       ;BEGIN DEFINITION
          +      PUSH    D
          +      PUSH    B
          +      PUSH    PSW

                 LXI     H,SCREEN+1  ;POINT TO FIRST CCI IN S

         LOOP:   MOV     M,E     ;PUT NEW CCI CODE DOWN

                 INX     H
                 INX     H       ;STEP TO NEXT CODE; IT'S TWO B

                 MOV     A,H     ;CHECK TO SEE IF WE'VE GONE P
                 ANI     0F0H    ;ZAP OUT LOW ORDER 4 BITS
                 CPI     070H    ;IF IT GETS TO 80, IT'S TOO
                 JZ      LOOP    ;STILL IN RANGE.

                 EXIT            ;MACRO: POP REGS AND EXIT
          +      POP     PSW
          +      POP     B
          +      POP     D
          +      POP     H
          +      RET

                 END
```

COLSCR Program

Listing 2

Lines 2 through 5 could have been written on one line as:

```
L80>/P:9000,SETUP,COLSCR/M,TEST/N/E
```

Line 1, of course, loads and runs L80. Line 2 sets the beginning of the PRG program to 9000H. We have no ORG statement, remember, so we need to tell L80 where the program should reside when run. (L80 will supply a default value of 8200H for CCII and 3621, A120H for 8000 series, and maybe even other values. On the Compucolor, L80 loads the program before returning to FCS, and will load the program right over the return stack, causing strange crashes at the very end of L80. To avoid them, place programs at 9000H instead of allowing L80 to use the default, or allow the crash and just hit CPU RESET. The PRG file will be OK.) Line 3 links our two routines; the /M option provides a map of all the external addresses and the beginning and ending addresses of the program. Step 4 assigns a name (TEST) to the PRG file, and Step 5 saves the PRG file, loads it, and exits L80. Listing the directory shows that TEST.PRG has been placed on disk, and is ready to be RUN.

For all its length, this article is just a brief overview of the Macro Assembler and related utilities. A full treatment would require a book. I hope that enough material is present that you can begin to use the program, and feel comfortable enough to experiment beyond the scope of this article. To wrap it up, there are some sections of the manual that should be ignored. They apply to other machines (those with CP/M or ISIS-II). There are other sections that should be deferred until you are at ease with the basics.

Ignore 2.1, 2.2, 2.2.1, 2.2.2, 2.11
Defer 2.6.1-3, 2.6.7, 2.6.14, 2.6.23, 2.6.28-29, 2.8

Tech Tip

by Gene Bailey
28 Dogwood Glen
Rochester, NY
14625

Some Compucolor owners who are bothered by colored squares of light on their screens will be interested in this:

Sometimes this problem can be caused by corroded connectors which corrode the power to the video RAM, which is especially sensitive. To cure the problem, **with the power off and the plug out,** remove the back cover, and remove and replace each of the three open-wire connectors on the right rear corner of the logic (bottom horizontal) board. Slide them up and down slightly to clean the contacts. Do the same with the connector on the power supply board (on the removable back cover). Put it all carefully back together.

(ed. - This simple trick does seem to work.)

PROGRAM PACKAGE INSTALLERS

New Compucolor Product: 8 May 1982

PROGRAM SELECTABLE CHARACTER SETS: PSC1

This small hardware unit plugs into the 50-pin bus to allow character sets held in EPROM to be selected either from the keyboard or within a BASIC program. Up to four sets of 128 characters can be selected using the OUT command. This considerably increases the flexibility of your Compucolor. All text can be in upper and lower case, while being able to rapidly switch to images using the full graphics set. An additional 256 characters can be selected; such as mathematical, electronic and music symbols. (Model PSC14)

The PSC1 can be connected to the existing dual character sets (graphics and lower case) to replace the panel switch. Three wires require reconnection.

In addition to its primary function, the PSC1 can also be used to switch on and off up to 8 remote devices. All data output lines are buffered and available with a positive or negative strobe pulse. The board also has spare gates and an IC socket for experimenters.

If supplied with EPROM character sets, no soldering is required for installation. A manual-overide switch is available as an option. (PSCMS)

PRICES (U.S. and Aust.$. Canada x1.25)

COMPUCOLOR 11

PSC1 (For existing dual character sets)	:$45
PSC12 (Including dual graphics/lower case)	:$75
PSC14 (Including four character sets)	:$95
PSCMS Three-position manual overide switch	:$10
PSCDC Define your own characters. Per 32:	:$20
MSC12 Dual character set with 2-pos switch	:$34
MSC14 Quad character set with 4-pos switch	:$56
PSC1-36 (For existing dual character sets on Intecolor 3651)	:$65

(All prices include airmail postage.)

P.P.I.
8 Hillcrest Drive,
Darlington,
Western Australia 6070

Another Debugger Bug

by Joseph Norris
David Hafler Co.
5910 Cresent Blvd.
Pleasantville, NJ 08109

Some versions of ISC's Machine Language Debug Package (MLDP) will fail to run on Series 3600 computers, V9.80. The difficulty is caused by a CALL to an improper ROM address. The symptom can be a keyboard "lockout" or a continuous scrolling of the prompt at run time.

To correct the problem, the first fourteen bytes of the program must be set to zero (NOP). The following simple procedure from BASIC will accomplish this. (Be sure to observe the single spaces in the lines typed in after the FCS prompts.)

A. For MLDP.PRG;01 (i.e., 16K versions):

1) **FCS>**LOAD MLDP.PRG;01
2) Go to BASIC with ESC E.
3) In immediate mode enter:

```
FOR N=57344 TO 57357:POKE N,0:NEXT N
```

4) Re-enter FCS with ESC D.
5) **FCS>**SAVE MLDP.PRG;3 E000 1F80
6) MLDP.PRG;01 may now be deleted and the new MLDP.PRG;03 RENamed MLDP.PRG;01

B. For MLDP.PRG;02 (i.e., 32K versions):

1) **FCS>**LOAD MLDP.PRG;02
2) Go to BASIC with ESC E.
3) In immediate mode enter:

```
FOR N=33280 TO 33293:POKE N,0:NEXT N
```

4) Re-enter FCS with ESC D.
5) **FCS>**SAVE MLDP.PRG;04 8200 1F80
6) MLDP.PRG;02 may now be deleted and the new MLDP.PRG;04 RENamed MLDP.PRG;02

An additional bug in the 16K version is discussed in the OCT/NOV 1981 issue of COLORCUE, page 3.

COMPUCOLORS FOR SALE

CCII, 32K, V6.78, std keyboard, manuals, disks. Factory reconditioned. $1300. Mel Bomze, 516-724-2054 evenings.

CCII, 16K, std keyboard, basic set of games, Assembler, Compuwriter, editor, FREDI.
Charles Lovejoy, 49 South St., Natick, MA 01760 800-225-2465x1365 days

CCII, 16K, V6.78, 71 key keyboard, manuals, disks, Assembler, games. Asking $1000 (list $2200.) Darryl Nadvornick, 213-864-0440 eves (PST) 213-868-0431x414 Weekdays 8-4 (PST)

CCII, 16K, 117 key keyboard, prog. and maint manuals, soundware, games and some household programs. Includes Paper Tiger 460 printer w/ 2K buffer, graphics, cable, plus free box of paper and 5 binders. Like new, $2200. Darrin Miller, 238 Alderson, Billings, MT 59101, 406-259-1924 eves. 406-252-2299 days.

COMPUCOLOR II (32K)

BUSINESS & EDUCATION SOFTWARE

TAXOMATION INCOME TAX UTILITY program written for tax professionals (1981 tax season). It handles all validation to meet the IRS. Has internal computation for earned income credit, excess FICA etc. Program is using the 24-hour banking arrow-type instructions.

Also comes with FED form 1040, 1040A, Sch A,B,G; New York IT200, IT201, IT214.

PRICE $200.00

CASH REGISTER SOFTWARE (FOR RESTAURANT)

Designed for general business as well as for restaurant. Display 30/60 items per screen, inventory control, cash/credit management, order entry/billing. It is excellent for Telephone Orders, Normal In-House and Fast Food Orders. Will convert to other computers.

PRICE $500.00

CHINESE MAJONG GAME

A complete simulation of ancient Chinese Majong on CompuColor II, written in BASIC. It is a one player game. Program will display the actual Chinese + Graphics.

PRICE $200.00

QSORT Utility program to sort fixed length record, variable key location, and length of key.

PRICE $ 20.00

INTERACTIVE OS

Interactive Operating System written to integrate your MENU program to make most control of your computer (like TSO). It has 30 commands which include calculator, set date and time, create data file, list CCII key memory location, printer control(14 commands) etc. Excellent for education demon. or hardware demon. Ten-page of program listing enclosed.

PRICE $ 79.99

* * *

Back Issues Sale

Back issues of **Colorcue** are an excellent source of information about Compucolor computers, ISC computers, and programming in general. Interviews, interesting articles, and programs are all there with a touch of history.

The list below includes every **Colorcue** ever published. If it's not on the list, then there wasn't one.

MULTI-ISSUES at $3.50 each

__ Oct, Nov, Dec 1978
__ Jan, Feb, Mar 1979
__ Apr, May/June 1979
__ Aug, Sept/Oct 1979

INDIVIDUAL ISSUES at $1.50 each

__ Dec 1979/Jan 1980
__ Feb 1980
__ Mar 1980
__ Apr 1980
__ May 1980
__ Jun/Jul 1980

INDIVIDUAL ISSUES at $2.50 each

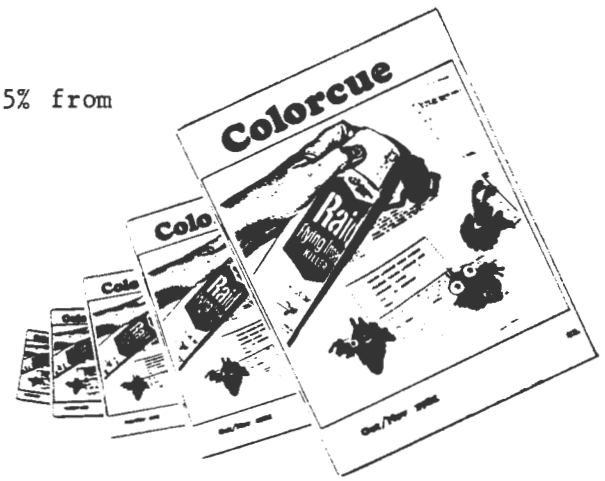
__ Dec 1980/Jan 1981
__ Aug/Sep 1981
__ Oct/Nov 1981
__ Dec 1981/Jan 1982
__ Feb/Mar 1982
__ April/May 1982
__ June/July 1982

POSTAGE

US and Canada -- First Class postage included.
Europe, S. America -- add $1.00 per item for air, or
$ .40 per item for surface.
Asia, Africa, Middle East -- add $1.40 per item for air, or
$ .60 per item for surface.

DISCOUNT

For orders of 10 or more items, subtract 25% from total after postage.



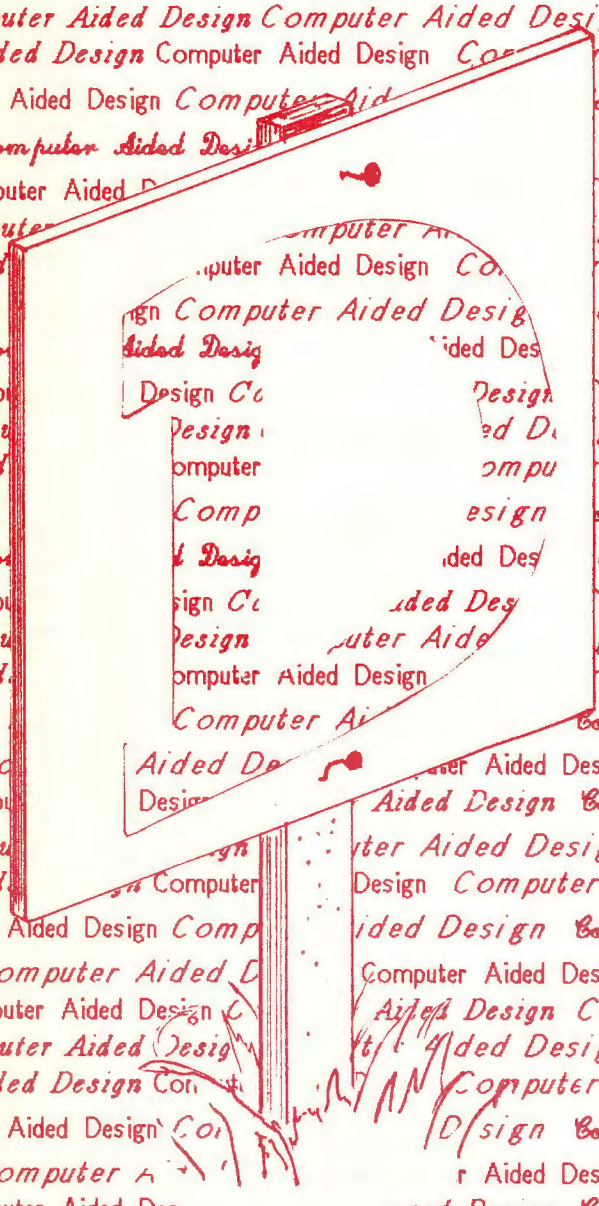
ORDER FROM:

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14623

BULK RATE
U.S. POSTAGE
PAID
Rochester, N. Y.
Permit No. 415

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

Colorcue



Oct/Nov 1982

$2.

Colorcue

A bi-monthly publication by and for Intecolor and Compucolor Users

August/September 1982
Volume 5, Number 2

Editors:
Ben Barlow
David B. Suits
Compuserve: 70045,1062

Authors: This is a user-oriented and user-supported publication. Your articles, tips, hints, programs, etc. are required to make it go. Write or scribble your ideas down; we'll edit them and provide all artwork. Send your articles or write for information.

COLORCUE is published bi-monthly. Subscriptions are US$12/year in the U.S., Canada and Mexico, and US$24 (includes air mail postage) elsewhere. Some back issues are available. All editorial and subscription correspondence should be addressed to **COLORCUE**, 161 Brookside Dr., Rochester, NY 14618, USA. All articles in **COLORCUE** are checked for accuracy to the best of our ability, but they are NOT guaranteed to be error free.

Editors' Notes

Compucolor owners have often been left in the cold when their machines break down; dealers have gone out of business or switched to other lines and factory service is not always satisfactory. **Colorcue** has located two sources that may be helpful:

Gary Sipple
Digital Devices
20560 West 8 Mile Road
Southfield, MI 48075
(313) 356-5140

Fred Calev
Compuworld
125 White Spruce Blvd.
Rochester, NY 14623
(716) 424-6260

Both companies will handle "shipped-in" machine service, but should be contacted first, of course.

Australian ISC and Compucolor owners are active, as you've no doubt noted from past issues of **Colorcue**. The maintenace tips in this issue come from **CUVIC**, of Melbourne - editor Barry Holt, 19 Woodhouse Grove, Box Hill North, 3129, Victoria, Australia. Local membership is $10. per year; non-Australian must be more, but we don't know exactly how much. The Western Australian User Group **CUWEST** is also active, and their newsletter typically runs 3 or 4 pages of worthwhile information.

Several times over the last year, we have received letters from subscribers saying that "such and such an issue of **Colorcue** didn't arrive." As you know, we use bulk mail (first class mailing would wreck our budget) and the Post Office seems to treat such mail with very low priority. If you miss an issue, or think you have (be careful - we seem to be running late recently), just let us know, and we'll send off the missing copy (first class).

User Groups - are you alive?? Drop us a note so we can publish an up-to-date user group list.

Notice to hardware and software suppliers: Our readers are frequently unaware of the availability of your wares and would like to be customers. **Colorcue** would like to spread the word - your word- to them. We can do this in two ways:

1. Through advertising. Our rates are low, and our readership is eager for your products.
2. By listing all known suppliers, which we plan to do in the Jan/Feb issue.

Our big problem - even we, who have been around for four years, don't know all the suppliers and their present status. So, if you supply hardware or software for Intecolor or Compucolor computers and would like to have more business, let us know. Even if you don't want to advertise, send your name and address for our suppliers listing.

In the last issue we promised to bring you more Fortran in this issue. We apologize to you Fortraners, but it'll be next issue before we get back into Fortran with a demonstration program and a set of subroutines to make use of the special ISC graphics features.

Bug Report!
Howard Rosen reports that a bug crept into his Fortran program in the last issue, FORTRAN Programming. After the line:

```
INX = INX + I
```

should have been a line like:

```
DINX = DINX + I
```

If you got strange results, take note.

Flash! It seems that a second (or is that a third by now?) BASIC compiler is in the works, somewhere Down Under. Hold your breath. We hope to have more details by next issue.

Intelligent Systems Corp. has announced that it has reached an agreement to acquire the Quadram Corporation. The acquisition is valued at $35,000,000 and will be financed through an issuance of approximately 1,750,000 additional shares of ISC stock. When the transaction is completed, ISC will have approximately 4,357,000 shares outstanding.

Peter J. Curnin, ISC president, stated that Quadram has a leadership position in

the design, manufacture and marketing of accessories for the IBM Personal Computer. Quadram will continue to operate as an independent business unit, and Tim Farris, co-founder and President of Quadram, and Leland Strange, Vice President and co-founder, will be added to ISC's Board of Directors.

ISC has announced the Executive Presentation System (EPS), which provides all the hardware and software users need to prepare presentation-quality color graphic visuals, including overhead projector transparancies, 35mm slides, and paper prints and plots. The EPS includes a graphics language that utilizes English statement commands such as PIE, BAR and LINE. EPS is a complete system featuring the Intecolor 8001R and ISC's newest microcomputer, the Intecolor 7000. Since EPS is device independent, the user can include output devices such as 8-pen color plotter, color camera system, and an 8-color ink-jet printer. And the resolution is determined by the output device.

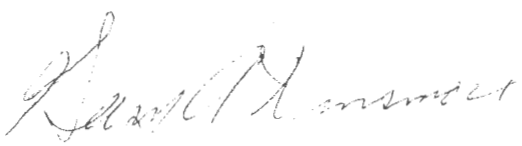
Moving?

If you're changing your address, please let both the Post Office and us know of your new address. (Tell us your old address and your new one.) We don't want you to miss a single issue of **Colorcue.**

Dear Ben and David:

I have occasionally lost some data by accidentally hitting the CPU reset key when reaching for the erase line key. I have hit upon an idea to make it more difficult to press this key. Pick up a piece of surgical rubber tubing a couple inches long. You can get this at a chemical supply house, or possibly a well stocked drug store. The tubing should be one fourth inch inside diameter, and have 1/16 inch thick walls.

Use a sharp knife to cut a piece of tubing exactly 5/8ths of an inch long with good square ends. Pry the CPU reset key from its switch, and slide the tubing piece over the socket in the underside of the key cap. The key can be then replaced. Test this setup to see that you can still depress the key, but that it requires a firm push to make contact. Just a slight change in the length of the tubing will make quite a change in the pressure required to depress the reset key. This way a normal typing touch will not reset the computer, and you may get a second chance at your data.



Gary A. Dinsmore

A CAD Program

by Doug Van Putte
18 Cross Bow Drive
Rochester, N.Y. 14624

A clever and educational Computer-Aided Design (CAD) program for a microcomputer appeared in the MACHINE DESIGN magazine which I have converted to ISC BASIC. The program, written for the APPLE II, appeared in a Tech Brief entitled 'Drawing Isometric Views with a Microcomputer' in Volume 54, Number 25. It was written by Professor Dad-Ning Ying, University of Wisconsin. The article gives a brief overview of the program but leaves the reader lacking the visualization of the conversion from a 3D object to 2D screen views. I intend, through use of a figure, to explain the conversion process for those interested in learning more about CAD.

The program in Listing 1 converts a three dimensional object, stored in coordinate form, to the conventional drafting views: Front, Top, and Profile. In addition, an Isometric view of the 3D object is drawn. At this point, the reader should run the program to obtain an understanding of the types of views created of the stored object. The lower left hand view on the screen is the Front, the lower right hand view is the Profile, and the upper left hand view is the Top. Now consider Figure 1, a representation of the aformentioned conversion process. The figure depicted here is a 'bookend' style object on its side. Note that the figure is represented by the dimensions of u, v, w on a cartesian coordinate system with the axes labels of X, Y, and Z. The X and Y coordinates are the typical screen coordinates, while the Z axis can be thought of as an axis which is perpendicular to the screen. The equations for computing the screen coordinates from the object coordinates are shown on Figure 1. Without further explanation, they simply resolve the u, v, and w coordinates of the object to the screen coordinates, X and Y, depending on the values of the rotation angles specified. See Professor Ying's article for further explanations of the equations. The rotation angles, θ and ϕ are what I will discuss in more detail in the paragraphs that follow.

To visualize the creation of each view by rotation, the object in Figure 1 should be rotated by the angles indicated in the table at the bottom of the page. For the Front view, no rotation is required. The screen X values become the u values and the screen Y values become the w values when the angles are substituted in the equations. For the Profile view, the object is rotated about the Y axis by 90 degrees. The X values become the v values and the Y values become w values.

For the Top view, the object is rotated about the X axis by 90 degrees. The X values become u values and the Y values become v values. By now the pattern should be clear, the object is rotated and its dimensions are projected by trigonometry to the plane of the screen to obtain the various views.

The isometric view is created by rotating the object thru two angles. To be strictly correct, the object should be rotated 45 degrees or its supplement(135) about the Y axis and 35.26 degrees about the X axis. To eliminate hidden lines from this view, the program cheats a bit by not plotting the last four data points. Other simulated 3D figures or axonometric views can be made by specifying different angles on line number 230 in Listing 1. To obtain the typical engineering drafting views, rotations are not required about the Z axis. These manipulations can be made after modifications are made to the equations. (See the

3D Graphics article in the Feb/Mar Issue of **Colorcue**).

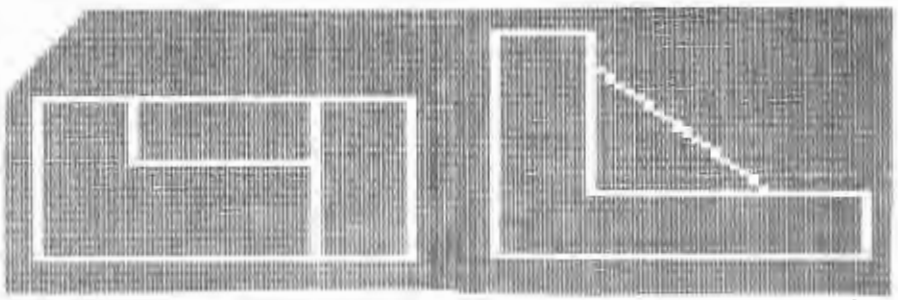
The program can be changed to draw other figures. I suggest that the new figures be layed out on a coordinate system similar to Figure 1 to assign the u, v, and w values, including the proper signs. Enter the number of data points in line 310 and the object data in groups of u, v, w values in the subsequent lines. As Ying suggests, the object data could be stored in files. With the program altered to read a specified file, unlimited types of figures could be plotted. This sounds like an excellent project for a high school drafting class to learn more about computer-aided design, doesn't it?

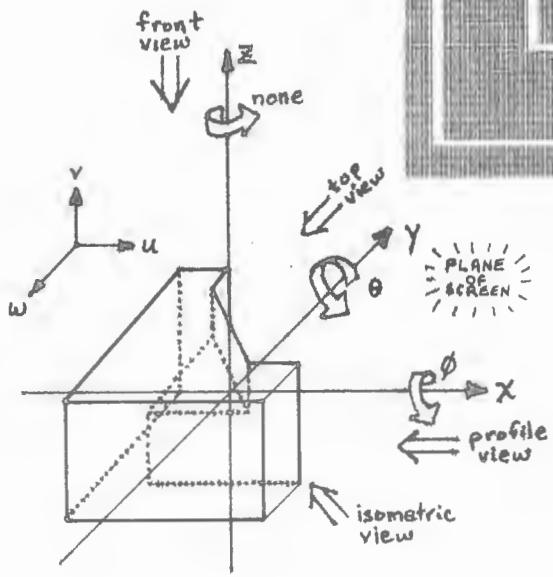
Pay attention, students of design of all ages, this is the design method of the future that is making a significant impact today...

```
 0   REM  LISTING 1: AN ISC BASIC INTREPRETATION OF
20   REM  'A PROGRAM FOR ISOMETRIC AND THREE VIEWS'
22   REM
24   REM
30   REM  WRITTEN BY DAO-NING YING, U. OF WISCONSIN.
40   REM  CONVERSION OF PROGRAM WHICH APPEARED IN
50   REM  'MACHINE DESIGN', V.54, #25, 1982 BY D.A. VAN PUTTE.
60   REM  ARTICLE WAS ENTITLED 'DRAWING ISOMETRIC VIEWS
70   REM  WITH A MICROCOMPUTER'

80   DIM U(100),V(100),W(100),X(100),Y(100)
90   PLOT 12,15
     :SC = 6.1                    :REM  INPUT SCALE OF IMAGE
100  READ N                       :REM  READ NO. OF DATA POINTS
110  FOR J= 1TO N                 :REM  READ 3-D DATA POINTS
120  READ U(J),V(J),W(J)
130  NEXT J

140  FOR K= 1TO 4
     :PLOT 16+ K                :REM COMPUTE & PLOT VIEWS
150  FOR I = 1 TO N               :REM COMPUTE & PLOT SCREEN COORDS

160  REM  TOP VIEW PARAMETERS
170  IF K= 1THEN TH= 0* .0174533
     :FI= 0* .0174533
     :X0= 15
     :Y0= 50
     :GOTO 250

180  REM  PROFILE VIEW PARAMETERS
190  IF K= 2THEN TH= 90* .0174533
     :FI= 0* .0174533
     :X0= 90
     :Y0= 50
     :GOTO 250

200  REM  TOP PARAMETERS
210  IF K= 3THEN TH= 0* .0174533
     :FI= 90* .0174533
     :X0= 15
     :Y0= 110
     :GOTO 250
```

```
220   REM  ISOMETRIC VIEW PARAMETERS
230   TH= 135* .0174533
      :FI= 35.26* .0174533
      :XO= 90
      :YO= 110

240   REM  COMPUTE TWO DIMENSIONAL COORDINATES
250   X(I)= U(I)* COS (TH)- V(I)* SIN (TH)
260   Y(I)= - U(I)* SIN (TH)* SIN (FI)- V(I)* COS (TH)* SIN (FI)- W(I)* C
      OS (FI)
270   X(I)= INT (X(I)* SC* .8+ XO+ .5)
      :Y(I)= INT (Y(I)* SC+ YO+ .5)
280   IF I= 1THEN PLOT 2,X(I),Y(I),242
      :GOTO 300
290   PLOT X(I),Y(I)
300   NEXT I
      :PLOT 255
      :NEXT K
      :INPUT "";ZZ              :REM HANG HERE TO KEEP SCREEN CLEAR
310   DATA  29                    :REM COMPLETE FIGURE HAS 33 DATA POINTS
320   DATA  0,0,0,0,5,0,-2,5,0,-2,0,0,0,0,0,0,0,5
330   DATA  0,3,5,0,3,1,0,5,1,0,5,0,0,5,1,4,5,5
340   DATA  4,3,5,0,3,1,0,3,5,4,3,5,4,5,5,6,5,5
350   DATA  6,5,7,6,0,7,6,0,5,6,0,7,-2,0,7,-2,0,0,0,0,0,0,0,5
360   DATA  6,0,5,6,5,5,6,5,7,-2,5,7,-2,0,7,-2,5,7,-2,5,0
370   END
```

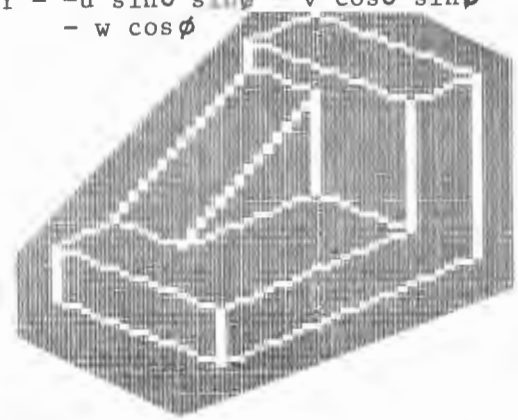
Figure 1





Equations: $X = u \cos\theta - v \sin\phi$

$Y = -u \sin\theta \sin\phi - v \cos\theta \sin\phi - w \cos\phi$



VIEWS	Rot. ∠'s θ	φ	Screen Coord. X	Y
front	0	0	u	-w
top	0	90	u	-v
profile	90	0	-v	-w
isometric	135	35.26	X_I	Y_I

$X_I = -.707\ u - .577\ v$

$Y_I = -.408\ u - .471\ v - .817\ w$

H O W A R D R O S E N, I N C.

P.O. Box 434
Huntingdon Valley, Pa.
19006
(215)-464-7145

*** * * * BUSINESS SOFTWARE * * * ***

LEDGER

Every business and home should have this program. LEDGER allows you to do a Receipt page, a Dispersment page, a Dues Collection List, a Budget, and any other form that you may have developed that uses rows & columns for numerical data storage with Titles. This easy and useful to use program allows 31 columns of data, and a 32nd column totals each row. There are 80 rows for each column and column totals. Intermediate row sub-total arithmetic is user defined. The arithmetic functions permitted are +,-,*,/,=. Saving, Loading, and Replacing data to the File Control System (FCS), Printing the Ledger sheet, and easy trial entries and changes make this a power-house. Requires 32K RAM and 117-key keyboard.

LEDGER disk includes **LEDGER, Instructions, PRINTER DRIVER, & Printer Driver Instructions.**

price 75.00

PERSONAL DATA BASE

PDB written in Assembly Language allows you to create a data base file consisting of data base records. Records are composed of a mix of literal and numerical fields as required. The records may then be used for statistical analysis, mail merge insertions for the mail merge word processor, data storage, retreival and sorting. Records may be added, changed, deleted, & searched. 32K holds 1200 records.

Personal Database II	price 85.00
Options:	
Plotting program - screen/printer	price 30.00
Distribution Analysis - Statistics	price 30.00
Encode/Decode Data/Hold Files	price 15.00
Math Option I - (+,-,*,/)	price 15.00
Math Option II - ($,+,-)	price 15.00
Form Processing	price 35.00
Left/Right Justification	price 10.00
Mail Merge Insertion	price 20.00

NOTE: PERSONAL DATA BASE and any 4 options priced at 10% discount.

EXECUTIVE WORD PROCESSOR	price 299.00
MAIL MERGE WORD PROCESSOR	price 349.00

Disc Data Recovery

by Myron T. Steffy
10833 Brookside Drive
Sun City, Arizona 85351

Have you ever accidentally dumped a disc containing a source file which meant hours and hours of work down the drain? Well, I have and if you work in assembly language it will happen to you some day. It has always been my practice to maintain duplicate files of work in progress but this time, even that wasn't enough. Somehow, the program which was quite long, developed an unexpected bug after a test assembly. I might mention that I have two of Tom Devlin's RAM cards at 4000H and park the screen editor in one and the assembler in the other which allows me to use them alternately without reloading.

When I tried to bring up the source file in question some unexpected things happened, probably due to the remnants of the defective program still in memory. The disc drive started but instead of the source file, I got some little red letters on the screen, saying 'ENVE'. This usually means that the directory is wiped out and the disc must be re-initialized. Oh well, the back-up disc will take care of the problem, said I confidently. Well, I should have run a program that I keep on hand that clears all of the memory but I didn't. Would you believe that I blew the second disc the same way?

When I stopped shaking, I tried to figure out a way to salvage at least part of the file. Due to its length, there were probably only two versions of it on any single disc. Here is the way I went about it: contrary to what you may have thought, when you re-initialize a disc, only the directory is affected. The file names are still present but the essential information of where they begin and their size has been erased. A new file is simply written over the old.

There is a command in the FCS system called 'READ' that you may have never used. This is not to be confused with the Basic command of the same name which performs a different function. What we need to do is get the information on the disc into memory where we can look at and possibly salvage it. We will assume that you have 32K of RAM and also have a 'Debugging' program similar to the Comtronics 'DBUG' or the later version, 'NBUG'. If you have it available at 4000H, so much the better. Load it so that it is ready to run.

Here is what we do with 'Read' function mentioned earlier. The command is 'READ CD0:00 8200-FFFF' which reads 7DFFH bytes off the disc into RAM starting at block 00. The first three or more blocks will have the directory names still present but no information about location and size of the file. Now bring up the NBUG program and start looking. If you remember a key word in the heading of the file, there is a 'Find' command in 'NBUG' that will locate it in a hurry.

Start at 8200H and make a note of the actual RAM location where the file begins. Now locate the end of the file the same way only this time the search word could be 'END'. Then place a fresh disc in the default drive and enter the command 'SAVE PROGRM.SRC 8580-89DC' where the two hex limits are the start and end of the file you have just determined. If you do everything right, you have a replica of the original source file that can be brought up with the screen editor in the usual manner. Don't worry about the load address appearing as '8580' on the disc directory. The editor will supply the correct address when it loads the file.

If you can make an educated guess of how far down the disc your file is located, you won't have to look through the entire 51K. Otherwise there will be 0FBH blocks left on the disc, a little less than half (190H-0FBH), because you took off 0FFFFH 8200H = 0FBH. The next time start at block number 0FAH for a little overlap and read off the rest of the disc. If you don't have the debugger at 4000H, you probably have it at 0A000H or 0E000H. Use the latter and only read off 8200H-0DFFFH. This will be 5DFFH divided by 80H or 0BBH blocks at a chunk. The second starting block would then be 0BAH. Your search will be from 8200H to 0DFFFH and will take a few more steps but will accomplish the same result.

If the arithmetic is confusing, remember that a disc contains 190H blocks of 80H, which total 0C800H or 51.2 K in decimal. Most formatters set you up with the first three blocks as directory. This is really not enough so mine allows five automatically. If you happen to remember approximately what the file's starting block number was, you won't have to look far. The NBUG 'Find' command is a real asset and the job is much less formidable than it sounds here. Don't panic and jump off the bridge until you try this first. Better yet, make a practice run before you are forced to use the method.

Cueties

How Did Sam Die? (A Computer Puzzle)
By David B. Suits

Here's a little puzzle for the mystery lovers among us. The program tells the tale. How did Sam die? Your computer knows. Can you figure it out without its help? (If not, type in the program exactly as it appears below and RUN it for the answer.)

```
10 TRUE = NOT(FALSE)
20 SAM IS DEAD = TRUE
30 SUICIDE = SAM IS DEAD AND NOT(ACCIDENT) AND NOT (HOMICIDE)
40 HOMICIDE = SAM IS DEAD AND NOT(SUICIDE OR ACCIDENT)
50 PRINT "SAM'S DEATH WAS: ";
60 IF ACCIDENT THEN PRINT "ACCIDENTAL"
70 IF HOMICIDE THEN PRINT "HOMICIDE"
80 IF SUICIDE THEN PRINT "SUICIDE"
```

Was Einstein Correct? (A Computer Puzzle)
By David B. Suits

Decades after Einstein's momentous Theory of Relativity, scientists still debate the question, "Was Einstein correct?" Your computer knows. Can you tell before asking its counsel? (To check your answer, key in the program below and RUN it.)

```
10 IF E = MC^2 THEN EINSTEIN = CORRECT
20 IF EINSTEIN = CORRECT THEN PRINT "EINSTEIN WAS CORRECT"
30 IF EINSTEIN = NOT(CORRECT) THEN PRINT "EINSTEIN WAS NOT CORRECT"
```

Software Handshake for Diablo 630

by Vance Pinter
P.O. Box 230
Columbus, Georgia 31902

I decided to use software handshaking with my Diablo 630 so that it wouldn't have to be modified internally. The stock 630 does not provide a "busy" signal unless so modified.

The Diablo 630 buffer holds 768 characters. When the buffer is nearly full the 630 sends DC3 (hex 13, decimal 19) out on pin 2. It continues printing until the buffer is nearly empty and then sends DC1 (hex 11, decimal 17). The cable requires three wires:

(1) Ground. Diablo pin 7 to CCII edge connector pin 1 or 7 or ISC DB25 pin 7.

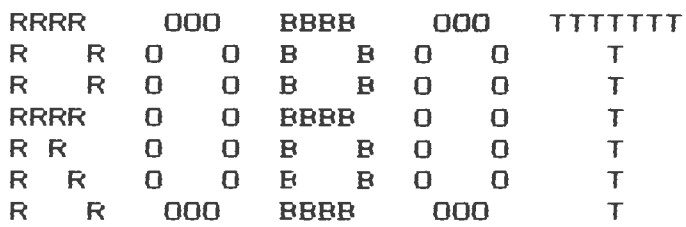
(2) Signal to printer. Diablo pin 3 to CCII edge connector pin 3 or ISC DB25 pin 2.

(3) Signal to computer. Diablo pin 2 to CCII edge connector pin 5 or ISC DB25 pin 3.

Diablo pin 4 must be connected to Diablo pin 6. Use a short wire inside the printer plug.

Here is an assembly language routine which sends a byte to the printer after checking the status of the 630.

```
;Character to be printed is in register E.
;S1OUT is the CCII serial output routine,
;V6.78 address 33C3H, V8.79 address 17F9H.

S1OUT   EQU     33C3H   ;V6.78 address, V8.79 is 17F9H.

OUT232: CALL    STATUS  ;Wait until clear to send.
OUT001: CALL    S1OUT   ;CCII serial output routine.
        RET

;Status routine returns only when "Clear To Send"

STATUS: IN      03H     ;TMS 5501 status register.
        ANI     20H     ;Diablo status present?
        RNZ             ;No, return, clear to send.
        IN      00H     ;Input Diablo status.
        CPI     13H     ;Diablo buffer full? (DC3)
        RNZ             ;No, return, clear to send.

;Diablo buffer full.
;STLOOP will loop until DC1 is received from 630.

STLOOP: IN      03H     ;TMS 5501 status register.
        ANI     20H     ;Diablo status present?
        JNZ     STLOOP  ;No, wait until it is.
        IN      00H     ;Input Diablo status.
        CPI     11H     ;Diablo buffer empty? (DC1)
        JNZ     STLOOP  ;No, continue loop.
        RET             ;Diablo buffer empty, return.
```



W A R S

(c) 1982 BY Steve Reddoch
(With sound)

$19.95 U.S. funds

For 5 1/4 inch disk drive only

Fast machine language almost like the arcade game Robotron(c). There are 2 songs, the sound never affect the speed of the program, and there are 8 different looking aliens. There are 3 skill levels that control the speed of one of the alien's shots. Also the high scores are save on disk.

For the Compucolor II & Intecolor computer
Runs on V9.80, V6.78, V8.79, V9.80-03

Uses the full color graphics
capibility of the Compucolor II/Intecolor
(Requires 16k RAM)

Please send check or money order for $19.95 to:

Steve Reddoch
1158 Via Bolzano
Santa Barbara, Ca 93111

Ca. Residents add 6% Sales Tax
Also include version number of computer
Please do not send cash

Typematic Keyboard

by Doug Pankhurst

Reprinted by permission from CUVIC
(Compucolor/Intecolor Melbourne
User Group newsletter, November, 1982)

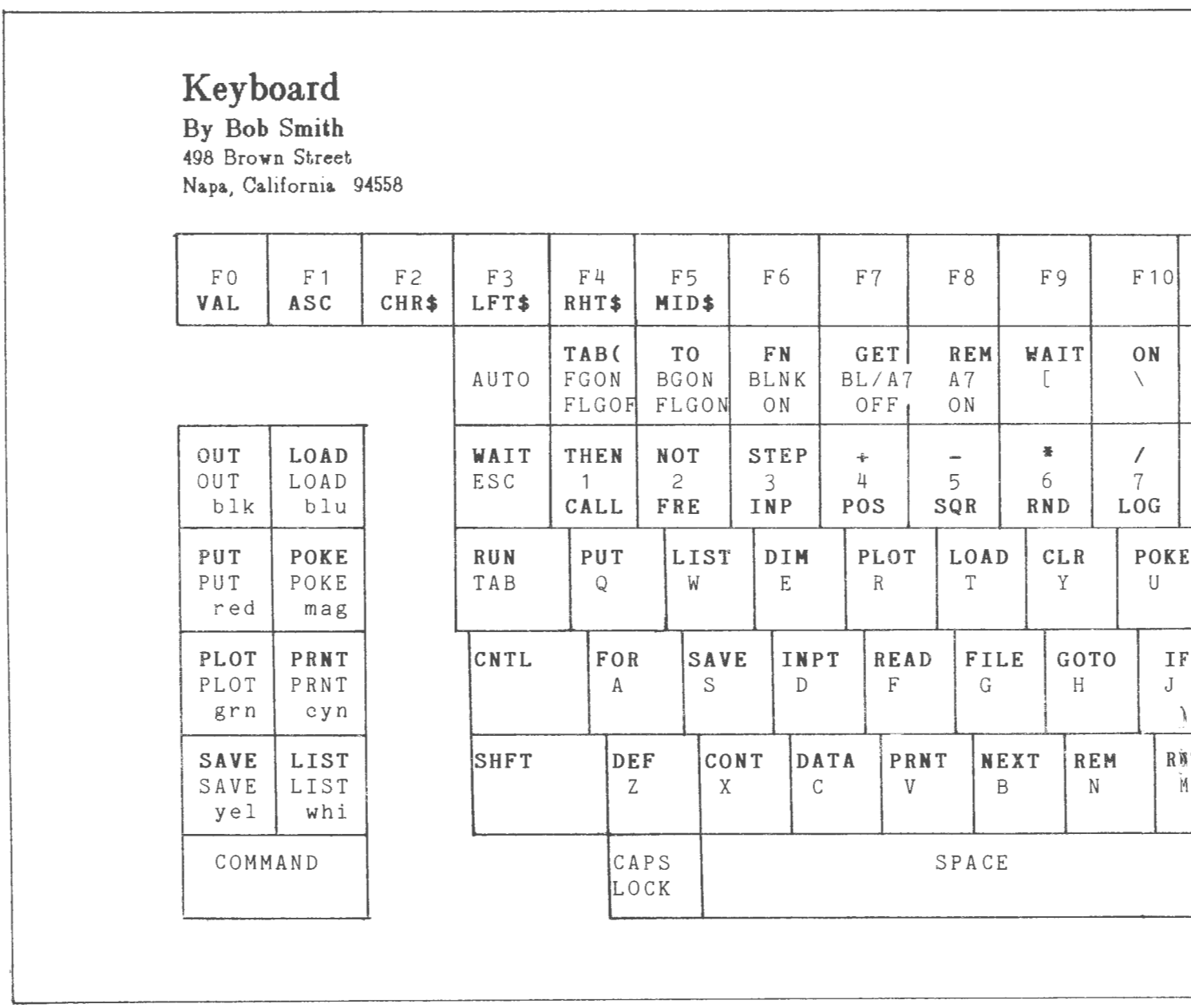
19 Woodhouse Grove
Box Hill North
3129 Victoria
Australia

```
;Sixty times a second an interrupt is generated which ini-
;tiates a keyboard scan.  Routines in ROM get the code for
;the key that was pressed and store it in location 81FE
;hex (33278 dec) and sets location 81FF hex (33279 dec) to
;80 hex (or 50 hex if the key pressed was the BREAK key).
;.....................................................
;       SYSTEM EQUATES
;.....................................................
;These should be included in the System Equate table at
;the start of the program.
KBDFL   EQU     81DFH   ;33247 decimal. Location of key-
                        ;board interrupt vector flag. By
                        ;inserting 1F hex in here program
                        ;control will vector to INPCRT.
LKC     EQU     81E4H   ;33252 dec. Last key code.
NKC     EQU     81E5H   ;33253 dec. New key code.
LWAIT   EQU     2400H   ;Count for initial wait when us-
                        ;ing auto repeat feature. (400H
                        ;approx. equal to eight per sec-
                        ;ond).
INPCRT  EQU     81C5H   ;33221 decimal. User input vector
                        ;in which a JMP [user routine] is
                        ;inserted.
KBCHA   EQU     81FEH   ;33278 dec. After keyboard scan,
                        ;code of key pressed (zero if
                        ;none) in here.
KBRDY   EQU     81FFH   ;33279 dec. Set to 80H if key is
                        ;pressed, or 50H if key was BREAK.
;.....................................................
;       INITIALIZATION
;.....................................................
;This routine should be carried out at the start of the
;program to initialize interrupt vectors. Note that if
;other interrupts use the vector INPCRT the CHRINT needs
;to determine where the interrupt was generated and what
;action, if any over and above that taken by the ROM based
;routines, needs to be taken.

        ORG     9000H   ;START HERE
START:
KYINIT: LXI     H,KBDFL ;Set user vector into keyboard
                        ;flag.
        MVI     M,1FH   ;
        LXI     H,INPCRT;
        MVI     M,0C3H  ;Opcode for JMP.
        LXI     H,CHRINT;Address of user interrupt
                        ;routines
        SHLD    INPCRT+1;into user i/p interrupt vector.
;       ... continue with mainline program.
;.....................................................
        JMP     MAIN    ;
;.....................................................
;       INTERRUPT SERVICE
;.....................................................
;Vector here from INPCRT and carry out any time critical
;routines, determine whence the interrupt came if multiple
;interrupts may occur (i.e. use i/p vector INPCRT) and
;carry out any user interrupt routines in addition to the
;ROM based system routines. The interrupt service MUST end
;with an RET instruction. The contents of all registers on
;entry will be preserved. BREAK key detection should be
;included here if the ability to break out of a routine is
;required. A BREAK may be detected by a code of 50H in the
;KBRDY flag as opposed to 80H for any other key. If a
;BREAK detection is used and program control is trans-
;ferred elsewhere, remember to adjust the Stack Pointer,
;as it has an extra return address on the top (from the
;periodic keyboard scan interrupt routine that got us
;here).
        ORG     0A000H  ;
CHRINT: RET             ;i/p vector - simply return.
;Note that by taking no action here, any key with the
;exception of the CPU RESET key may be passed on to the
;user via GETCHA or GETKEY and used for any purpose
;(i.e. the action taken is entirely up to the user to de-
```

```
;fine). All codes from 00H up to FFH (255 dec) are avail-
;able. Be aware that the BREAK key, as well as setting
;KBRDY to 50H, puts zero into KBCHA.
;.....................................................
;       GET CHARACTER ROUTINES
;.....................................................
;These routines may be included at any place within the
;program and may be called by any user. Be aware that both
;will hang until a key is pressed (or accept the last key-
;press). Both routines preserve all registers except the A
;register, in which the value of the key pressed is re-
;turned.
MAIN:   JMP     HMWRK   ;
;Get character with no auto repeat.
GETCHA: EI              ;Get keypress, return character
        CALL    GKEY1   ;without auto repeat
        PUSH    PSW     ;in Acc.
        XRA     A       ;
        STA     KBRDY   ;Clear k/board ready
        STA     KBCHA   ;and character buffer
        POP     PSW     ;for single char
        RET             ;operation.
;Get character with auto repeat as long as key is
;pressed.
GETKEY: EI              ;Typematic key get.
        CALL    GKEY1   ;Go get keypress
        PUSH    PSW     ;
        XRA     A       ;and clear k/board
        STA     KBRDY   ;ready flag.
        POP     PSW     ;
        RET             ;
```

Keyboard

By Bob Smith
498 Brown Street
Napa, California 94558



```
;       Subroutine used by GETCHA and GETKEY.
GKEY1:  PUSH    H       ;
        LHLD    REPCHR  ;Get wait counter.
GKLOOP: LDA     LKC     ;If last key code is zero,
        ORA     A       ;go wait for another
        JZ      GKEYEX  ;keypress,
        DCX     H       ;else start decrementing
        MOV     A,L     ;HL until zero or
        ORA     H       ;last keycode is zero.
        JNZ     GKLOOP  ;
GKEY2:  LXI     H,SWAIT ;Set up for short
        SHLD    REPCHR  ;wait and get last
        LDA     KBCHA   ;key pressed.
        ORA     A       ;Is it zero?
        JZ      GKEYEX  ;If not, go exit,
        POP     H       ;else return with
        RET             ;character in A
GKEYEX: LXI     H,LWAIT ;else set up long wait
        SHLD    REPCHR  ;
GKEY3:  LDA     KBRDY   ;and wait
        ORA     A       ;for keypress.
        JZ      GKEY3   ;
        LDA     KBCHA   ;Get character into A
        POP     H       ;and return.
        RET
;       TEMPORARY STORAGE
REPCHR: DS      1       ;Repeat rate counter used
                        ;by auto repeat feature.
;.....................................................
;       END OF KEYBOARD ROUTINES
;.....................................................
```

Top - **Token when struck with COMMAND (Control and Shift)**
Center - Character that appears on keys
Bottom - **Token when struck with Control**
(Bottom)- Token when struck with Shift

F11	F12	F13	F14	**ON** **(ON)**	F15
TAB( ]	**TO** ^	**FN** _	**DEF** **(DEF)**	**GOTO** HOME **(GOTO)**	**CLR** **(CLR)**
^ 8 **EXP**	**AND** 9 **COS**	**SPC(** 0 **ABS**	**PEEK** - <	**IF** **(IF)**	ATTN BRK
RUN I	**GET** O	**OUT** P	**END** @	RETURN	
	RESTR K	**GOSUB** L	>**TAN** ; :	**OR** **SIN**	ENTER
TRN	**ATN** , =	**LEN** . **SGN**	**STR$** / **INT**	SHIFT	
			REPT		

GOSUB E/PG	**RESTR** E/LN		CPU RESET
- D/CHR _	I/CHR	D/LN	I/LN

' 7 **LOG**	^ 8 **EXP**	**AND** 9 **COS**	**STR$** **INT**
+ 4 **POS**	- 5 **SQR**	* 6 **RND**	* x *
THEN 1 **CALL**	**NOT** 2 **FRE**	**STEP** 3 **INP**	**PEEK** - <
SPC(0 **ABS**	**LEN** . **SGN**	= = =	+ + +

Gary Dinsmore's

Creative Software

Presents

BOOK & SURVEY

BOOK is a programed text that can turn the Compucolor into a teaching machine. The keyboard is under complete control, so young fingers find it difficult to derail the program. Courses can be designed using an editor program, or I will adapt your material to BOOK, for use in your teaching arena. Comes with complete doccumentation manual that shows you how to set up the lessons and run the programs. Self testing freature allows you to judge your students' performance by the number of false starts made. Each correct response is reinforced, and wrong responses elicit helpful hints to put the student back on the right track.

BOOK isn't just for kids either. I currently have written two courses on Assembly language programing. An introduction course that teaches you about the 8080 CPU and the Compucolor Assembler. What the format of the instructions is like and a number of other very basic concepts to get the complete novice started. There is also an intermediate course that breaks down all of the op. codes into logical groups and tells you how to use them. A third course still being written goes into applications used by the Compucolor.

SURVEY is a program that will allow you to be your own polster. Place your Compucolor out in your place of business and invite your customers to take a confidential survey. Ask them what they think of your service, your product, your competitors. Ask them if they will vote for Regan again in 1984! You write the questions, the Compucolor records their responses as a percentage of persons responding to the survey

A complete doccumentaion manual will assist you in putting together your survey, or I will take your material and adapt it to SURVEY. Like BOOK, SURVEY has keyboard control to be fairly idiot proof. Text and questions are created using an editor program.

For Information

Write to:
Gary Dinsmore
Creative Software
Rt 3 Box 3216
Warren Oregon 97053

BOOK $29.95
SURVEY $29.95
Introduction to Assembly Assembly Language Programing $11.95
WORDY (word processor/editor) $29.95

Calendar Printer

by David B. Suits

Here's a program for a new year. It is written for the Okidata Microline 82A printer, but it would require only minor changes to work for any other printer.

```
 10 REM  ************************************************
 15 REM  *                                              *
 20 REM  *              CALENDAR PRINTER                *
 25 REM  *                                              *
 30 REM  *             FOR MICROLINE 82A                *
 35 REM  *                                              *
 40 REM  *             D. B. SUITS, 14 AL               *
 45 REM  *                                              *
 50 REM  *           ALGORITHM BASED UPON               *
 55 REM  *         "DAY OF WEEK" PROGRAM FROM           *
 60 REM  *         SOME COMMON BASIC PROGRAMS           *
 65 REM  *        (OSBORNE & ASSOCIATES, 1978)          *
 70 REM  *                                              *
 75 REM  *  NOTE: Color changes entered from the key-    *
 80 REM  *        board are given in brackets. Thus,    *
 85 REM  *        [17] is red, [18] is green, etc.      *
 90 REM  *                                              *
 95 REM  ************************************************
 97
 98 REM  Set up screen.
 99
100 CLEAR 200
110 GOSUB 9000
197
198 REM  Get starting year, etc.
199
200 GOSUB 8000
297
298 REM  Set up printer.
299
300 PLOT 27,18,4   :REM  Baud rate = 1200.
310 POKE 33289,255 :REM  Lots of room on terminal output.
320 PLOT 27,13     :REM  Output to printer.
330 PLOT 24        :REM  'CAN' = clear printer buffer.
340 PLOT 27,6      :REM  6 lines/inch.
350 PLOT 27,65     :REM  Long line.
360 PLOT 30        :REM  10 cpi.
370 PLOT 27,5      :REM  Top of form.
397
398 REM   Print the calendar.
399
400 GOSUB 1000
497
498 REM  More?
499
```

```
500 POKE 33265,0   :REM  Output to CRT.
510 PRINT :PRINT
520 INPUT "[19]WOULD YOU LIKE ANOTHER? ";A$
530 A$=LEFT$(A$,1):IF A$="Y" OR A$="O" THEN 200
540 IF A$()"N" THEN 520
549
550 END
997
998 REM  **********  Subroutine to print the calendar.

1000 IF YEAR(100 THEN YEAR=YEAR+1900
1010 M=STARTMONTH:Y=YEAR:IF STARTMONTH)2 THEN 1030
1020 M=STARTMONTH+12:Y=YEAR-1
1030 A=M+M+INT(.6*(M+1))+Y+INT(Y/4)+INT(Y/400)-INT(Y/100)+2
1040 COLUMN=FN M(7):REM  PRINTCOLUMN=A MOD 7
1099
1100 FOR INDEX=STARTMONTH TO STARTMONTH+NUMMONTHS-1
1110   A=(INDEX-1):MONTH=FN M(12)+1
1117
1118   REM  Top margin.
1119
1120   FOR J=1 TO 6:PRINT :NEXT
1127
1128   REM  Print month's name in double width letters.
1129
1130   PRINT TAB(MARGIN)CHR$(31)MO$(MONTH)CHR$(30):PRINT
1136
1137   REM  Print year AD (and AL for year ) 1968).
1138   REM  AL = Anno Lunae. (First moon landing was 1969.)
1139
1140   AD$="AD"+STR$(YEAR):AL$="AL"+STR$(YEAR-1968)+"    "
1150   PRINT TAB(MARGIN);:IF YEAR(1969 THEN 1170
1160   PRINT AL$;
1170   PRINT AD$
1179
1180   PRINT :PRINT :PRINT :PRINT
1190   PRINT TAB(MARGIN-5);
1197
1198   REM  Print names of days.
1199
1200   FOR J=1 TO 7
1210     PRINT SPC(7)DAY$(J);
1220   NEXT
1221
1230   PRINT
1237
1238   REM  Now a horizontal line.
1239
1240   GOSUB 7000
1249
1250   DAYS=DAYS(MONTH)
1257
1258   REM  Is it February?
1259
1260   IF MONTH()2 THEN 1290  :REM  No.
1267
1268   REM  Yes. Check for leap year.
1269
1270   A=YEAR
1280   IF (FN M(4)=0) AND ((FN M(100)()0) OR (FN M(400)=0)) THEN DAYS=29
1289
1290   FOR JJ=1 TO DAYS
1297
1298     REM  Vertical lines.
1299
1300     GOSUB 6000:PRINT     :REM  CR and LF.
1310     GOSUB 6000:PLOT 13   :REM  CR only.
1319
1320     PRINT TAB(MARGIN+10*COLUMN+11)RIGHT$(" "+STR$(JJ),3);
1330     COLUMN=COLUMN+1
1340     IF (COLUMN(7) AND (JJ()DAYS) THEN JJ=JJ+1:GOTO 1320
1347
1348     REM  End of week (or month).
1349
1350     PRINT :IF COLUMN=7 THEN COLUMN=0
1360     FOR L=1 TO 3:GOSUB 6000:PRINT :NEXT :REM  Vertical lines.
1370     GOSUB 6000:PLOT 13:GOSUB 7000
1380   NEXT
1381
1390   PLOT 12                                  :REM  Next page.
1400   IF MONTH=12 THEN YEAR=YEAR+1
1410 NEXT
1411
1420 RETURN
5997
5998 REM  **********  Subroutine to print the vertical divisions.
5999
6000 PRINT TAB(MARGIN-1);
6009
6010 FOR J=1 TO 7
6020   PLOT VBAR:FOR KK=1 TO 9:PLOT SPACE:NEXT
```

```
6040 NEXT
6049
6050 PLOT VBAR
6060 RETURN
6997
6998 REM  **********  Subroutine to print a horizontal line.
6999
7000 FOR J=1 TO MARGIN:PLOT SPACE:NEXT
7010 FOR J=1 TO 69:PLOT UNDERLINE:NEXT
7020 PRINT
7030 RETURN
7997
7998 REM  **********  Subroutine to get starting year, etc.
7999
8000 PRINT
8010 INPUT "[19]ENTER STARTING YEAR FOR THE CALENDAR: [17]";Y$
8020 YEAR=VAL(Y$):IF YEAR(0 OR YEAR()INT(YEAR) THEN 8010
8030 PRINT
8040 INPUT "[19]ENTER START MONTH (1 FOR JAN, 2 FOR FEB, ETC.): [17]";M$
8050 STARTMONTH=VAL(M$)
8060 IF STARTMONTH(1 OR STARTMONTH()INT(STARTMONTH) OR STARTMONTH)12 THEN 8040
8070 PRINT
8080 INPUT "[19]ENTER NUMBER OF MONTHS DESIRED: [17]";NUMMONTHS$
8090 NUMMONTHS=VAL(NUMMONTHS$)
8100 IF NUMMONTHS(1 OR NUMMONTHS()INT(NUMMONTHS) THEN 8080
8110 PLOT 15,18
8120 PRINT :PRINT
8130 PRINT "TURN ON PRINTER AND SET TO TOP OF PAGE."
8140 PRINT :INPUT "[19]PRESS [23]RETURN [19]WHEN READY: ";A$
8150 RETURN
8997
8998 REM  **********  Subroutine to get data.
8999
9000 DEF FN M(MD)=INT(A)-INT(MD)*INT(INT(A)/INT(MD)):REM  A modulo MD.
9010 DIM DAYS(12),MO$(12),DAY$(7)
9019
9020 DATA January,31,February,28,March,31,April,30
9030 DATA May,31,June,30,July,31,August,31,September,30
9040 DATA October,31,November,30,December,31
9050 DATA Sun,Mon,Tue,Wed,Thu,Fri,Sat
9059
9060 FOR J=1 TO 12:READ MO$(J),DAYS(J):NEXT
9070 FOR J=1 TO 7:READ DAY$(J):NEXT
9079
9080 PLOT 14,6,6,29,12
9090 PRINT TAB(15)"C A L E N D A R   P R I N T E R"
9100 PLOT 15
9110 PRINT TAB(20)"FOR MICROLINE PRINTER":PRINT
9119
9120 MARGIN=6            :REM  Left margin.
9129
9130 REM  Codes for PLOTting.
9139
9140 SPACE=32
9150 UNDERLINE=95
9160 VBAR=124
9169
9170 RETURN
```

Compucolor Transistor Equivalents

Reprinted by permission from CUVIC (Compucolor/Intecolor Melbourne User Group newsletter, November, 1982)

[We felt that this information was important enough to reprint, although it has been published in several other newsletters. - eds]

Here are some equivalent components for the high voltage transistors on the Compucolor analog board:

TRANSISTOR	I.S.C.	EQUIVALENT
Q1	BU500	BDX32
Q2	FT410	BUX81
Q3	MJE3439	2N5657
Q4	MJE3439	2N5657
Q9	A43A	BF336 or 2N5551 or MPSA43A
Video drivers	2N2905A	BFX30

In order to keep the voltage on most of these transistors below their limits, keep the width of the screen display such that there is about 1.5 cms border on each side. Because this squeezes up the characters, the brightness should be lowered a little.

Converting Screen Editor Files to Comp-U-Writer .DOC Files

by Joseph J. Charles
P.O. Box 750
Hilton, NY 14468

We have been using a screen editor as a poor man's word processor to create files for school reports, letters, and other documents. Recently my son started typing the rough draft of a new book I'm writing for the Timex/Sinclair computers. While the original intent was simply to get a legible rough draft, it became more desireable to do the whole job (through to camera ready copy) on the Compucolor computer. Therefore, after having my son type about seventy double spaced pages using a screen editor (CTE), I bought the COMP-U-writer word processor.

COMP-U-writer is designed to format disks and create files that are not compatible with normal FCS format and file structure. However, COMP-U-writer comes with enhancements to allow one to create and use files in regular FCS format. These files are customarily named with a .DOC file name extension. If you then try to load any file other than a COMP-U-writer FCS-compatible file, you will find that the first line of the text is garbage, and COMP-U-writer will not respond properly to commands. However, it is easy to make any file of ASCII characters compatible with COMP-U-writer.

COMP-U-writer creates a directory entry of 128 bytes for each file, whereas FCS directory entries are only 21 bytes. The additional space in COMP-U-writer directories allow 29 character file names. The directory also contains the values for a number of document parameters such as: number of characters per line, number of lines per page, baud rate, etc. When an FCS compatible file is created, these document parameters are still saved, but since there isn't room for them in the FCS directory, they are saved in the file itself in 64 bytes which are added to the beginning of the file. Thus when any type of file other then a .DOC file is used, the first 64 bytes of the text are taken as the document parameters. This is enough to foul things up since the parameters are now extremely unlikely to be correct. The solution is to simply add the correct values for the parameters as the first 64 bytes of the file. This could be done with a BASIC program or with FCS commands.

I must admit that I haven't taken the time to determine the meanings of all the bytes that are added to the file. Several bytes were zero for every file we created. However, we did not use every document option available. Knowledge of the meanins of the bytes will allow you to make the conversion easily, and, once converted, additional options could easily be set in the COMP-U-writer file. The bytes whose meanings we discovered are given in Table 1. (For two byte entries, the low byte is given first, then the high byte.)

I use the following procedure in FCS on my 32K machine to add COMP-U-writer type file parameters onto the front of a file created by a screen editor:

1. Use an existing .DOC file to get the first 64 bytes into memory:

   ```
   FCS>LOAD filename.DOC 8500
   ```

2. Load the file to be modified into memory 64 (40 hex) bytes after the start of the previous file:

   ```
   FCS>LOAD screeneditorfilename
           .SRC 8540
   ```

3. Save the file including the additional 64 bytes (you will have to calculate the new end of file address):

```
FCS>SAVE screeneditorfilename
        .DOC 8500-endaddress
```

A BASIC program to do the job is given in Listing 1.

Table 1

Byte	Description	Typical value	
1	lines/page	55	
2	"	0	
3	characters/line	60	
4	"	0	
5	left margin	8	
6	"	0	
7	indentation	5	
8	"	0	
9	baud rate	176	
10	"	4	4x256+176=1200
11	starting page number	13	
12	"	0	
13	1st tab column	1	
14	2nd "	9	
15	3rd "	19	
16	4th "	29	
17	5th "	39	
.	...	.	
.	...	.	
12+no. of tabs+1	255	255	end of tabs
" +2	0	0	
" +3	double space (0/255)	255	0=no; 255=yes
" +4	0	0	
" +5	cont forms (0/255)	0	
" +6	0	0	
" +7	two col/page (0/255)	0	
" +8	0	0	
" +9	just rt marg (0/255)	255	
" +10	0	0	
" +11	marked text (0/255)	0	
" +12	0	0	
" +13	page nums (0/255)	255	
" +14	0	0	
" +15	2-side print (0/255)	0	
" +16	0	0	
" +17	parallel port (0/255)	0	
" +18	0	0	
" +19	?	0	
" +20	?	0	
" +21	?	0	
" +22	?	0	
" +23	?	0	
" +24	?	0	
" +25	?	0	
" +26	?	0	
" +27	?	0	
" +28	255	255	
" +29	?	0	
" +30	?	0	
" +31	?	0	
" +32	?	0	
" +33	?	0	
" +34	?	0	
" +35	page width	85	
" +36	0	0	
" +37	text starts here		

Listing 1

```
  1 REM             [wht]PROGRAM: [red]CONVRT.BAS[grn]
  2
  3 REM  [wht]CONVERTS ANY ASCII FILE TO A [yel].DOC FILE FOR USE[grn]
  4 REM  [wht]WITH THE [yel]COMP-U-WRITER [wht]WORD PROCESSOR AND SAVES[grn]
  5 REM  [wht]THE [yel].DOC [yel]COPY ON DISK.[grn]
  6
  7 REM      [yel]WRITTEN BY J.J.CHARLES & J.J.CHARLES JR.[grn]
  8 REM             [yel]OCT 11, 1982  10:23  PM[grn]
  9
 10 REM  [cyn]POKE DEFAULT VALUES INTO MEMORY AT 9000[grn]
 20
 30 FOR I= 0TO 63:READ DV:POKE 36864+ I,DV:NEXT I
 40
 50 REM  [cyn]GET DIRECTORY INFORMATION FROM USER[grn]
 60
 70 PLOT 15,12
 80 INPUT "[grn]INPUT DISK DRIVE NUMBER: [red]";DN
 90 IF DN< > 0AND DN< > 1THEN 80
100 PLOT 6,6
110 PLOT 12,27,4:PRINT "DIR";DN;":":PLOT 27,27
120 INPUT "[grn]ENTER THE FILE NAME: [wht]";FI$
130 INPUT "[grn]ENTER THE FILE TYPE: [wht]";FT$
140 INPUT "[grn]ENTER THE VERSION NUMBER: [wht]";VN
150 INPUT "[grn]FILE SIZE FROM DIRECTORY [yel](4 DIGITS)[grn]:  [wht]";SZ$
160 INPUT "[grn]FILE LAST BLOCK COUNT [yel](LBC-2 DIGITS)[grn]: [wht]";LBC$
170
180 REM  [cyn]LOAD .SRC TYPE FILE AT 9040[grn]
190
200 PLOT 27,4:PRINT "LOA ";DN;":";FI$;".";FT$;";";VN;" 9040":PLOT 27,27
210 REM  [cyn]CONVERT NUMBER OF SECTORS TO DEC[grn]
220
230 N$= SZ$:GOSUB 5000
240 SZ= N
250
260 REM  [cyn]CONVERT NUMBER OF BYTES IN LAST SECTOR TO DEC[grn]
270
280 N$= "00"+ LBC$:GOSUB 5000
290
300 REM  [cyn]CALCULATE FILE SIZE IN DECIMAL[grn]
310
320 NN= (SZ- 1)* 128+ N
330
340 REM  [cyn]CALCULATE END OF FILE LOCATION[grn]
350
360 NN= NN+ 36927
370 PRINT :PRINT "FILE END (DEC) = ";NN
380
390 REM  [cyn]CONVERT TO HEX[grn]
400
410 GOSUB 6000
420 PRINT "FILE END (HEX) = ";N$
430
440 REM  [cyn]SAVE FILE AS .DOC TYPE[grn]
450
460 PRINT "[yel]INSERT THE DISK ON WHICH YOU WISH TO SAVE THE FILE[grn]"
470 INPUT "[yel]IN [wht]DRIVE 0 [yel]AND HIT RETURN[grn]";AZ
480 PLOT 27,4:PRINT "SAVE " CHR$(34);".DOC 9000-";N$:PLOT 27,27
490 INPUT "DO YOU WISH TO CONVERT ANOTHER FILE? ";AN$
500 IF AN$= "Y"GOTO 70
1000 END
4997
4998 REM  [red]*****  [wht]HEXADECIMAL TO DECIMAL SUBROUTINE  [red]*****[grn]
4999
5000 N= 0
5001 FOR I= 0TO 3
5002   N1$= MID$ (N$,4- I,1)
5003   N1= ASC (N1$)
5004   IF N1> = 48AND N1< = 57OR N1> = 65AND N1< = 70GOTO 5006
5005   RETURN
5006   IF N1> = 65THEN N2= N1- 55
5007   IF N1< = 57THEN N2= N1- 48
5008   N= N+ N2* 16↑ I
5009 NEXT I
5010 RETURN
```

```
 5997
 5998 REM [red]*****  [wht]DECIMAL TO HEXADECIMAL CONVERSION  [red]*****[grn]
 5999
 6000 N= NN
 6010 IF NN> = 0AND NN< = 65535GOTO 6030
 6020 N= - 1:PLOT 6,2:RETURN
 6030 FOR I= 3TO 0STEP - 1
 6040   FOR J= 15TO 0STEP - 1
 6050     T= J* 16↑ I
 6060     IF T> NNTHEN NEXT J
 6070   D(I)= J
 6080   NN= NN- T
 6090 NEXT I
 6100 N$= ""
 6110 FOR I= 3TO 0STEP - 1
 6120   IF D(I)> = 0AND D(I)< = 9THEN N$= N$+ CHR$ (D(I)+ 48)
 6130   IF D(I)> = 10AND D(I)< = 15THEN N$= N$+ CHR$ (D(I)+ 55)
 6140 NEXT I
 6150 RETURN
 9998
 9999 REM       [cyn]COMP-U-WRITER'S 64 DEFAULT VALUES[grn]
10000 DATA 55,0,56,0,8,0,0,0,128,37,1,0,255,255,0,0,0,0,0,0
10010 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
10020 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,0,0,0,0
10030 DATA 0,0,72,0
```

Keyboard Expansion

by Bill Anthony
655 E. Wells Way
Camano Island, WA 98292

This article explains how to expand a small Compucolor keyboard to a large one, a relatively easy task for anyone with a desire for a larger keyboard and a fondness for money. The tools required include a Phillips screw driver, a soldering iron, some solder, a desoldering tool, a drill, and a keyhole saw. You'll also need some keys with keytops to add to the board. The number will depend on the extent of your expansion. From smallest to largest size, 46 keys are needed.

If you live near a good surplus store, you may find the Cherry keys and appropriate tops there, or perhaps through a local electronics store such as Schweber. One good source for keys and tops is Arkay Engineers, 2013 Newbridge Road, Bellmore, New York 11710, phone 516-781-9859. Ken Kaplan is a good contact there.

With everything assembled, and with the computer **OFF**, disconnect the keyboard and turn it over onto a towel to prevent scratching. Remove the two small screws holding the cover to the keyboard, and slip the cover off the circuit board. Disconnect the cable leading to the keyboard, (mark it first so you can replace it the same way) and set all the parts aside except the circuit board.

With your soldering iron and desoldering tool (either wick or suction type), remove the solder from the holes of the keys you plan to add. The location of each key is marked on the bottom of the circuit board, so that shouldn't be too hard. Use caution, and don't overheat.

Next, insert the keys into the metal mask and very carefully push them all the way in. Watch that the leads go through the holes you have so patiently cleared and don't bend. Then solder the leads to the

Continued on page 26

Assembly Language Programming

by David B. Suits

Part VIII:
Simple Math

Rotating

The contents of the accumulator may be "rotated" left or right. A left rotation will move each bit one position to the left. The MSB (Most Significant Bit) will be moved around to become the new LSB (Least Significant Bit). For example:

Before left rotation: 1 0 0 1 1 0 1 0

After left rotation: 0 0 1 1 0 1 0 1

Another version of the rotate instruction involves the carry flag. The bit that is shoved off the end during the rotate instruction will replace the carry flag, and the bit that was in the carry flag will move into the other end of the accumulator. In effect, then, the carry flag acts as though it were the ninth bit of the accumulator. For example:

Before left rotation:
Carry: 0
Accum: 1 0 0 1 1 0 1 0

After left rotation:
Carry: 1
Accum: 0 0 1 1 0 1 0 0

As with some other 8080 mneumonics, the rotate and rotate-through-carry instructions appear to be what they are not. "RLC" does not stand for "rotate left through carry", and "RAL" does not stand for "rotate accumulator left". Nope. It's the other way around. Similarly for the right rotate instructions RRC and RAR.

If you don't want the bit from one end to "wrap around" into the other end of the accumulator, you will want to use a "shift" instruction. Unfortunately, the 8080 has none. Thus, you'll have to use a rotate instruction and then mask out the unwanted bit with an ANI instruction. For example, the sequence:

```
RLC
ANI  11111110B  ;or ANI 254
```

will shift all bits to the left, insuring that the LSB is zero. By the way, that will have the same effect as:

```
ADD  A
```

except that the flag settings might differ. And this brings us to the subject of addition.

Addition

The first thing to notice about addition is that it is done two numbers at a time. Even if you're adding up a whole column of numbers, you can do it by adding the first two numbers, then adding their sum to the next number, and so on. One consequence of this is that the largest carry during such an addition will be 1. Or, to put it another way, there is either a carry (i.e., 1) or there isn't (i.e., a "carry" of 0). This is not peculiar to binary arithmetic; it's true in all bases--decimal, hexadecimal, and so on. So if you add two 8 bit binary numbers, their sum cannot possibly be more than 9 bits.

Using the 8080 instruction set, there is an easy way to determine whether there has been a carry: the carry flag will be set. Of course, it is not always an easy task to extend the number of bits representing a number just because there has

been a carry. So in any given program we will decide in advance what the allowable range of numbers will be. If there is a carry out of that range, then you can have the program either ignore it or else signal an overflow error.

Suppose we wished to limit the range of numbers from 0 to 255. (That's convenient!) How do we carry on byte addition? Simple. Put one number in the accumulator and other other number in a register (including M, the pseudo register which is actually the byte of memory whose address is in HL). Then the instruction:

```
ADD  <reg>
```

will add the two numbers and leave the results in the accumulator. The carry flag will be set if the sum was too large to fit into one byte. For example:

```
MVI  B,6          B:   0 0 0 0 0 1 1 0
MVI  A,253        A:   1 1 1 1 1 1 0 1
ADD  B          sum: 1 0 0 0 0 0 1 1
```

The result in the accumulator is 3, which is actually 6+253 = 259 MODULO 256 = 3, and the carry flag is set. (The zero, sign and parity flags are also affected, but we're not concerned with them here.)

Multi-byte addition is easy. To add a one byte number to a two byte number, it's probably easiest to use a routine in ROM called ADHLA (at 3518H for V6.78 and 194EH for V8.79), which adds the number in the accumulator to the contents of HL. The result is in HL. (You might want to fire up your debugger and take a look at that ROM routine.)

A simple way to add a two-byte number to another two-byte number is to put one of them in DE (or else BC) and the other in HL. Then the instruction DAD D (or DAD B) will add them, with the result in HL. This "Double ADd" instruction is frequently used for adding offsets to pointers. For example, HL might be keeping track of something on the screen. To point up one line, 128 (for a 64 character line) must be subtracted from it. These assembly language instructions would do it:

```
LXI  D,-128
DAD  D
```

If the results of a DAD are greater than two bytes, the carry flag will be set.

If you want to add still larger numbers, you'll have to write a more complex--but still fairly simple--routine. Consult an 8080 assembly language book for some examples.

Since multiplication can be seen as repetitive addition, we shall advance to our next topic....

Multiplication

The ADD A instruction multiplies the contents of the accumulator by two. To multiply by four, simply ADD A twice. ADD A three times to multiply by eight. And so on by powers of two. How about multiplication which won't fit that scheme? Suppose you want to multiply the contents of the accumulator by six. First, notice this equality:

$$6*A = 2*A + 4*A$$

Since the factors 2 and 4 are powers of two, the multiplication by six can be accomplished by two multiplications and an addition. But each of those two simpler multiplications is easy.

```
ADD  A      ; *2
MOV  B,A    ;Save 2*A
ADD  A      ; *4
ADD  B      ; *6
```

Similarly, multiplication by, say, 13 can be expressed as:

$$13*A = A + 4*A + 8*A$$

And a routine to do that would be:

```
MOV  B,A    ;Save 1*A.
ADD  A      ; *2
ADD  A      ; *4
MOV  C,A    ;Save 4*A.
ADD  A      ; *8
ADD  C      ; *12
ADD  B      ; *13
```

Because of the possibility of a carry out of the MSB, your one byte result (in A) will not be correct if the result would have been greater than 255. Thus, the

routines above should be taken as MODULO 256. If you want to be able to obtain larger results, you'll have to use a two byte (three byte, four byte, etc.) routine. Also, the method I've presented can be used only when you already know the value of one of the numbers. But suppose you don't know either of them. There is some number in A and some number in B. How do you multiply them? Perhaps the easiest way is to use the ROM routine MULHD (at 3562H for V6.78 and 1998H for V8.97). This routine allows you to multiply a one or two byte number by another one or two byte number. Simply load DE with one number, HL with the other, and CALL MULHD. If a number has only one byte, then load it into the low byte of the register pair and set the high byte of the pair to zero. The result comes back as four bytes (two "words"), the high word (most significant two bytes) in DE and the low word in HL. If both numbers are only one byte, then the result is necessarily no more than two bytes. Thus, to multiply the contents of A by the contents of B, you could:

```
MOV  E,A
MVI  D,0
MOV  L,B
MVI  H,0
CALL MULHD
```

Since the low word is returned in HL, that will contain the answer.

Next time: Numerical I/O and random numbers.

Keyboard Expansion

(Continued from page 23)

circuit board, making sure the solder joint is well made. Connect the keyboard to the computer (without its case) and try it out to make sure the keys are properly installed. (The circuit board **must** be on a non-conductive surface.)

Stick on the keytops, and measure the cover to determine what must be cut. Mark it on the cover in pencil, take a break, and measure it again. Once cut, that's it. Cut the cover with your keyhole saw, or a jigsaw, if you (or a friend) have one, smooth the edges with sandpaper, and reassemble. Figure out where to spend the money you just saved.

Back Issues Sale

Back issues of **Colorcue** are an excellent source of information about Compucolor computers, ISC computers, and programming in general. Interviews, interesting articles, and programs are all there with a touch of history.

The list below includes every **Colorcue** ever published. If it's not on the list, then there wasn't one.

MULTI-ISSUES at $3.50 each

__ Oct, Nov, Dec 1978
__ Jan, Feb, Mar 1979
__ Apr, May/June 1979
__ Aug, Sept/Oct, Nov 1979

INDIVIDUAL ISSUES at $1.50 each

__ Dec 1979/Jan 1980
__ Feb 1980
__ Mar 1980
__ Apr 1980
__ May 1980
__ Jun/Jul 1980

INDIVIDUAL ISSUES at $2.50 each

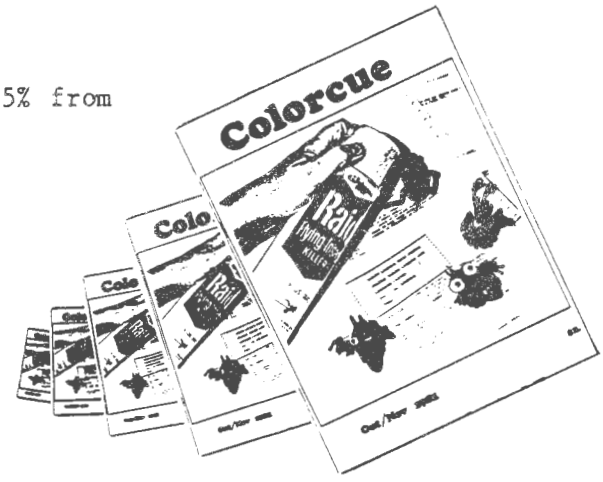
__ Dec 1980/Jan 1981
__ Aug/Sep 1981
__ Oct/Nov 1981
__ Dec 1981/Jan 1982
__ Feb/Mar 1982
__ April/May 1982
__ June/July 1982
__ Aug/Sep 1982

POSTAGE

US and Canada -- First Class postage included.
Europe, S. America -- add $1.00 per item for air, or
$ .40 per item for surface.
Asia, Africa, Middle East -- add $1.40 per item for air, or
$ .60 per item for surface.

DISCOUNT

For orders of 10 or more items, subtract 25% from total after postage.



ORDER FROM:

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14623

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

Colorcue

$9,587	$9,518	$10,253
19/25	16/25	17/26
684/900	352/550	315/481
8	8	6
120.1 in.	90.2 in.	91.2 in.
201.5 cu. ft.	87.9 cu. ft.	99.4 cu. ft.
225 Slant-6 95 @ 3600 170 @ 1200	5.0 L V8 150 @ 4000 240 @ 2400	5.0 EFI V8 130 @ 3200 240 @ 2000
39.6" x 47.2"	28.1" x 37.9"	28.9" x 37.0"
49.2" x 47.2"	48.2" x 28.7"	49.0" x 28.1"

1 800 874-4000

83 7409

1977 78 79 80 81 82 83 84 85

1.50

911 15 324

550

800

4780o 6:00

1500

79

1.50 1541 1000

924-2393
244-7243
424-5853
334-6415
436-0715
225-5002
244-0577
244-3979
352-3347
621-7566
473-2019
436-5446

28840

2025
2020
2050
2035
2030
2040
2060
2055
2015
2045

-$ 3,869,000
-$ 5,351,000
-$ 6,721,000
-$ 9,867,000
-$10,739,000
-$14,238,000

642 (1-49) 22015 40 11 200 95

37

79

Dec/Jan 1983 $2.

Colorcue

A bi-monthly publication by and for
Intecolor and Compucolor Users

DEC/JAN 1983
Volume 5, Number 3

Editors:
Ben Barlow
David B. Suits
Compuserve: 70045,1062

Advertisers: A good way to get in touch with potential customers is through the pages of **COLORCUE**. You will find our advertising policies attractive. Write for details.

Authors: This is a user-oriented and user-supported publication. Your articles, tips, hints, programs, etc. are required to make it go. Write or scribble your ideas down; we'll edit them and provide all artwork. Send your articles or write for information.

COLORCUE is published bi-monthly. Subscriptions are US$12/year in the U.S., Canada and Mexico, and US$24 (includes air mail postage) elsewhere. Some back issues are available. All editorial and subscription correspondence should be addressed to **COLORCUE**, 161 Brookside Dr., Rochester, NY 14618, USA. All articles in **COLORCUE** are checked for accuracy to the best of our ability, but they are NOT guaranteed to be error free.

Editors' Notes:

We goofed last issue with the date on the Contents page. Instead of "October /November" we had "August/September". The Volume and Issue number were correct, though. And so was the date on the front cover.

Say, are you going to send us an article on your latest hardware or software project, or are you going to let it languish forever? Articles (long, short, medium) are always in demand. How about it?

Have you been having troubles with ISC's Macro Assembler? Some users--even users very experienced in such things--have been miserably unsuccessful with it. The problem might well be not in the assembler itself but rather in the linking loader. Perhaps early versions were bugful? Let us know your experiences with the Macro Assembler, and we'll pass them on in these pages.

Speaking of compilers (we were speaking of BASIC compilers in the last issue), there's no word yet about the Australian BASIC compiler. We're holding our collective breaths. In the meantime, if you're interested in learning about compilers but haven't found a book which is readable by non-mathematicians, there is a very good book which you should have a look at. It is Compiler Design Theory by Lewis, Rosenkrantz and Stearn (Addison-Wesley, 1978). Don't let the title fool you; the authors don't inundate you with chapters and chapters of the formal theory of compilers. The book has a more practical feel to it. In fact, the object of the book is to develop a Mini-BASIC compiler. Perhaps with a bit of work (er...a lot of work) the compiler could be enhanced so as to be able to compile something akin to Microsoft BASIC for your Compucolor/Intecolor. Well, it's a thought, anyway.

A question from a reader: Does anyone know of a spelling correction and/or a proofreading program that will run on a Compucolor? (For ISC CP/M owners, the Word Plus programs from Oasis systems are excellent. - ed.)

Another question. A reader having difficulties getting anything to run on a cold Compucolor uses a hair drier to warm the disk drive. After about a minute, all is fine. Can anyone suggest a solution that would eliminate the problem?

Readers with an interest in educational software will be interested in a set of programs written for ISC machines by Dr. Marjorie A. Fitting to teach trigonometry. Using a circular functions approach, the programs provide experience with radian measure, the development of the sine function, graphing the sums of functions, and drills with identities and polar graphs. We have not personally seen the programs, but they received an excellent review in a recent issue of Mathematics Teacher which rated accompanying documentation as above average. The individual price is just $29.95; school and dealer prices are somewhat higher. Contact METIER, PO Box 51204, San Jose, CA 95151.

Tech Tip

by Alexander Pinter
PO Box 230
Columbus, GA 31902

A few weeks ago I bought a used disk drive for my 6.78 CCII and figured out how to change it from CD0: to CD1:. Here's how:

In the right rear of the disk controller board, to the left of the DIP socket used by internal drives, are five holes numbered one through five. CD0: drives have hole one connected to hole five, and CD1: drives have a small loop of wire connecting hole five to hole two. Just solder in the loop of wire to holes five and two, and cut the circuit board trace between holes five and one.

INTELLIGENT COMPUTER SYSTEMS INC.

UNDERSTANDABLE SOFTWARE ******RELIABLE HARDWARE
GOOD SUPPORT **** AND ALWAYS DISCOUNT PRICES**

NASHUA DISKETTE

* single sided * double density * 40 tracks * soft sector * hub ring *
* * AND free exchange on all defective diskettes within 30 days * *

10 blank diskettes	US$ 26	10 formatted diskettes	US$ 31
20 blank diskettes	US$ 50	20 formatted diskettes	US$ 60

SPECIAL

FORMATTER WITH SPEED CONTROL program
Formats your CCII diskette, and displays the speed of the diskdrive graphically on the screen ORDER# SP845 US$ 24

EXTERNAL HOUSING FOR DISKDRIVE
complete with cable and detailed instructions (only 6 left) ORDER# IM790 US$ 79

COMPUCALC
The Superversion of the Visicalc for the CCII ORDER# BG115 US$ 120

SOFTWARE CATALOG WITH OVER 150 PROGRAMS AVAILABLE ON REQUEST
** SEND US$1 FOR COMPLETE SOFTWARE AND HARDWARE CATALOG **
*** MASTER CHARGE, VISA AND AMERICAN EXPRESS ACCEPTED ***

INTELLIGENT COMPUTER SYSTEM, 12117 COMANCHE TRAIL
HUNTSVILLE, AL 35803 USA, PHONE 205-881-3800

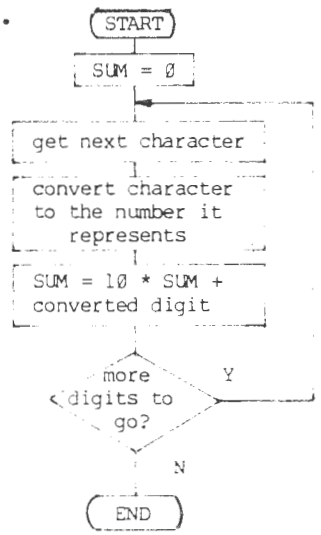
Assembly Language Programming

by David B. Suits

Part IX: Numerical I/O and Random Numbers

The computer handles numbers in binary. The computer **user** handles numbers in decimal (usually). How do we make the conversion? For a single decimal digit the conversion is simple. The ASCII codes for the digits '0'--'9' are 30H--39H. Thus, translating from ASCII to binary is simply a matter of subtracting 30H. And to translate from binary to ASCII, just add 30H. This latter conversion won't work if the binary number is greater than 00001001. Suppose, for example, the accumulator contains 00010001 (=11H). Adding 30H would yield 41H and sending that code to the screen would produce the letter 'A'. So, to deal with numbers having more than one decimal digit, a more complex routine is required. That is, we want a routine which will accept decimal characters from the keyboard and convert them to appropriate binary equivalents for internal manipulation.

Figure 1.



As usual, many assembly language routines can be invented by trying first to write a BASIC routine. How would you convert a string of digits into a number if you didn't have BASIC's VAL() function? Figure 1 shows a flowchart for such a method, and Listing 1 gives a BASIC subroutine. This is not too difficult to do in assembly language, as long as we limit the number to two bytes maximum (65535 decimal). Let's assume that the string of digits has been read into a buffer. The BC register pair points to the start of the buffer, and the end of buffer is signalled by a carriage return (0DH). Listing 2 shows the routine.

Next is the problem of translating a number from binary to a string of ASCII digits for printing. More or less the reverse of the preceding routine will do the job: keep **dividing** SUM by 10 and converting the remainders into ASCII digits until SUM=0. Listing 3 is one version of such a method. It is based on a routine by Graeme Smith in the March, 1980 (V3, No.3) issue of **Colorcue**. The number is translated into a string of

Listing 1.

```
997 REM  Subroutine to convert string of
998 REM  digits to a number.
999 REM  Input string is D$.
1000 SUM=0
1010 FOR J=1 TO LEN(D$)
1020   A$=MID$(D$,J,1)
1030   A=ASC(A$)-48
1040   SUM=10*SUM+A
1050 NEXT
1060 RETURN
```

ASCII digits which are moved into a buffer, one at a time, from right to left. This is a very handy routine because you can first clear the buffer with space characters (20H) so that by printing the contents of the entire buffer, the number will be right justified and with no leading zeros. The second method, in Listing 4, is a slight modification which stores the string of digits on the stack and then prints them by POPping them off the stack one at a time. This is a simple, all-in-one routine for printing out a binary number in decimal.

Random Numbers

In BASIC we have that nifty RND() function. There's no such beast available in assembly language, and so we'll have to create our own. Really good pseudo-random number generators produce a series of numbers evenly distributed over a certain range. Unfortunately, they are also often complex to implement in assembly language. For some time I searched around for a quick and dirty routine, but I was never satisfied with what I found, mainly because the series of numbers generated by such routines would begin to repeat much too quickly, and so a pattern would be apparent. It finally occurred to me that my computer's ROM consisted of about 16K bytes of mixed up numbers. Couldn't we use them as a large table of pseudo-random numbers and just pick them out, one by one? Well, almost. The trouble is that some numbers in the range 0-255 are for some reason not very well represented in ROM, and other numbers appear much too often. So my pseudo-random number generator takes the contents of ROM and applies a very simple function to them to generate a more evenly distributed series of numbers (all in the range 0-255). This has the virtue of speed. In addition, I don't think I've seen the series of generated numbers

Listing 2

```
;************************************************************
;
; A2BIN -- Translate string of ASCII characters
;       to a two byte (maximum) binary number.
;
;       ENTRY:  <BC> -> start of string.
;               String must end with carriage return (ØDH).
;
;       EXIT:    HL  = converted number (two bytes).
;               <BC> -> end of string.
;               All else lost.
;
;       CALLS:  MULHD, ADHLA
;
;************************************************************

CR      EQU     ØDH     ;Carriage return character.
MULHD   EQU     3562H   ;V6.78 ROM routine to multiply HL by DE.
                        ;  High word of result ends up in DE,
                        ;  low word in HL. BC is unchanged.
                        ;N.B.: For V8.79 and V9.8Ø use
                        ;  MULDH   EQU   1998H
ADHLA   EQU     3518H   ;V6.78 ROM routine to add A to HL.
                        ;N.B.: For V8.79 and V9.8Ø use
                        ;  ADHLA   EQU   194EH

A2BIN:  LXI     H,Ø     ;SUM=Ø.

A2BIN1: LDAX    B       ;Get next digit.
        CPI     CR      ;End of string?
        RZ              ;Yes.
        SUI     3ØH     ;ASCII to binary.
        PUSH    PSW     ;Save new digit.
        LXI     D,1Ø
        CALL    MULHD   ;SUM=1Ø*SUM.
        POP     PSW     ;Retrieve new digit.
        CALL    ADHLA   ;SUM=SUM+newdigit.
        INX     B       ;Bump string pointer.
        JMP     A2BIN1  ;Back for more.
```

repeat, even though I've generated several hundred thousand of them in a row! Listing 5 is my pseudo-random number generator. If you wish to generate numbers greater than 255, just call RANDM for each byte desired.

Try generating two byte pseudo-random numbers and printing the results in decimal using the B2ASB or B2ASP routine. Have fun.

Now that we are able to generate random numbers in the range 0-255, how can we arrange things to get a random number in any given subrange within 0-255? That is, how do we get a random number, R, such that it is equal to or greater than some lower bound, L, and equal to or less than some upper bound, U?

(1) Calculate D=U-L.
(2) Generate a random number, R', such that 0<=R'<=D
(3) Then let R=L+R'.

This makes things a bit easier, since the random number generator will always give a number =>0. But how do we make sure it is <=D? A quick and dirty method is to subtract D from the R' until R' is equal to or less than D. Figure 2 is a flow chart of the process, and Listing 6 is a BASIC routine to further help you visualize the method. Listing 7 is the final version.

Listing 3.

```
;************************************************************
;
; B2ASB — Convert two byte binary number and put string of
;       ASCII digits in a buffer for later printing.
;
;       ENTRY:  <BC> -> right-most spot in print buffer.
;               <HL> = binary number to convert.
;
;       EXIT:   <BC> -> left of left-most char in print buffer.
;               <HL> = Ø.
;               All else lost.
;
;       CALLS:  DIVHD
;
;************************************************************

DIVHD   EQU     3581H   ;V6.78 ROM routine to divide DE by HL.
                        ;  Quotient is returned in HL, remainder
                        ;  in DE. BC is unchanged.
                        ;N.B.: For V8.79 and V9.8Ø use
                        ;  DIVHD   EQU   19B7H

B2ASB:  LXI     D,1Ø
        XCHG            ;Required by ROM routine.
        CALL    DIVHD
        MOV     A,E     ;Get low byte of remainder.
        ADI     3ØH     ;Binary to ASCII.
        STAX    B       ;Put character into buffer.
        DCX     B       ;Move buffer pointer left.
        MOV     A,L     ;Continue
        ORA     H       ; until quotient
        JNZ     B2ASB   ;  is zero.
        RET
```

Listing 4.

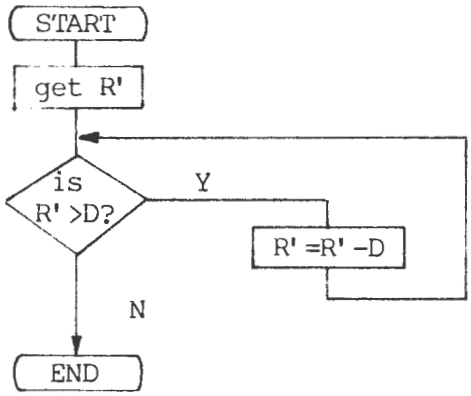
```
;************************************************************
;
; B2ASP — Convert two byte binary number to string of ASCII
;       characters and print the resulting string.
;
;       ENTRY:  <HL> = binary number to convert and print.
;
;       EXIT:   <BC>  unchanged.
;               <HL> = Ø.
;               All else lost.
;
;       CALLS:  DIVHD, LO
;
;************************************************************

DIVHD   EQU     3581H   ;V6.78 ROM routine to divide DE by HL.
                        ;  Quotient is returned in HL, remainder
                        ;  in DE.  BC is unchanged.
                        ;N.B.: For V8.79 and V9.8Ø use
                        ;  DIVHD   EQU   19B7H.
```

```
LO      EQU     3392H   ;V6.78 ROM routine to send character in
                        ;  A to screen.
                        ;N.B.: For V8.79 amd V9.80 use
                        ;  LO   EQU   17C8H.
B2ASP:  LXI     D,0
        PUSH    D       ;Zero on stack to indicate end of string.
B2ASP1: LXI     D,10    ;Divide by 10.
        XCHG            ;Required by ROM routine.
        CALL    DIVHD
        MOV     A,E     ;Get low byte of remainder.
        ADI     30H     ;Binary to ASCII.
        PUSH    PSW     ;Add to string.
        MOV     A,L     ;Continue
        ORA     H       ;  until quotient
        JNZ     B2ASP1  ;  is zero.

        ;Now print out the ASCII string thus created.

B2ASP2: POP     PSW     ;Get character.
        ORA     A       ;End of string?
        JZ      XB2ASP  ;Yes.
        CALL    LO      ;No. Print the character.
        JMP     B2ASP2  ;Back for more.

XB2ASP: RET
```

Listing 5.

```
;************************************************************
;
; RANDM -- Generate a pseudo-random number (0-255). Given a
;       one byte seed, multiply it by 17 (or any prime). Add
;       to that the byte at location ROMPTR, which is a pointer
;       to somewhere or other in ROM. (It could point into
;       program memory just as well, since this routine does
;       not alter the contents of the location at ROMPTR.)
;       Each time RANDM is called, the pointer is incremented
;       (and the one byte pseudo-random number is added to the
;       low byte of ROMPTR). If the pointer passes the end of
;       ROM, it is reset to the beginning.
;
;       ENTRY:  No register values expected.
;
;       EXIT:   <A> = pseudo-random number.
;               All else preserved.
;
;       CALLS:  None.
;
;************************************************************

PTRMIN  EQU     0000H   ;Start of ROM.
PTRMAX  EQU     40H     ;High byte of end of ROM (4000H).

RANDM:  PUSH    B       ;Save working registers.
        PUSH    H
        LDA     SEED
        MOV     B,A
        ADD     A       ;SEED=SEED*17 (ignoring any overflow).
        ADD     A
        ADD     A
        ADD     A
        ADD     B
        LHLD    ROMPTR
        ADD     M       ;Add some byte or other.
        STA     SEED    ;Result is pseudo-random number and
                        ;  seed for next time.
        MOV     B,A     ;Temporary.
        INX     H       ;Bump pointer.
        ADD     L       ;Mix it up a bit.
        MOV     L,A
        MOV     A,H     ;If pointer goes too far
        CPI     PTRMAX  ;  then
        JC      RANDM1  ;  reset it
        LXI     H,PTRMIN ; to beginning.
RANDM1: SHLD    ROMPTR
        MOV     A,B     ;Un-temporary.
        POP     H       ;Restore register.
        POP     B
        RET             ;Return with <A> = pseudo-random number.
```

Figure 2.



Listing 6

```
10 PRINT
20 INPUT "Upper bound: ";UP
30 INPUT "Lower bound: ";LO
40 DELTA = UP - LO
50 IF DELTA > 256 THEN PRINT "Range too big!": GOTO 10
60 RAN = INT(256 * RND(1))
70 IF RAN > DELTA THEN RAN = RAN - DELTA: GOTO 70
80 RAN = LO + RAN
90 PRINT "RANDOM ="RAN
100 GOTO 10
```

Listing 7

```
;*************************************************************
;
; BOUND -- This routine creates a pseudo-random number between
;       two bounds, each of which is 0--255.
;
;       ENTRY:  <B> = upper bound.
;               <C> = lower bound.
;
;       EXIT:   Required number in <A>.
;               B, C, E, H and L unchanged.
;               D lost.
;
;       CALLS:  RANDM
;
;*************************************************************

BOUND:  MOV     A,B     :Get upper bound.
        SUB     C       ;Subtract lower bound.
        MOV     D,A     ;Difference in D.
        CALL    RANDM   ;<A> = 0...255.
BOUND1: INR     D
        CMP     D       ;Is (difference + 1) > random #?
        JC      BOUND2  ;Yes. Just what we want.
        DCR     D
        SUB     D       ;Random = random - difference.
        JMP     BOUND1
BOUND2: ADD     C       ;Add lower bound.
        RET             ;Return with result in <A>.
```

First Aid for Compucolor Disk Drives

by Thomas J. Herold

Is your disk drive giving you error messages, particularly when you are loading a new disk for the first time? Does it fail to read or write as you think it should?

In almost all cases, these problems are caused by a mis-match between the current disk drive speed and the original speed used to record the disk. Both the internal and external disk drives for the CCI are notorious for changing speed without prior notice. For this reason, the speed of all CCII disk drives should be checked and adjusted as necessary, at least once a month. The allowable operating range is only 299 to 300 RPM. If your disk drive's speed is out of this range, you will have problems loading programs, exchanging disks with other CCII owners and possibly even reading your own disks.

Both the internal and external drives can be adjusted visually, by turning the potentiometer on the disk drive's internal printed circuit board while watching the stroboscopic indicator lines on the bottom of the disk drive rotor (under flourescent light, not sunlight) in much the same way a stereophonic turntable is adjusted to rotate at exactly 33-1/3 r.p.m. This method provides reasonable accuracy if performed carefully.

For the internal disk drive, it is necessary to remove the back housing from the computer. The potentiometer can then be adjusted by using a small screwdriver, and the indicator lines can be watched by using a small mechanic's mirror; or the disk drive can be dismounted from the computer frame and allowed to sit beside the computer during the adjustment. Be extremely careful when working with a screwdriver on a powered-up computer. Don't touch anything else.

There is a more reliable way of keeping your disk drive speed properly adjusted. This is with the FORMATTER program. which allows you to check the speed of your drive at any time, graphically, right on the screen, without removing or dismounting anything. Of course the drive will still have to be dismounted and adjusted with a screwdriver if the speed control program indicates that the drive speed is outside the allowable operating range. This same FORMATTER program also allows you to format and initialize your disks.

There is also a much easier way to gain access to and adjust your internal drive. Since an external drive is considerably easier to adjust than an internally mounted drive, many CCII owners have purchased an external disk drive housing, dismounted their internal drive from the computer, and installed it in the external housing where it is very easy to adjust. These factory-provided metal and plastic housings provide protection from physical damage, electronic radiation, spilled liquids and dust, and come complete with LED and factory ribbon cable to plug into the computer. You can order a complete housing from Intelligent Computer Systems in Huntsville, Alabama. The housing comes with good documentation on how to take the drive out of the unit and install it into the housing. The price is $79 US, and the FORMATTER with Speed Control is available for $24 US. Good luck, and be careful!

Some Thoughts on BASIC Speed and Style

by Joseph Norris
Apartment 5B
42 Conshohocken State Road
Bala Cynwyd, PA 19004

One may speak about two distinct elements of "speed" in the performance of a program. The first is the finite, measurable time required to complete an operation. The second is an "apparent speed", which coincides with the operator's subjective response to the timing of console events, especially time during which the operator is inactive. Helpful material is available concerning the first element and I recommend two references in particular which cover generalities in Microsoft BASIC [1].

As for the second element, consider what it means to format a program for minimum "apparent" speed. If a screen display is changing or the operator is required to make keyboard entries, time passes rather smoothly. And while it seems no great trial to wait for an occasional disk read/write, to the operator processing scores of files a day even a small wait can be most frustrating. You will be made aware of your profound reaction to this waiting period if you have experienced the difference between access times on a 5" and 8" disk system. The 8" disk will access twice as fast, but it can seem many times faster on first experience.

The "operator wait state" can be minimized simply by placing screen and keyboard activity as close as possible to the disk read/write period. This way the operator will be idle (visually and manually) for the least amount of time. Even "busy work" operations are helpful, such as screen printouts announcing the beginning and end of a disk access, or an in-process report, if the disk period is very long. Good sense predicates that such activities not significantly increase the waiting period; a simple message will do.

There are times when lengthy calculations must be performed. These calculations probably need not all be done in one great lump. There can be ample opportunity to intersperse calculations with screen messages and keyboard inputs, even if these are not concerned with the specific calculations taking place. Such activity might be concerned with something to be handled later, whose inputted data can be temporarily stored until needed. When my programs involve lengthy calculations I at least give the operator some idea of what is happening; "CALCULATING WEEKLY TOTALS", or "FORMATTING DATA FOR DISK WRITE" are typical screen messages. These are infinitely better than holding a stagnant screen with a blinking cursor (which office personnel tell me is simply annoying), because they allow the operator some measure of time remaining for the procedure to finish. If the idle period is longer than five seconds, I display a blinking "WAIT" or a similar message to remind the operator that the computer isn't broken! These things are effective in reducing mild anxiety when "nothing is happening" and are, after all, a courtesy on the operator's behalf.

The graphics capabilities at our disposal with ISC hardware make it difficult not to splurge occasionally with pyrotechniques that are spectacularly unnecessary and distracting (to everyone but the programmer). Such displays use time and, unless they are truly helpful, can appear more as self-indulgence than skill. We can strive for the level of sophistication that leads us, with wisdom, past these temptations and into a deeper sense of what creative programming

means. Examination even of simple programs, with creative imagination and awareness of the psychological element, will show opportunities for program refinement at any level of programming skill.

A time-consuming operation in protracted programs is the pressing of the RETURN key. I make it a rule that all single key responses will not require a subsequent RETURN. COLORCUE has had several articles describing ways to avoid RETURN. Listing 1 shows some lines of code that function as OPTION LINE directives demonstrating one such procedure and I invite you to examine it on your computer. (The '<' marking in my programs indicates to the operator a single key response. Multiple-key responses followed by RETURN are indicated by '>'.) Line 50 sets the position and color scheme of the Option line. The routine at 56 moves the cursor off the screen and gets the single-key response. Line 504 screens the response for validity, and line 506 vectors the option selection. Line 58 isn't necessary really; it converts the ASCII value returned from KB to the number printed on the line generated on 502.

The next thing we will consider in example demonstrates three techniques for speed; PLOTing and PRINTing, GETting and PUTting disk file strings, and the minimum use of variables. Figure 1 shows two lines from a screen display. The first is generated by the BASIC program and consists of field labels. The second is an intact data string, J$(1), of 64 bytes from a file record, printed exactly as it exists in the file. (A third line marker for byte counting is provided for your convenience.) Notice that each field length is separated from the next by two blank spaces, which are 'wasted', reserved spaces both on the screen and in the disk file. The GET statement for the string in File #1 is: GET 1,1,1;J$(1)[64]. This string, with all its data, PLOTs and PRINTs in a single line. To enter, delete or edit data I need but one variable pair which I will call S$ and S. To extract the Stock Number I GET 1,1,8;S$[5]: S=VAL(S$) and replace changed data with an equivalent PUT statement. If I have 'declared' S$ and S early in my program, they will be

Listing 1.

```
[Subroutines, located on early program lines]

50 PLOT 6,35,3,0,5,11,3,0,5: RETURN
55 POKE KB,0
56 PLOT 6,0,3,65,5: B = PEEK(KB): IF B = 0 THEN 56
58 B = B - 48: RETURN


[Main program reference to subroutines]

500 GOSUB 50
502 PRINT "1 DIRECTORY  2 ADD  3 DELETE  4 EDIT  5 PRINT <"
504 GOSUB 55: IF B < 1 OR B > 5 THEN 500
506 ON B GOTO 900,100,200,300,400


[GOTO references contain sub-programs]

100 REM  'ADD' ROUTINE: ENDS WITH GOTO 500
200 REM  'DELETE' ROUTINE: ENDS WITH GOTO 500
300 REM  etc.
--------
900 LOAD "DIRECT": RUN
```

Figure 1.

```
PROD#  STOCK  BKORD  VENDOR     DATE/DUE  CONTACT   TELEPHONE
12345  10005  25000  APEX CORP  12/10/83  A. MAHNS  123-456-7890


Field Specifications, String J$(1)

     Label: PROD# STOCK BKORD VENDOR DATE/DUE CONTACT TELEPHONE
    Length:   5     5     5     9       8       8        12
Start byte:   1     8     15    22      33      43       53
```

near the start of variable space and will be found quickly. I will use S$ (and S if required) to access any of the variables in this string.

Compare this procedure with GETting, PLOTting and PRINTing seven isolated variables, both in time of execution and memory space required. The PUT statement performs a string length adjustment as well. The VENDOR field is nine bytes long, but if my entry is only four bytes (i.e., APEX) I can still PUT 1,1,22;S$[9] which will write my four byte string plus enough space to make up the indicated byte count. This eliminates the pre-formatting of the string, and the necessity for clearing out old field data in the disk file.

When I change data in this way, I follow it with a FILE "D",1 statement and GET, PLOT and PRINT the string again, so my screen shows the updated content. A 'dummy' variable, N$[2], may be used to extract all of the variables in J$(1):

```
GET 1,1,1;ST$[5],N$[2],BK$[5],
          N$[2],VE$[9],N$[2], etc.
```

There is a consequence of this simple demonstration which is not at all trivial. I am really using GET and PUT for string manipulation in the manner one might use LEFT$, MID$ and RIGHT$, only the procedure is much more rapid. In fact, nearly all my longer industrial programs open a dummy file, DUM.RND[1,128,1] which serves no other purpose than to provide me with free file buffer space which is used to format strings, using PUT and GET. This technique has been especially valuable when the string being edited has had complex color coding, with various background and foreground combinations. These codes need not be considered while I plug in new data at the appropriate byte locations, and the economy in code is significant. A completed string is retrieved from DUM.RND and PUT into the proper file before closing it. In this manner I have been able, with color, to fill the entire screen with data, all needed at once to be meaningful. You might say I have, in effect, an 80 column display.

Another extension of this technique is the realization that there is no benefit from GETting all my 64-byte data strings which I only want to work with a few at a time. This allows multiple use of the necessary variables involved. Sorting procedures and the like may require access to all the data, but more often this is not the case.

We have looked at a few psychological implications in programming and at the advantages of using file buffer space for string manipulation. Though the presented examples are over-simplified, they may serve as suggestions for a more complicated implementation. I confess that BASIC remains my favorite high-level code, speed limitations notwithstanding, and if the proposed increase to 15 megahertz and 15 gigahertz speeds are realized in the next fifteen years, these limitations may become largely academic. In the meantime, creative programming will help realize the full potential of this splendid computer language.

[1] Tesler, Glenn. BYTE, May 1982, pp. 318, 328, 330, and APPLESOFT PROGRAMMING MANUAL, 1978, pp. 118-120. Most material covered here is applicable to CCII and 3651 equipment.

FLASH --- STOP THE PRESSES --- FLASH --- STOP THE PRESSES

A BASIC compiler has just been released by Peter Hiner in England. It is called FASBAS and generates what appears to be a pseudo-compiled BASIC which runs faster than interpreted code. We've had trouble getting it to access disk (e.g., PLOT 27,4), and there are certain restrictions on the kinds of BASIC programs it can translate, but it **does** work. What's more, the price for the disk and documentation is $25 outside the U.K. (Discount for bulk orders from user groups.) Contact Peter Hiner, 11 Penny Croft, Harpenden, Herts, AL5 2PD England. We'll try to say more about FASBAS in the next **COLORCUE**.

Compucolor Hardware Options

- ★ LOWER CASE Character set. (Switchable) MSC12 — $29 U.S.
- ★ MULTI-CHARACTER sets. (Lowercase, Electronic, Music etc.) — $39
- ★ REMOTE DEVICE CONTROLLER. Switch ON/OFF 8 devices. PSC1 — $45
- ★ 16K RAM Upgrade. (Increase from 16K to 32K.) — $99
- ★ ROMPACKS. Easy exchange of 8K EPROM MODULES. The interface board can hold an additional 8 or 16K EPROM.
  - Interface board, cable and ROMPACK socket: $48
  - Each blank ROMPACK (Including EPROMS): $25

  (Please write for full details of ROMPACK system and available software.)

PPI

PROGRAM PACKAGE INSTALLERS,
P O Box 37,
DARLINGTON,
WESTERN AUSTRALIA 6070

All prices incl. airmail.

★ EPPS Word Processor V5.2 ★

for the COMPUCOLOR II (V6.78, 8.79), 3621 and INTECOLOR 3651. (32K RAM)

only $55 (U.S.) Incl. airmail

* Full screen, fast operation with 20K byte buffer. (Assembler written)
* Can be used with any level keyboard. (101 key is recommended.)
* Automatic word wrap on screen and printer with justification. (30-199 col)
* Block and character Move, Copy, Delete, Save and Print.
* String search with optional replace. (Both up and down file.)
* Operates with or without lowercase character set. (Selectable).
* HELP facility. Full command summary on screen. (Two pages.)
* Automatic repeat on all keys.
* Imbedded control codes allows operation of any printer function.
* Screen preview of printout at any time.
* Compact file storage in FCS format. Can process existing .SRC files.

PPI

PROGRAM PACKAGE INSTALLERS,
P O Box 37,
DARLINGTON,
WESTERN AUSTRALIA 6070 (Ph.092996153)

Please include payment with order.

★ ★

What's New for the CCII?

by Rick Taubold
197 Hollybrook Road
Rochester, NY 14623

There's been a lot of 'doom and gloom' about the CCII lately, about how it has passed the way of many other obsolete items. Personally, I can't understand all the fuss. The CCII still lives on as the Intecolor line. All of us realize what a bargain the CCII was at the time. Granted, the Intecolors are a bit more expensive, but then you get more. I suppose what I'm saying is that, if the CCII is really dead, why has there been so much recent growth in software and hardware for this 'dead' machine? Or maybe you haven't noticed.

I know of at least three good sources of software plus several smaller ones. The most complete is Intelligent Computer Systems, 12117 Comanche Trail, Huntsville, Alabama 35803. They carry nearly every decent piece of software ever written for the CCII/Intecolor computers. Service is good, prices are low, and there are only occasional goof ups with orders. A second source is COM-TRONICS CO., 144 Cloverside CT., Buffalo, New York 14224. Again the service is good, the quality is very high. However, and I am not alone in this feeling, the prices are also rather high. COM-TRONICS' own words are that "if you want quality, you gotta pay for it." Perhaps this is true, but, despite the quality, I can't really say that what I've seen of their software is a bargain. In my opinion there is other software, just as good, for less money in many cases. By the way, unlike the other sources listed here, COM-TRONICS markets only their own software. Their products are also available from the other sources, but the prices are the same. I will probably hear from COM-TRONICS on this one, but I know several other people who agree with me.

Perhaps the most exciting new source of CCII and Intecolor add-ons is FREPOST Computers, 431 East 20th Street, New York, NY 10010. It is to this source that I will devote most of the remainder of this article. Have you ever wished that someone would make a particular item and then later find it available even better than you had dreamed of? I must confess that I came up with the idea of doing something useful with that blank 8K ROM space in the CCII some time ago. Unfortunately, I was still a novice and lacked the resources to carry out my idea. I thought it would be nice to be able to put several switchable programs permanently into my CCII memory in the 4000H-5FFFH ROM space. Programs like the Screen Editor, Assembler, FREDI (this was before 'The' BASIC Editor came along), etc. I envisioned a set of boards mounted inside or outside the computer with a switch to select the desired board. I was told by those more knowledgeable than myself that this was not feasible, something about the length of the data lines. Then, lo and behold, some months later I saw a demo of the new FREPOST Bank Select ROM board. I was awed. This was exactly what I had wanted. Further, it was under software control, no clumsy switches, and it plugged neatly inside the CCII. Well, I didn't buy one on the spot, but I did not wait long. I am pleased to say that there is no piece of hardware, in my opinion, currently available for the CCII that is more worthwhile having. The price of $250 assembled is a bargain. The unit is quality constructed and easy to install.

There's more. Not only will this beauty hold seven sets of 8K programs (EPROM chips), but the recent Tom Deviin RAM board will plug into the 8th position. That's a total of 64K of user-defined program space that is in addition to the 32K RAM already in your machine.

The possibilities are endless. Consider how awkward it would be to have to load BASIC from tape or disk every time you wanted to use it. That's what one had to do with the earlier home microcomputers (and still a few current ones, I think). All we have to do is ESC E and there it is. How would you like your favorite utilities that handy, always there at power-up time? The FREPOST system is the answer. All it takes is a couple of key strokes and away you go. Already there are many programs available in EPROM ready to plug in. FREPOST will even custom program EPROMs for you at modest cost.

It is also worth saying that FREPOST makes a single ROM board should you not need all that fancy stuff. For some people a simple BASIC editor is enough. I cannot praise the company highly enough for their friendly manner. After I purchased the Bank Select Board they called me on more than one occasion to be sure that (1) I had received it, (2) I had no trouble installing it, and (3) that it functioned properly and to my satisfaction. We had a minor problem at first with the Devlin RAM board due to a minor error in the FREPOST instructions. This has since been corrected.

Neither was this my last contact with them. Bill Freiberger has personally called me at least twice since to keep me informed of new products and developments in which he felt I might be interested. Now THAT is service. Do yourself a favor. Write to FREPOST Computers for a catalog of all their stuff. They carry things like add-on lower case, add-on 16K RAM (for those who still have 16K machines), and lots more. As I said before, their prices are very reasonable. I'm looking forward to their next coup.

I suggested earlier that other sources of software did exist. There are distributors in Canada and Australia as well. In the U.S., Jim Helms, 1121 Warbler, Kerrville, TX, 78028, writes and sells software. I have several of his programs, and all of them are excellent. All of his programs should be available from FREPOST (in EPROM if desired) or from Intelligent Computer Systems.

I wish to comment on one more source. This is COLORWARE in Canada (not to be confused with Quality Software Associates). Some time ago I ordered three programs from them. One was a Pac-man program called GOBBLER. I found the company to be slow (7 weeks instead of the stated 4-6 weeks) in delivery time for one thing. Two of the three programs gave disk errors, which COLORWARE claimed was a fault of my disk drives. In all fairness I should state that I was able to load the programs on another machine even though my own disk drives rarely act up on other outside programs. But the most annoying thing was that GOBBLER would simply not run on a V8.79 system even though both versions were on the disk. This was a ridiculous situation for it was obvious that COLORWARE had not tested the V8.79 version before marketing. Indeed, the problem was faulty conversion of V6.78 to V8.79 addresses. Even more frustrating was the fact that after I received the supposed 'corrected' version from them it still would not run. I went in myself with the MLDP program, patiently disassembled all system ROM calls, and compared each with what it should have been for V8.79. Incredible! Four of the addresses had not even been changed! Again, COLORWARE failed to test the program even after the customer complained! After I made these corrections to my own copy, all was fine. I sent the corrections to the company with a to-the-point letter. They did not do me the courtesy of responding. The GOBBLER program itself was a disappointment. For a purported Pac-man, it is way below standards. Its only mark of distinction is its excellent sound routines. I you crave Pac-man, buy the CHOMP program (from Intelligent Computer Systems). In my opinion it is the best arcade game currently available for the CCII.

In all fairness I should say that the other two programs worked well and that one of them, a utility called AGILIS is a real boon. AGILIS allows the user to create a screen display, after which the program will itself write a BASIC program to re-create that display. It's a marvelous program. It is a shame that the company selling it has chosen to run their business in so poor a way. This is the only supplier of CCII software which I have encountered that has not met my standards. I feel that other CCII users should know this. I trust that this will be an isolated case. The CCII is too good a machine to settle for anything less than the best in support.

The CCII may be 'dead', but its supporters are alive and active. I suspect there are more surprises yet in store for us.

Controlling Keyboard Input in BASIC

by Dan Murray
7064 35th NE
Seattle, WA 98115

The Dec/Jan 1982 issue of **Colorcue** has an article by Bernie Raffee that presents a method for controlling user input by using a machine language subprogram. The assembler language program makes interesting reading, and I learned some techniques for interfacing machine code with BASIC. However, it seems like a lot of work for a simple objective - preventing the user from making mistakes. To avoid mistakes, you must control the user's actions, i.e., **when** he types, and **what** he types. Bernie Raffee has certainly done this, and very effectively, but if the same result can be achieved from BASIC it would be easier and more convenient. Buried amongst some past issues of **Colorcue** are the two tools need to control the user from BASIC.

When

A prior issue of **Colorcue** says to use the command "OUT 8,241" to lockout the keyboard (except for the CPU reset key). whenever you want the keyboard reactivated, just use "OUT 8,255", and the keyboard will work as usual until the next "OUT 8,241" command. This function is important, because ISC's BASIC immediately echoes any keyboard input at the current cursor location. This can be disastrous if you are building a screen format and the user starts to hit keys at random. So, by using this function, you have control over **when** the computer will receive input from the keyboard.

What

Another past issue of **Colorcue** gives a short BASIC routine that POKEs a tiny machine code routine into high memory and sets up linkage for CALLing the routine from BASIC. The beauty of this machine code is that it performs a very simple, primitive function - interrogate the keyboard and return the ASCII value of any key pressed, returning a -1 if no key was pressed. Also, nothing is echoed to the screen. As an extra frill, you can tell the routine how many seconds to wait for input, up to 255. By keeping the routine simple, you can integrate it into many different applications, treating it much like any other BASIC function. When using this function in a larger BASIC subroutine, you can perform any number of input and editing tasks, controlling input entirely from BASIC.

Listing 1 is a sample program to demonstrate the use of these functions, including numeric verification with signed fractional values. Since learning these techniques, I have developed the habit of keeping the BASIC "POKE" routine on disk and LOADing it into memory before starting a new program so that the routine forms the beginning of every program I write. If it later turns out that a particular program won't need this routine, it's a simple matter to delete those few lines of code. NOTE: it is important to remember t "CLEAR xxx" after executing the BASIC "POKE" routine so that the BASIC system will reset the string space pointer, otherwise BASIC will clobber your machine code with any strings used later in the program.

Before I leave you with the impression that this method is perfect, let me assure you that it's not. The most obvious problem is BASIC's speed, or lack of it. The second problem is related to the keyboard. It seems that ISC designed their keyboards with what is called "three key rollover". This means that you can hit a second and even a third key before releasing the first one. As long as each key is released before the next

one is pressed (even at high speed), the program will work just fine.

I hope these techniques will encourage programmers to write thorough programs that are user-friendly but still easy to design, code, and understand. Also, if someone can find any "holes " in my error traps, please let me know.

```
  1   REM  ***********************************
  2   REM  *   SAMPLE DATA ENTRY ROUTINES    *
  3   REM  *         BY DAN MURRAY           *
  4   REM  *            11/04/82             *
  9   REM  ***********************************
 20   GOSUB 63000        :REM  LOAD INPUT ROUTINE INTO HIGH MEMORY
 50   CLEAR 1000

100   REM  * CONTROL ROUTINE *
120   GOSUB 1000         :REM  - INITIALIZE
140   GOSUB 2000         :REM  - SET UP SCREEN

160   GOSUB 3000         :REM  - INPUT SOME DATA
180   REM  * * * * * * * * * * * * * * * *
200   REM  --- FURTHER PROCESSING GOES HERE ---
210   REM  ... FILE UPDATING/DISPLAY, ETC.  ...
230   REM  * * * * * * * * * * * * * * * *
480   GOTO 160

500   REM  * FIELD INPUT ROUTINE *
510   A$= ""
      :A= 0
      :PLOT 3,0,0
      :PRINT CHR$ (13)
      :IF NN= 0THEN NN= 1
520   PLOT 3,XX,YY
      :PRINT LEFT$ (FILL$,NN);
      :PLOT 3,XX,YY
530   OUT 8,255
      :AA= CALL (0)
      :OUT 8,241
      :IF AA> 31AND AA< 97AND A< NNTHEN A= A+ 1
      :PRINT CHR$ (AA);
      :A$= A$+ CHR$ (AA)
      :GOTO 530
540   IF AA= 26AND A> 0THEN A= A- 1
      :PRINT BU$;
      :A$= MID$ (A$,1,A)
      :GOTO 530
550   IF AA= 13THEN PRINT SPC( NN- A+ 1);
      :RETURN
560   IF AA= 11THEN 500
570   GOTO 530

1000  REM  * INITIALIZE *
1020  PLOT 15,6,2,12
      :REM  SET CCI AND CLEAR SCREEN
1030  PRINT SPC( 15);"DATA ENTRY DEMONSTRATION"
1050  OUT 8,255          :REM  GET READY TO SHUT OFF KEYBOARD
1060  OUT 8,241          :REM  KEYBOARD IS NOW OFF
1070  POKE 33289,255         :REM  MAXIMUM CHARS/LINE

1100  REM  - DEFINE GLOBAL VARIABLES -
1120  REM  'FILL' CHARACTERS SHOW THE FIELD'S SIZE
1130  FILL$= "----------------------------------------------------------------
      ------"
1150  REM   BACKUP VARIABLE USED TO ERASE 1 CHARACTER
1160  BU$= CHR$ (26)+ LEFT$ (FILL$,1)+ CHR$ (26)
1980  RETURN

2000  REM  * SET UP SCREEN DISPLAY *
2020  PLOT 3,0,5
      :PRINT "NAME:"
2030  PLOT 3,15,8
      :PRINT "- ADDRESS -"
2040  PLOT 3,0,10
      :PRINT "STREET:"
2050  PLOT 3,0,12
      :PRINT "STATE:"
```

```
2060   PLOT 3,0,14
       :PRINT "ZIP CODE:"
2070   PLOT 3,0,17
       :PRINT "TELEPHONE - WORK:"
2080   PLOT 3,32,17
       :PRINT "HOME:"
2090   PLOT 3,0,19
       :PRINT "YEARLY INCOME: $"
2980   RETURN

3000   REM  * DATA INPUT *
3020   REM  - INPUT NAME -
3030   XX= 6
       :YY= 5
       :NN= 30
       :GOSUB 500
3040   IF A$= ""THEN 3030
3060   REM  - INPUT STREET -
3070   XX= 8
       :YY= 10
       :NN= 50
       :GOSUB 500
3080   IF A$= ""THEN 3070
3100   REM  - INPUT STATE -
3110   XX= 7
       :YY= 12
       :NN= 10
       :GOSUB 500
3120   IF A$= ""THEN 3110
3150   REM  - INPUT ZIP CODE -
3160   XX= 10
       :YY= 14
       :NN= 5
       :GOSUB 500
       :GOSUB 9500
3170   IF LEN (A$)< 5THEN 3160
3200   REM  - INPUT WORK PHONE -
3210   XX= 18
       :YY= 17
       :NN= 8
       :GOSUB 500
       :IF A$ = "" THEN 3210
3250   REM  - INPUT HOME PHONE -
3260   XX= 38
       :YY= 17
       :NN= 8
       :GOSUB 500
3270   IF A$= ""THEN 3260
3300   REM  - INPUT INCOME -
3310   XX= 16
       :YY= 19
       :NN= 9
       :GOSUB 500
       :GOSUB 9000
3320   IF A$= ""THEN 3310
3980   RETURN

9000   REM  * NUMERIC VERIFICATION *
9020   N= 0
       :IF A$= ""THEN RETURN
9030   FOR X= 1TO LEN (A$)
       :Y= ASC (MID$ (A$,X,1))
       :IF Y> 47AND Y< 58THEN NEXT X
       :RETURN
9040   IF (Y= 43OR Y= 45)AND X= 1THEN NEXT X
9050   IF Y= 46AND N= 0THEN N= 1
       :NEXT X
9060   A$= ""
       :RETURN

9500   REM  * INTEGER VERIFICATION *
9520   N= 0
       :IF A$= ""THEN RETURN
9530   FOR X= 1TO LEN (A$)
       :Y= ASC (MID$ (A$,X,1))
       :IF Y> 47AND Y< 58THEN NEXT X
       :RETURN
9540   A$= ""
       :RETURN
```

```
63000  REM  *** INKEY ROUTINE ***
63010  DATA 245,229,197,1,206,40,205,36,0,202,-1,-1,11
63020  DATA 121,176,194,-1,-1,29,194,-1,-1,17,255,255,195
63030  DATA -1,-1,95,175,87,175,50,255,129,193,225,241,201
63040  TM= 256* PEEK (32941)+ PEEK (32940)
63050  IF TM> 65503THEN 63110
63060  RESTORE 63010
63070  FOR I= 1TO 39
       :READ A
63080  IF A> = 0AND A< > PEEK (TM+ I)THEN I= 39
       :A= 999
63090  NEXT I
63100  IF A< 256THEN 63200
63110  TM= TM- 39
       :RESTORE 63010
63120  FOR I= 1TO 39
       :READ A
       :POKE TM+ I,A- (A< 0)
63130  NEXT I
63140  Z= TM+ 29
       :AD= TM+ 11
       :GOSUB 63180
63150  Z= TM+ 7
       :AD= TM+ 17
       :GOSUB 63180
63160  Z= TM+ 4
       :AD= TM+ 21
       :GOSUB 63180
63170  Z= TM+ 32
       :AD= TM+ 27
       :GOSUB 63180
       :GOTO 63190
63180  ZZ= INT (Z/ 256)
       :POKE AD,Z- 256* ZZ
       :POKE AD+ 1,ZZ
       :RETURN
63190  Z= TM
       :AD= 32940
       :GOSUB 63180
63200  Z= TM+ 1
       :AD= 33283
       :GOSUB 63180
63210  POKE 33282,195
63220  RETURN
```

A FORTRAN Plot Library

by Joseph J. Charles
P.O. Box 750
Hilton, NY 14468

Here is a library of FORTRAN plot subroutines and a time-delay subroutine which you can add to your FORTRAN programs. Listing 7 is the FORDEM program which simply demonstrates the use of the plot subroutines.

The plot subroutines should be compiled into a relocatable file named PLTLIB.REL. To use them, all you need to do is CALL them as desired in a FORTRAN mainline or other subroutine. Then, after compiling the FORTRAN source files, link the files using L80. However, before linking in the FORLIB.REL library, link in the PLTLIB.REL library by specifying the /S switch:

```
L80>PLTLIB/S
```

This will cause L80 to search PLTLIB.REL for the plot, timer and scaler calls. When the next L80> prompt appears, respond with FORLIB/S as usual.

The subroutine calls and arguments are described below.

SUBROUTINE SCALE(ARRAY1,ARRAY2,N,MIN,MAX)

ARRAY1 is the input array to be scaled and converted to logical. ARRAY2 is the converted, scaled array. It is scaled to screen coordinates. N is the dimension of ARRAY1. I think it's fine to let N=d1+d2+...dn if ARRAY1 is multidimensional. MIN is the minimum value for the scaled array. You supply this in the CALL. MAX is the maximum value for the scaled array. MIN and MAX should be in screen coordinates and integer variables or integer constants.

SUBROUTINE LINE(X,Y,N,FCOLOR,
BCOLOR,IBL,IFL)

X, Y are the arrays of X and Y coordinate pairs of dimension N. They must be logical arrays and scaled to screen coordinates. This is accomplished by two CALLS to SCALE prior to the CALL to LINE. LINE plots a line or curve of connected points. It is essentially a routine for doing vector graphics. FCOLOR and BCOLOR are the foreground and background colors desired. They must be logical variables whose value corresponds to 16-23 for the various colors. IBL is a blink determiner. If IBL=1 the plotted line will blink; no blink otherwise. IFL is a flag determiner. If IFL=1 the flag is turned on and the plot is exclusive ORed with each pixel just as for PLOT 30 in BASIC.

SUBROUTINE PPLOT(....)

A point plot subroutine essentially identical to LINE.

SUBROUTINE XBAR(X0,Y,XMAX,FCOLOR,BCOLOR,
IBL,IFL)

Each CALL plots one x-bar graph with X, Y and XMAX as in BASIC. The other arguments are as for LINE. As before, X, Y and XMAX must be scaled and logical.

SUBROUTINE YBAR(Y0,X,YMAX,...)

Same as for XBAR.

SUBROUTINE TIMER(SECS)

Provides a time delay of "SECS" seconds. SECS must be an integer variable or integer constant.

```
C       ========================================================
C
        SUBROUTINE SCALE(ARRAY1,ARRAY2,N,MIN,MAX)
C
C       THIS SUBROUTINE WILL SCALE ARRAY1 ,A REAL ARRAY, TO
C         THE LOGICAL ARRAY, ARRAY2. ARRAY2 MAY BE SENT TO THE
C         SCREEN VIA THE PLOT ROUTINES, LOGICAL AND PROPERLY SCALED.
C
C       WRITTEN BY: JOSEPH J. CHARLES, 130 SHERWOOD DRIVE,
C                   HILTON, NY 14468  TEL:(716) 392-8152
C
C
C       VERSION: JULY 18,1982,  6:17 PM
C
        DIMENSION ARRAY1(1)
        LOGICAL ARRAY2(1)
C
        ALOW=1.E30
        AHIGH=-1.E30
C
        DO 1 I=1,N
        IF(ARRAY1(I) .LT. ALOW) ALOW=ARRAY1(I)
1       IF(ARRAY1(I) .GT. AHIGH) AHIGH=ARRAY1(I)
C
        DELA=AHIGH-ALOW
        DELTA=MAX-MIN
        SCALER=DELTA/DELA
C
        DO 2 I=1,N
2       ARRAY2(I)=(ARRAY1(I)-ALOW)*SCALER+MIN
C
        RETURN
        END
C
```

Listing 1

```
C       ========================================================
C
        SUBROUTINE LINE (X,Y,N,FCOLOR,BCOLOR,IBL,IFL)
C
C       THIS SUBROUTINE WILL PLOT CURVES WITH OPTIONS
C         FOR COLOR,BLINKING, AND "EXCLUSIVE OR"-ING.
C
C       WRITTEN BY: JOSEPH J. CHARLES, 130 SHERWOOD DRIVE,
C                   HILTON, NY 14468  TEL:(716) 392-8152
C
C       VERSION: JULY 18,1982,  6:17 PM
C
        DIMENSION X(1),Y(1)
C
        LOGICAL X,Y,FCOLOR,BCOLOR,BLINK,BLA7OF,GPM,VECTOR
        LOGICAL FGOFL0,BGOFLO,PLEND,GREEN,BLACK
C
        DATA GPM,VECTOR,PLEND,FGOFL0,BGOFLO/2,242,255,29,30/
        DATA BLA7OF,BLINK,GREEN,BLACK/15,31,18,16/
C
        WRITE(3) BGOFLO,BCOLOR,FGOFL0,FCOLOR
C
        WRITE(3) BLA7OF
        IF(IBL .EQ. 1) WRITE(3) BLINK
C
        IF(IFL .EQ.1) WRITE(3) BGOFLO
C
        WRITE(3) GPM,X(1),Y(1),VECTOR
C
        DO 1 I=2,N
1       WRITE(3) X(I),Y(I)
C
        WRITE(3)PLEND,BGOFLO,BLACK,FGOFL0,GREEN,BLA7OF
C
        RETURN
        END
C
```

Listing 2

```
C       ========================================================
C
        SUBROUTINE PPLOT (X,Y,N,FCOLOR,BCOLOR,IBL,IFL)
C
C       THIS SUBROUTINE WILL PRODUCE POINT PLOTS WITH OPTIONS
C         FOR COLOR,BLINKING, AND "EXCLUSIVE OR"-ING.
C
C       WRITTEN BY: JOSEPH J. CHARLES, 130 SHERWOOD DRIVE,
C                   HILTON, NY 14468  TEL:(716) 392-8152
C
C
C       VERSION: JULY 18,1982,  6:17 PM
C
        DIMENSION X(1),Y(1)
C
        LOGICAL X,Y,FGOFLO,BGOFLO,BLINK,GPM,POINT,BLAZOF
        LOGICAL FCOLOR,BCOLOR,PLEND,GREEN,BLACK
C
        DATA GPM,POINT,PLEND,FGOFLO,BGOFLO/2,253,255,29,30/
        DATA BLAZOF,BLINK,GREEN,BLACK/15,31,18,16/
C
        WRITE(3) BGOFLO,BCOLOR,FGOFLO,FCOLOR
C
        WRITE(3) BLAZOF
        IF(IBL .EQ. 1) WRITE(3) BLINK
C
        IF(IFL .EQ. 1) WRITE(3) BGOFLO
C
        WRITE(3) GPM,X(1),Y(1),POINT
C
        DO 1 I=2,N
1       WRITE(3) X(I),Y(I)
C
        WRITE(3) PLEND,BGOFLO,BLACK,FGOFLO,GREEN,BLAZOF
        RETURN
        END
C
```

Listing 3

```
C       ========================================================
C
        SUBROUTINE XBAR(X0,Y,XMAX,FCOLOR,BCOLOR,IBL,IFL)
C
C       THIS SUBROUTINE PRODUCES X-BAR-GRAPHS WITH OPTIONS
C         FOR COLOR,BLINKING,AND "EXCLUSIVE OR"-ING.
C
C       WRITTEN BY: JOSEPH J. CHARLES, 130 SHERWOOD DRIVE,
C                   HILTON, NY 14468  TEL:(716) 392-8152
C
C
C       VERSION: JULY 18,1982,  6:17 PM
C
        LOGICAL GPM,XBARG,FCOLOR,BCOLOR,BLINK,X0,Y,XMAX,PLEND
        LOGICAL GREEN,BLACK,FGOFLO,BGOFLO,BLAZOF
C
        DATA GPM,XBARG,PLEND,BLINK/2,250,255,31/
        DATA FGOFLO,GREEN,BGOFLO,BLACK,BLAZOF/29,18,30,16,15/
C
        WRITE(3) BGOFLO,BCOLOR,FGOFLO,FCOLOR
C
        WRITE(3) BLAZOF
        IF (IBL .EQ. 1) WRITE(3) BLINK
C
        IF (IFL .EQ. 1) WRITE(3) BGOFLO
C
        WRITE(3) GPM,XBARG,X0,Y,XMAX,PLEND
C
        WRITE(3) BGOFLO,BLACK,FGOFLO,GREEN,BLAZOF
C
        RETURN
        END
C
```

Listing 4

```
C       ======================================================
C
C
        SUBROUTINE YBAR(Y0,X,YMAX,FCOLOR,BCOLOR,IBL,IFL)
C
C       THIS SUBROUTINE PRODUCES Y-BAR-GRAPHS WITH OPTIONS
C         FOR COLOR,BLINKING,AND "EXCLUSIVE OR"-ING.
C
C       WRITTEN BY: JOSEPH J. CHARLES, 130 SHERWOOD DRIVE,
C                   HILTON, NY 14468  TEL:(716) 392-8152
C
C
C       VERSION: JULY 18,1982,  6:17 PM
C
        LOGICAL GPM,YBARG,FCOLOR,BCOLOR,BLINK,Y0,X,YMAX,PLEND
        LOGICAL GREEN,BLACK,FGOFLO,BGOFLO,BLA7OF
C
        DATA GPM,YBARG,PLEND,BLINK/2,246,255,31/
        DATA FGOFLO,GREEN,BGOFLO,BLACK,BLA7OF/29,18,30,16,15/
C
        WRITE(3) BGOFLO,BCOLOR;FGOFLO,FCOLOR
C
        WRITE(3) BLA7OF
        IF (IBL .EQ. 1) WRITE(3) BLINK
C
        IF (IFL .EQ. 1) WRITE(3) BGOFLO
C
        WRITE(3) GPM,YBARG,Y0,X,YMAX,PLEND
C
        WRITE(3) BGOFLO,BLACK,FGOFLO,GREEN,BLA7OF
C
        RETURN
        END
```

Listing 5

```
C       ======================================================
C
        SUBROUTINE TIMER(SECS)
C
C       THIS SUBROUTINE PROVIDES A DELAY OF "SECS" SECONDS
C
C       WRITTEN BY: JOSEPH J. CHARLES, 130 SHERWOOD DRIVE,
C                   HILTON, NY 14468  TEL:(716) 392-8152
C
C
C       VERSION: JULY 18,1982,  6:17 PM
C
        INTEGER DELTA,SECS
C
        LOGICAL HR,MIN,SEC
C
        HR=PEEK(X'81BB')
        MIN=PEEK(X'81BA')
        SEC=PEEK(X'81B9')
C
C       STARTING TIME
        STIME=3600.*HR+60.*MIN+SEC
C
1       HR=PEEK(X'81BB')
        MIN=PEEK(X'81BA')
        SEC=PEEK(X'81B9')
C
C       PRESENT TIME
        PRESTM=3600.*HR+60.*MIN+SEC
C
        DELTA=PRESTM-STIME
        IF(DELTA .LT. SECS) GOTO 1
C
        RETURN
        END
C
```

Listing 6

Listing 7

```
      PROGRAM FORDEM
      DIMENSION X(51),Y(51),X2(51),Y2(51),Y3(51)
      LOGICAL BLACK,RED,GREEN,YELLOW,BLUE,MAGNTA,CYAN,WHITE
      LOGICAL FGOFLO,BGOFLO,BLA7OF,BLINK,ERASE,SETBR,ESC
      LOGICAL A7ON,BRC,PAGE,SCROLL,CURSOR,LX,LY,COLOR,HOME
      LOGICAL X2,Y2,Y3,HR,MIN,SEC
      DATA BLACK,RED,GREEN,YELLOW,BLUE,MAGNTA/16,17,18,19,20,21/
      DATA CYAN,WHITE,ERASE,BLA7OF,BLINK,HOME/22,23,12,15,31,8/
      DATA ESC,SETBR,A7ON,PAGE,SCROLL,CURSOR/27,18,14,24,11,3/
      DATA BGOFLO,FGOFLO/30,29/
      WRITE(3)BGOFLO,BLACK,FGOFLO,YELLOW,BLA7OF
      WRITE(3,105)
105   FORMAT(' INPUT DATA MUST BE IN FORM INDICATED. FOR EXAMPLE'/
     1 ' ## ## ##  MEANS 3 TWO DIGIT NUMBERS ,RIGHT JUSTIFIED, WITH A',
     2 '  SPACE BETWEEN THEM. E.G. 09 17 30' )
      WRITE(3,106)
106   FORMAT(' ENTER THE TIME (HOURS MINUTES SECONDS) ## ## ##' )
      READ(3,107) IHR,IMIN,ISEC
107   FORMAT(3(I2,1X))
C     CONVERT TO  ONE BYTE EACH
      HR=IHR
      MIN=IMIN
      SEC=ISEC
C     POKE TO 33211,33210,33209
C           (X'81BB',X'81BA',X'81B9' IN HEX)
      CALL POKE(X'81BB',HR)
      CALL POKE(X'81BA',MIN)
      CALL POKE(X'81B9',SEC)
      WRITE(3,111)
111   FORMAT(' ENTER BAUD RATE CODE FOR YOUR PRINTER IF'/,
     1 ' YOU WANT TO USE IT. (1-7) ENTER 0 OTHERWISE. # ')
      READ(3,112) IBRC
112   FORMAT(I1)
      IF( IBRC .EQ. 0) GO TO 2
      BRC=IBRC
3     WRITE(3) ESC,SETBR,BRC
      WRITE(3,113)
113   FORMAT(' ENTER NUMBER OF STOP BITS FOR YOUR PRINTER.'/,
     1 ' (1 OR 2) # ')
      READ(3,112) NSB
      IF(NSB .EQ. 1) WRITE(3) A7ON
      IF(NSB .EQ. 2) WRITE(3) BLA7OF
2     WRITE(3,130)
130   FORMAT(' FOR HOW MANY SECONDS WOULD YOU LIKE TO OBSERVE '/
     1 ' EACH DEMONSTRATION PLOT? ## ')
      READ (3,107) NSEC
      WRITE(3,101)
101   FORMAT(' ENTER X,Y FOR LOWER LEFT CORNER: ### ###'/,
     1 ' ENTER X GREATER THAN 127 TO END PROGRAM.')
      READ (3,102)IXMIN,IYMIN
102   FORMAT(I3,1X,I3)
      IF(IXMIN .GT. 127) GOTO 8
      WRITE(3,103)
103   FORMAT(' ENTER X,Y FOR UPPER RIGHT CORNER: ### ###')
      READ (3,102) IXMAX,IYMAX
C     DEFINE GAUSSIAN FUNCTION:
C           N(0,1) ( NOT NORMALIZED TO UNIT AREA )
      Y(26)=1.
      X(26)=0.
      DO 1 I=1,25
      X(I)=(I-26)/8.
      Y(I)=EXP(-0.5*(X(I)**2))
      X(I+26)=I/8.
1     Y(I+26)=EXP(-0.5*((I+26)**2))
      WRITE DATA TO PRINTER IF PRINTER IS THERE
      IF(IBRC .EQ. 0) GO TO 4
6     WRITE(3,104)
104   FORMAT(' ',   I',5X,'X(I)',8X,'Y(I)',9X,'X(I+1)    Y(I+1)')
```

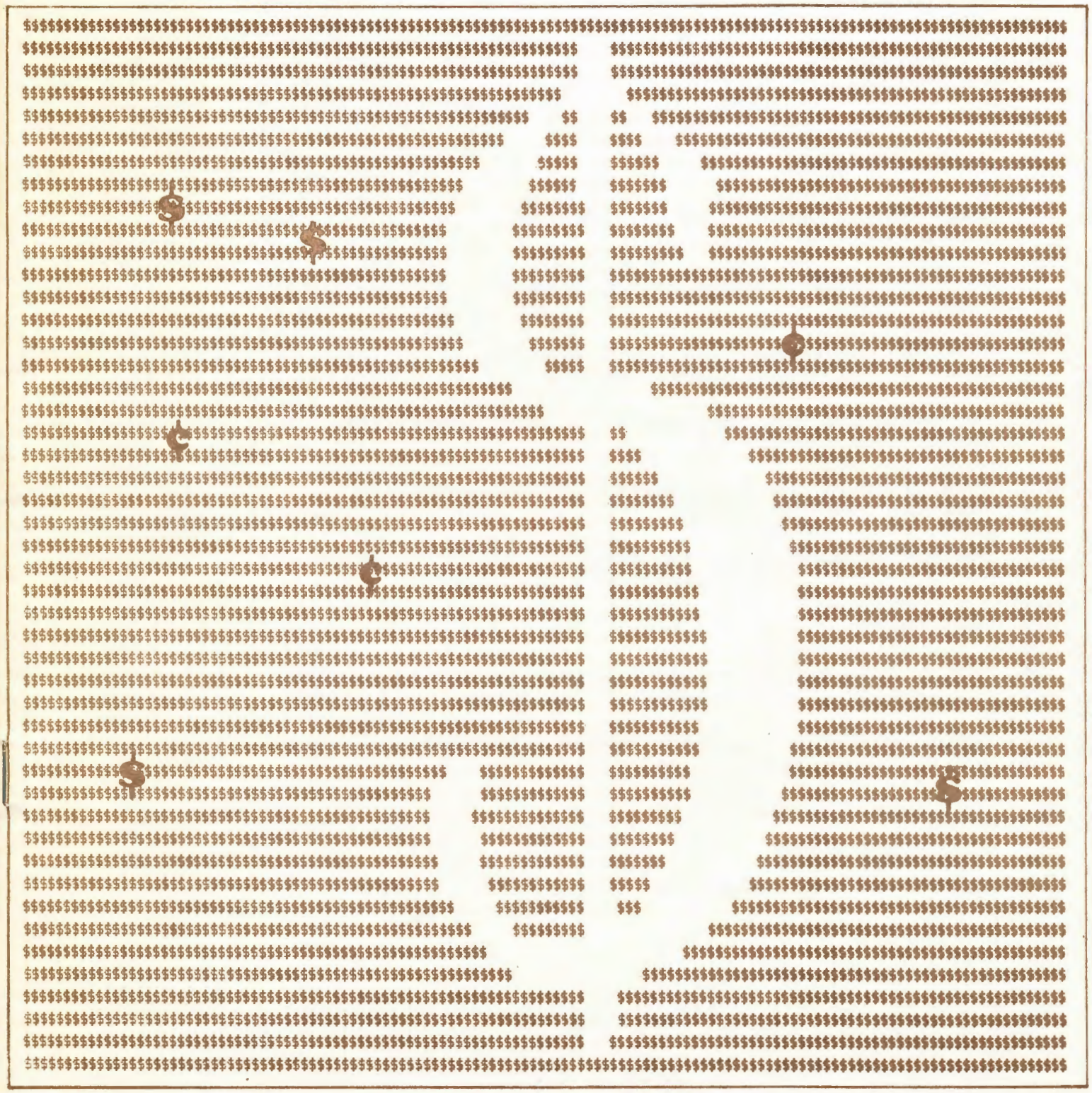
```
        WRITE(2,100) (I,X(I),Y(I),X(I+1),Y(I+1), I=1,49,2)
100     FORMAT(' ',I5,2(F10.4,F10.6,5X))
C
        WRITE(2,110)
C
4       CALL SCALE(X,X2,51,IXMIN,IXMAX)
C
        CALL SCALE(Y,Y2,51,IYMIN,IYMAX)
C
C       WRITE SCALED VALUES TO PRINTER
        IF( IBRC .EQ.0 ) GOTO 5
C
7       WRITE (2,108)
108     FORMAT('////' SCALED BYTE VALUES'/)
C
        WRITE(2,109) (I,X2(I),Y2(I),X2(I+1),Y2(I+1), I=1,49,2)
109     FORMAT (' ',I5,5X,I3,7X,I3,12X,I3,7X,I3)
C
        WRITE(2,110)
110     FORMAT(///)
C
5       WRITE(3) ESC,PAGE,ERASE
C
        WRITE(3,140)
140     FORMAT(' PLAIN OLD BELL-SHAPED CURVE FOR STARTERS...')
        CALL TIMER(2)
        CALL LINE (X2,Y2,51,RED,BLACK,0,0)
        CALL TIMER(NSEC)
        WRITE(3) ERASE
C
        WRITE(3,141)
141     FORMAT(' SAME, BUT IN POINT PLOT MODE WITH BLINKING.')
        CALL TIMER(3)
        WRITE(3) ERASE
        CALL PPLOT(X2,Y2,51,GREEN,BLACK,1,0)
        CALL TIMER(NSEC)
C
        WRITE(3,142)
142     FORMAT(' PUT TWO CURVES UP...')
        CALL TIMER(2)
        WRITE(3) ERASE
        CALL LINE (X2,Y2,51,YELLOW,BLACK,0,0)
        DO 14 I=1,41
14      Y3(I)=Y2(I+10)
        CALL LINE (X2,Y3,41,BLUE,BLACK,0,0)
        CALL TIMER(NSEC)
C
        WRITE(3) HOME
        WRITE(3,143)
143     FORMAT(' ERASE ONE BY REPLOTTING WITH XOR-ING')
        CALL TIMER(2)
        CALL LINE (X2,Y2,51,YELLOW,BLACK,0,1)
        CALL TIMER(NSEC)
        WRITE(3) ERASE
C
        WRITE(3,144)
144     FORMAT(' Y BAR-GRAPH MODE...ONE SUB CALL FOR EACH LINE')
        CALL TIMER(4)
        DO 13 I=1,51
13      CALL YBAR(Y2(1),X2(I),Y2(I),RED,BLACK,0,0)
        CALL TIMER(NSEC)
        WRITE(3) ERASE
C
        WRITE(3,145)
145     FORMAT(' X BARS...')
        CALL TIMER(2)
        WRITE(3) ERASE
        DO 9 I=1,51
9       CALL XBAR(Y2(1),X2(I),Y2(I),YELLOW,BLACK,0,0)
        CALL TIMER(NSEC)
C
        WRITE(3) HOME
        WRITE(3,146)
146     FORMAT(' ERASE BY XOR-ING EVERY OTHER ONE...')
        CALL TIMER(4)
        DO 10 I=2,50,2
10      CALL XBAR(Y2(1),X2(I),Y2(I),YELLOW,BLACK,0,1)
```

```
            CALL TIMER(NSEC)
C
            WRITE(3) HOME
            WRITE(3,147)
147         FORMAT(' REPLOT REMAINING ONES BLINKING YELLOW.')
            CALL TIMER(4)
            DO 11 I=1,51,2
11          CALL XBAR(Y2(1),X2(I),Y2(I),YELLOW,BLACK,1,0)
            CALL TIMER(NSEC)
            WRITE(3) ERASE
C
            DO 12 K=1,5
            DO 12 I=17,23
            COLOR=I
12          WRITE(3)BGOFLO,COLOR,ERASE
C
            LX=0
            LY=16
            WRITE(3)BGOFLO,BLACK,FGOFLO,RED,ERASE,BLINK,A7ON,CURSOR,LX,LY
            WRITE(3,131)
131         FORMAT(17X,'TH..TH..THAT''S ALL, FOLKS!!! ')
C
            CALL TIMER(15)
            WRITE(3) ESC,SCROLL,FGOFLO,GREEN,BLA7OF
C
            GOTO 2
C
C           END
```

BULK RATE
U.S. POSTAGE
PAID
Rochester, N. Y.
Permit No. 415

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

Colorcue



Feb/Mar 1983

$2.

Colorcue

A bi-monthly publication by and for Intecolor and Compucolor Users

February/March, 1983
Volume 5, Number 4

Editors:
Ben Barlow
David B. Suits
Compuserve: 70045,1062

COLORCUE is published bi-monthly. Subscriptions are US$12/year in the U.S., Canada and Mexico, and US$24 (includes air mail postage) elsewhere. Some back issues are available. All editorial and subscription correspondence should be addressed to **COLORCUE**, 161 Brookside Dr., Rochester, NY 14618, USA. All articles in **COLORCUE** are checked for accuracy to the best of our ability, but they are NOT guaranteed to be error free.

Editors' Notes

R.I.P.

The latest edition of **Forum International** recently arrived. Sadly, its date indicates that it is the "Final Edition". Lack of funds has forced its discontinuation.

This marks an unfortunate turn of events for Compucolor/Intecolor owners. **FORUM** was a publication which helped keep us going with worthwhile, informative and timely articles. (This latest issue is no exception.) The rights to its name and content pass to CUVIC. If we're lucky, CUVIC will manage to resurrect **Forum** in some shape. We all owe Doug Peel, **Forum**'s indefatigable editor, a large "Thank you" and a "Well done" for his creation of **Forum** in the first place, and for his time and energy (and, we suspect, money) in keeping the publication alive for so long.

What About COLORCUE?

Which brings up a related topic. Will **Colorcue** be able to keep going? The answer at this point is a qualified "Maybe." Frankly, we're not in the best of shape, although we're by no means done for. We've had letters recently suggesting that, if necessary, we ought to increase the subscription price rather than discontinue publication. We haven't liked the idea of increasing subscription prices, of course. But we are even less happy about the prospect of quitting.

Currently, all US subscriptions are mailed 3rd class. Our experience with this manner of trying to get **Colorcue** out is that 3rd class US mail is unreliable. (You can say that again.) Moreover, it is slow. Moreover, it is extra work for us. (The Post Office charges us less because we do some of the work for them.) We would like very much to send out all **Colorcues** 1st class, even though this will be a bit more expensive. The savings in terms of time and energy would be a boon to us. The increased subscription rates would not be a boon to you. Well, subscription rates have to go up a bit anyway, since we've been carrying increased printing costs (and, incidentally, increased 3rd class rates) for over a year. So here is our proposal. Starting with the August/September issue (which, by the way, will mark the start of our third year of editorship! You're invited to the birthday party), subscription rates will be US$18 in North America, and US$30 elsewhere. Each issue will be sent 1st class (air mail where appropriate), and we might even be able to get back on schedule. (Promises, promises....) If this places a horrific burden on you and/or your wallet, please let us know. We hope, though, that you'll be able to stay with us and to continue to support **Colorcue** with your dollars and your articles. You've kept Compucolor's publication going so far; especially now in the absence of **Forum**, we need **Colorcue** more than ever.

Many Compucolor owners have been hit with the "blown transistor on the analog board" problem when they don't hit the CPU reset key quickly enough after power on (or sometimes when they do). The analog board relies on an oscillator on the digital board to provide a pulse train for the switching power supply. If that oscillator doesn't begin oscillating when the power comes on, the power supply puts out full power and after a few seconds, poof! $. Infrequently, even CPU reset doesn't kick off the oscillator, and still poof! Tom Devlin, Compucolor maven of the midwest (would you believe he's got two?) has devised a new board with a phase locked loop chip on it that plugs easily onto the digital board and always generates the pulse train. Installation is super simple, and worries of blown out power transistors (whether caused by experience or hearsay) are banished. $35 US money, from Tom Devlin, 3809 Airport Road, Waterford, MI 48095.

We have recently received a copy of the manual for Bill Greene's machine language debugger (or, as he calls it, the IDA--Interpreter, Disassembler, Assembler). We have not seen the actual program in operation, but from its description, it's a powerful tool. In addition to the usual debug features, his IDA allows you to set the baud rate,

print to the printer, compare memory contents, search memory, search and replace, and execute FCS commands. Bill also has what appears to be a fairly powerful FORTH interpreter. Contact Bill Greene at 3601 Noble Creek Drive, N.W., Atlanta, GA 30327.

Speaking of FORTH, the Rochester users group (CHIP) has a FORTH interpreter in their library (costs you $10 to join the group), implemented by Jim Minor. Jim has also recently added to the CHIP library (are you ready for this?) a PASCAL compiler written in FORTH! Can't be bad. If you really must work with Pascal (anyone who knows me knows how much I—DBS—dislike Pascal), then you might as well look into this unexpected way of implementing it.

M. F. Pezok asks, "Are there any Technical Wizards out there that can/WILL design a cheap serial line buffer with handshaking? 8K of buffer would be great; 16K of buffer would be OUTSTANDING!" A buffering device is a good and useful idea. As if having read your mind, M.F., it just so happens that Lou Milich and Dave Suits have been working on just such a project. Why, though, stop with 16K? We're building a Z80A controller with 48K of RAM. The prototype is almost built, and we hope to be able to publish the design sometime soon. The cost, if you build it yourself, won't be exactly cheap, but it will beat the price of comparable units on the market.

ISC has introduced its 8001R/M and 9001R/M terminals configured for use with Sperry Univac's MAPPER system in order to emulate and be plug compatible with Univac U-200. Both terminals offer 80 characters by 24 or 48 lines, 8 colors and dot addressable graphics--480H by 384V. Prices start at $3995.

ISC's 3rd Quarter Report (December 31, 1982) says that Peter J. Curnin resigned as president of the company. Charles A. Muench is now president (and Chairman of the Board). One wonders what permanent changes this will generate. ISC has been prowling about for small companies (such as Quadram) to buy. Whether they will make some concerted effort to push the Intecolor line in new directions is not clear.

Review – Robot Wars

a game by Steve Reddoch
review by Bill Barlow

Whew! That was a close call! Watch out for that Squirmer! Oh, no here comes a Blaster, you have be careful not to get in line with his diagonal shots! I'd better get out of here! They're closing in on me. There, made it. Oh, I forgot that I can get points by running over the yellow guy (Wanderer). I've got to shoot everything in sight to advance another level. Aiiii! I got shot! --GAME OVER--

This is ROBOT WARS, an exciting new game by Steve Reddoch. Shoot down alien robots, run over the Wanderer to gain points, but don't get in any aliens' way or they will blast you away. ROBOT WARS has excellent graphics, color, and sound. This game can use the keyboard or an ATARI Joystick, but your man can't move diagonally. I prefer the keyboard myself. You receive an extra man at 30,000 points. Sometimes there can be up to 12-16 robots on the screen shooting, beeping, and moving. The robots can pass over each other. You can start the game off with High, Medium, or Low levels. The computer stores the high scores on the disk if you wish. On a scale from 1 to 10 I would rate this game an 8.5. Thank you, Steve, for all this excitement and fun to people with Compucolors. You can get this game for $24.00 from Intelligent Computer Systems, 12117 Comanche Trail, Huntsville, AL 35803.

A Portfolio Record-Keeping Program

by John R. Thirtle
105 Conifer Lane
Rochester, NY 14622

It took me about a year and a half to acquire sufficient programming ability to accomplish what was one of my purposes in buying my Compucolor II: to write a program that would produce a record of a common stock portfolio. It was a frustrating time! I looked at many books and magazines for a suitable program to adapt to my needs. Finding none, I began to teach myself via the manual and any other documentation that I could find. I had had no experience with computers until I got my CCII. Most programs that had some elements of what I needed were a foreign language to me. They were not internally documented and had no hard copy documentation to describe them to the neophyte. One example which will illustrate some of my frustration is a technique for getting a hard copy listing of a program so that I could edit it. Neither the manual for my CCII nor that for my printer gave the simple one line command to do so:

```
PLOT 27,18,4,27,13:LIST:POKE 33265,0 (RET)
```

How simple it would have been to place that one line in the manual or any other document, with a little explanation of its meaning, and to suggest possible variants to suit different printers. Well, so much for griping. It wasn't until the summer of 1980 that I was able to handle simple one-dimensional arrays, tabular printouts, listings, etc. Yet the first program that partly met my needs suffered the same deficiency that I mentioned above: lack of documentation. However, the first version of PORTXX, which was placed in the CHIP User Group Library (Disk #39) was so unrefined that no great harm was done. Since then, I have come to recognize its defects and am describing here a revision of PORTXX that includes DIMensioning of the variables, more REMark statements, INTegers, TABbing, subroutines for setting up the printer, adding trailing zeros in the cents column, right justifying columnar data, etc. Some of this will be elementary to many readers, but I am sure there are enough learners who will profit from it. I want to give credit to Joseph Charles for his book, BASIC Training for Compucolor Computers, and to "THE BASIC Editor" program from Quality Software Associates. Both were very helpful to me.

Program Notes

LINES 1000-1260. These lines are primarily bibliographic and self-explanatory. Line 1080 clears about 300 bytes more than the program needs for the demo provided.

LINES 1290-1380. Variables are defined to closely match their meanings. One of my frustrations in looking at published programs has been to relate variables to their meanings.

LINES 1400-1480. The variable, TN, is defined here where it can be easily seen. The number will have to be adjusted to the size of the individual's portfolio. The variables are all dimensioned to the value of TN. The string variables are required for subroutines mentioned above.

LINES 1590-1720. In a personal portfolio this section can be left as is for

demonstration and a table of personal data assembled. The program can be modified slightly to preserve confidentiality of the personal portfolio. DATA statements give the fixed data on the stocks in the demo portfolio. Company names are not limited to six characters. But the table produced by the program will have to be retabbed if longer names are used. If a stock is split, it is a simple matter to correct the number of shares in the appropriate statement. Base costs include the commissions. Also, these data are completely fictitious; they bear no relationship to my personal holding.

LINE 1550. This is the only data input required to run the demo program. Let's say you need some action.

LINES 1740-1790. More action. If you choose the demo you will use data on prices in lines 2070-2090. If you choose 2 you can input any prices you care to for the 'dummy' companies in 1600-1720, or for your own portfolio.

LINES 1810-2010. This is where you input prices. The queries about corrections are there because I have made errors and wanted to fix them immediately. I often use Q$ in such instances. It fits my aim to relate variable symbols to the words.

LINES 2040-2060. These tell you what you have done and to be patient.

LINES 2150-2170. Likewise. The time required to calculate and integrate the products is about 18 seconds.

LINES 2190-2250. These lines integrate fixed and input values, make the necessary calculations, and integrate the results.

LINES 2280-2760. Adding the trailing zeros to dollar and cents input is a nice touch. It makes the output much easier. It even makes the input easier; prices in whole dollar amounts can just be entered that way. I used to fake such input; for example, inputting $22 as 22.01 or 21.50 as 21.51. Sloppy! The subroutine is from the book by William Barden, Jr., **Programming Techniques for Level II BASIC**, Radio Shack, 1981. The subroutine for right-adjusting the columnar output is that described by Rick Taubold at a users group meeting. Note that the string length specified by L is one unit longer than the longest string expected in a given column because numerical string length includes a real or implied sign (+ or -).

LINE 2780. This line finally sends you to a decision on where you want to have your output (line 3090). It skips you over lines 2800-3050 (next).

LINES 2800-3050. After the table is output, errors are sometimes obvious or you might want to see what change in a price would do to the value of your portfolio. By inserting corrections here, only the new data are manipulated--in a fraction of a second. The REM statements describe what goes on here.

LINE 3090. If you selected the demo option, it took about 18 seconds to get here. In this case I used O$ (for Output) as the query ID. If you choose to use the printer, you fall through to line 3120, which takes you to the subroutine (lines 3510-3540) which does what the REMs say. I generally don't select the printer until I have seen the output on the screen and made sure everything is OK.

LINES 3140-3270. Formatting a table with so many variables is tedious unless one uses an editor program which can easily set and clear tabs. Note that the output is in strings, resulting from the operations described above (trailing zeros, right justified).

LINES 3290-3420. Even though strings have resulted from these operations, the 'unstrung' data remain in memory. It is these on which the calculations are done to produce the totals. Note that the variables BT, VT, and GT have to be initialized; PCT does not because it is calculated from initialized values.

LINE 3440. This line sends the operation to line 3560, whether you have been to the printer or not, and puts the query

about price corrections (line 3460) that we discussed above (lines 2800-3050). En route, you get a message about the string space remaining. Ultimately, you come back to that query in line 3090 and you hit "E", exiting the program, and reading "READY". ◘

```
1000 REM  ** PORTXX ** FOR PORTFOLIO STATUS **
1010 :
1020 REM  ** WRITTEN 1981 BY J. R. THIRTLE
1030 REM  ** 105 CONIFER LANE, ROCHESTER, NY, 14622
1040 REM  ** TEL. 716-467-9676
1050 REM  ** FIRST VERSION ON 'CHIP' LIBRARY DISK #39.
1060 REM  ** REVISED 3,5,83
1070 :
1080 CLEAR 1000:PLOT 14:REM  ** LG CHAR **
1090 PRINT ,,,"PORTXX"
1100 PRINT
1110 :
1120 PRINT ,"PORTXX ('XX' IS FOR PERSONAL ID) PRODUCES
1130 PRINT ,"A TABLE ON THE SCREEN OR PRINTER SHOWING
1140 PRINT ,"THE STATUS OF A STOCK PORTFOLIO AT ANY TIME.
1150 PRINT ,"NO DATA FILES ARE REQUIRED. SAMPLE 'READ DATA'
1160 PRINT ,"ARE SUPPLIED TO ILLUSTRATE OUTPUT.
1170 PRINT
1180 PRINT ,"FOR AN INDIVIDUAL PORTFOLIO THE USER WILL
1190 PRINT ,"CONVERT THE 'DUMMY' DATA TABLE TO REAL DATA,
1200 PRINT ,"CHANGE 'TN' TO THE REAL NUMBER OF COMPANIES,
1210 PRINT ,"AND WILL INPUT THE REAL PRICES.
1220 :
1230 PRINT
1240 PRINT ,"REVISED BY JOHN R. THIRTLE, MARCH 5, 1983"
1250 PRINT
1260 :
1270 INPUT "TO CONTINUE HIT RETURN ";RET
1280 :
1290 REM  ** DEFINITION OF VARIABLES                    **
1300 REM  ** C$=COMPANY NAME (=<6 CHAR)                 **
1310 REM  ** N=NUMBER OF SHARES                         **
1320 REM  ** B=BASE COST OF THE SHARES ( ROUNDED OFF)   **
1330 REM  ** CS=COST PER SHARE                          **
1340 REM  ** P=CURRENT PRICE PER SHARE                  **
1350 REM  ** V=CURRENT VALUE OF THE HOLDING            **
1360 REM  ** G=GAIN                                     **
1370 REM  ** PC=PERCENT GAIN                           **
1380 REM  ** M,D,Y=DATE OF PURCHASE                    **
1390 :
1400 REM  ** DIMENSIONING VARIABLES **
1410 REM  ** TN=TOTAL NUMBER OF COMPANIES IN PORTFOLIO **
1420 :
1430 TN= 13
1440 :
1450 DIM C$(TN),N(TN),B(TN),CS(TN),P(TN),V(TN),G(TN)
1460 DIM PC(TN),M(TN),D(TN),Y(TN)
1470 DIM I$(TN),N$(TN),B$(TN),CS$(TN),P$(TN),V$(TN),G$(TN)
1480 DIM PC$(TN),M$(TN),D$(TN),Y$(TN)
1490 :
1500 REM  ** INPUT **
1510 PLOT 12:REM  ** ERASE SCREEN  **
1520 :
1530 A$= "PORTFOLIO RECORD: PORTXX"
1540 :
1550 INPUT "DATE: MONTH, DAY, YEAR: ";M,D,Y
1560 PRINT
1570 :
1580 REM  ** READ DATA ** CO, SHRS, BASE COST, PURCH DATE **
1590 FOR I= 1TO TN:READ C$(I),N(I),B(I),M(I),D(I),Y(I):NEXT I
1600 DATA  "AAAAA",150,4612,5,8,81
1610 DATA  "BBBBB",100,2008,5,8,81
1620 DATA  "CCCCC",150,2826,6,11,80
1630 DATA  "DDDDDD",200,5025,11,29,82
1640 DATA  "EEEEE",150,4558,10,3,80
1650 DATA  "FFFFFF",100,3272,5,8,81
1660 DATA  "GGGGG",400,3344,7,3,80
1670 DATA  "HHHHHH",100,2346,1,24,83
1680 DATA  "IIII",100,1774,12,31,81
1690 DATA  "JJJJJJ",125,1545,12,31,81
1700 DATA  "KKKKK",100,1698,3,19,81
1710 DATA  "[[[ LL",50,1011,10,3,80
1720 DATA       1",50,1349,3,26,82
1730 :
1740 PRINT ,,"1-DEMO USING DATA PRICES PROVIDED
1750 PRINT ,,"2-INPUT YOUR OWN PRICES
1760 PRINT
1770 INPUT "ENTER YOUR CHOICE:  ";Q
1780 PRINT
1790 ON QGOTO 2040,1810
1800 :
1810 PLOT 12:PRINT ,,"YOU ELECTED  TO INPUT PRICES"
1820 PRINT
1830 PRINT ,"PROGRAM WILL ROUND 3 DECIMALS TO 2 (EG, .125=.13)"
1840 PRINT ,"FOLLOWING ZEROS WILL BE PROVIDED WHERE NEEDED"
1850 PRINT
```

```
1860 PRINT "ID$";TAB( 4)"COMP.";TAB( 11)"PRICE"
1870 :
1880 FOR I= 1TO TN:PRINT I;TAB( 4)C$(I);TAB( 11);
1890 INPUT "";P(I):NEXT I
1900 :
1910 PRINT
1920 INPUT "ANY PRICE  CORRECTIONS (Y/N)? ";Q$
1930 IF Q$= "N"THEN 2140
1940 PRINT
1950 INPUT "ENTER CO ID$ ";I:PRINT
1960 PRINT I;TAB( 4)C$(I);TAB( 15);
1970 INPUT "";P(I)
1980 PRINT
1990 INPUT "ANY MORE PRICE CORRECTIONS? ";Q$
2000 IF Q$= "N"THEN 2140
2010 PRINT :GOTO 1950:PRINT
2020 :
2030 REM  ** PRESET STOCK PRICES **
2040 PLOT 12:PRINT ,,"YOU HAVE SELECTED PRESET PRICES"
2050 PRINT
2060 PRINT ,,,"PLEASE WAIT"
2070 FOR I= 1TO TN:READ P(I):NEXT I:GOTO 2190
2080 DATA  101.00,22,43.375,44.50,55.625,60.75,16.875
2090 DATA  28.25,29.125,21.50,41.00,52.75,103.50
2100 :
2110 REM  ** INTEGRATE VARIABLES **
2120 REM  ** COST/SHR(CS),PRICE(P),VAL(V)
2130 REM  ** GAIN(G),%GAIN(PC) **
2140 PLOT 12
2150 PRINT ,,"YOU HAVE INPUT  PRICES"
2160 PRINT
2170 PRINT ,,,"PLEASE WAIT"
2180 PRINT
2190 FOR I= 1TO TN
2200 CS(I)= INT (B(I)/ N(I)* 10↑ 2+ .5)/ 10↑ 2
2210 P(I)= INT ( 100* P(I)+ .5)/ 100
2220 V(I)= INT (P(I)* N(I)+ .5)
2230 G(I)= V(I)- B(I)
2240 PC(I)= INT (100* G(I)/ B(I)+ .5)
2250 NEXT I
2260 :
2270 REM  ** CONVERT PRICES TO STRINGS, ADD TRAILING ZEROS **
2280 FOR I= 1TO TN:GOSUB 2310:P$(I)= ZZ$:NEXT I:GOTO 2440
2290 :
2300 REM  ** SUBROUTINE: ADD TRAILING ZEROS IN CENTS COLUMN **
2310 ZZ$= STR$ (P(I))
2320 FOR J= 1TO LEN (ZZ$)
2330 IF MID$ (ZZ$,J,1)= "."THEN 2360
2340 NEXT J
2350 ZZ$= ZZ$+ ".00":RETURN
2360 IF J= LEN (ZZ$)- 2THEN RETURN
2370 IF J< > LEN (ZZ$)- 1THEN 2390
2380 ZZ$= ZZ$+ "0":RETURN
2390 ZZ$= LEFT$ (ZZ$,J+ 2):RETURN
2400 :
2410 REM  ** CONVERT VARIABLES TO STRINGS AND RIGHT JUSTIFY **
2420 REM  ** L=EXPECTED NUMBER OF $ CHARACTERS + 1
2430 :
2440 FOR I= 1TO TN
2450 :
2460 REM  ** ID$ **
2470 L= 3:Z$= STR$ (I):GOSUB 3620:I$(I)= Z$
2480 :
2490 REM  ** # OF SHRS **
2500 L= 4:Z$= STR$ (N(I)):GOSUB 3620:N$(I)= Z$
2510 :
2520 REM  ** BASE COST OF SHARES **
2530 L= 5:Z$= STR$ (B(I)):GOSUB 3620:B$(I)= Z$
2540 :
2550 REM  ** BASE COST/SHR **
2560 L= 6:Z$= STR$ (CS(I)):GOSUB 3620:CS$(I)= Z$
2570 :
2580 REM  ** PRICE **
2590 L= 7:Z$= P$(I):GOSUB 3620:P$(I)= Z$
2600 :
2610 REM  ** VALUE **
2620 L= 6:Z$= STR$ (V(I)):GOSUB 3620:V$(I)= Z$
2630 :
2640 REM  ** GAIN **
2650 L= 6:Z$= STR$ (G(I)):GOSUB 3620:G$(I)= Z$
2660 :
2670 REM  ** % GAIN (PC) **
2680 L= 4:Z$= STR$ (PC(I)):GOSUB 3620:PC$(I)= Z$
2690 :
2700 REM  ** DATE **
2710 L= 3
2720 Z$= STR$ (M(I)):GOSUB 3620:M$(I)= Z$
2730 Z$= STR$ (D(I)):GOSUB 3620:D$(I)= Z$
2740 Z$= STR$ (Y(I)):GOSUB 3620:Y$(I)= Z$
2750 :
2760 NEXT I
2770 :
2780 GOTO 3080
2790 :
2800 REM  ** HERE CORRECT PRICES AFTER VIEWING OUTPUT **
2810 :
2820 PRINT
2830 INPUT "ENTER CO. ID$ ";I
2840 PRINT
2850 PRINT I;TAB( 4)C$(I);TAB( 15);
2860 INPUT "CORRECTED PRICE=";P(I)
2870 P(I)= INT ( 100* P(I)+ .5)/ 100
2880 V(I)= INT (P(I)* N(I)+ .5)
2890 G(I)= V(I)- B(I)
2900 PC(I)= INT (100* G(I)/ B(I)+ .5)
2910 GOSUB 2310
```

```
2920 P$(I)= ZZ$:REM  ** TRAILING ZEROS ADDED **
2930 :
2940 REM  ** RIGHT JUSTIFYING CORRECTIONS AND CALCULATIONS **
2950 L= 7
2960 Z$= P$(I):GOSUB 3620:P$(I)= Z$
2970 L= 6
2980 Z$= STR$ (V(I)):GOSUB 3620:V$(I)= Z$
2990 Z$= STR$ (G(I)):GOSUB 3620:G$(I)= Z$
3000 L= 4
3010 Z$= STR$ (PC(I)):GOSUB 3620:PC$(I)= Z$
3020 :
3030 PRINT
3040 INPUT "ANY OTHER PRICE CORRECTIONS (Y/N)? ";Q$
3050 IF Q$= "Y"THEN 2820:GOTO 3080
3060 :
3070 REM  ** DIRECT OUTPUT **
3080 PRINT
3090 INPUT "OUTPUT TO PRINTER, SCREEN, END (P/S/E)? ";O$
3100 IF O$= "S"THEN 3150
3110 IF O$= "E"THEN 3660
3120 GOSUB 3510:GOTO 3160
3130 :
3140 REM  ** FORMAT OUTPUT **
3150 PLOT 12,15:REM  ** ERASE, SMALL CHAR
3160 PRINT A$;TAB( 33);"DATE: ";M;D;Y
3170 PRINT
3180 PRINT "ID#";TAB( 4)"COMP.";TAB( 11)"SHRS";TAB( 16)"COST";
3190 PRINT TAB( 22)"C/SHR";TAB( 29)"PRICE";TAB( 36)"VALUE";
3200 PRINT TAB( 43)"GAIN";TAB( 50)"%G";TAB( 53)"PURCHDATE"
3210 PRINT
3220 FOR I= 1TO TN
3230 PRINT I$(I);TAB( 4)C$(I);TAB( 10)N$(I);TAB( 15)B$(I);
3240 PRINT TAB( 21)CS$(I);TAB( 28)P$(I);TAB( 35)V$(I);
3250 PRINT TAB( 41)G$(I);TAB( 48)PC$(I);TAB( 53)M$(I);
3260 PRINT D$(I);Y$(I)
3270 NEXT I
3280 :
3290 REM  ** CALCULATE TOTALS **
3300 REM  ** BT=TOTAL COST, VT=TOTAL VALUE, GT=TOTAL GAIN **
3310 REM  ** PCT=TOTAL PERCENT GAIN **
3320 BT= 0
3330 FOR I= 1TO TN:BT= BT+ B(I):NEXT I
3340 VT= 0
3350 FOR I= 1TO TN:VT= VT+ V(I):NEXT I
3360 GT= 0
3370 FOR I= 1TO TN:GT= GT+ G(I):NEXT I
3380 PCT= INT (100* GT/ BT+ .5)
3390 PRINT
3400 :
3410 PRINT "TOTALS";TAB( 14)BT;TAB( 35)VT;TAB( 41)GT;
3420 PRINT TAB( 48)PCT
3430 PRINT
3440 GOSUB 3560:REM  ** BACK TO CRT **
3450 :
3460 INPUT "ANY PRICE CORRECTIONS IN TABLE (Y/N)? ";Q$
3470 IF Q$= "N"THEN 3080
3480 PRINT :GOTO 2830
3490 :
3500 REM  ** SET UP PRINTER **
3510 TMP= PEEK (33265):REM  ** SAVE BASIC OUTPUT FLAG **
3520 PLOT 15,27,18,4:REM  ** 2 STOP BITS, 1200 BAUD **
3530 PLOT 27,13:REM  ** DIRECT OUTPUT TO RS-232C PORT **
3540 RETURN
3550 :
3560 POKE 33265,TMP:REM  ** RESET OUTPUT TO CRT **
3570 PRINT "FREE STRING SPACE= ";FRE (B$)
3580 PRINT
3590 RETURN
3600 :
3610 REM  ** RIGHT JUSTIFYING COLUMNS **
3620 Y$= "       ":REM  ** 7 SPACES **
3630 X= L- LEN (Z$)
3640 IF X< = 0THEN X$= "":Z$= X$+ Z$:RETURN
3650 X$= LEFT$ (Y$,X):Z$= X$+ Z$:RETURN
3660 END
```

Moving?

If you're changing your address, please let both the Post Office and us know of your new address. (Tell us your old address and your new one.) We don't want you to miss a single issue of **Colorcue.**

ADVERTISEMENT from

HOWARD ROSEN, Inc
P.O. Box 434
(215)-464-7145 Huntingdon Valley, Pa. 19006

There are some new and interesting products available for CCII owners these days. Software includes Ledger, Database, and word processor. Hardware enhancements cover many areas. We have come across an 8K buffer for the Epson printer which seems to get rid of some of those bugs that may be plaguing you. Here's a quote from a customer's letter.:

> THE THANK YOU IS FOR THE MBS-8K RS232 SERIAL INTERFACE BOARD WHICH YOU SOLD ME. AS YOU CAN TELL BY THIS LETTER THE INTERFACE DIFFICULTIES BETWEEN MY COMPUCOLOR II AND THE EPSON MX-80, WHICH HAVE PLAGUED ME FOR NEARLY THE LAST YEAR, ARE NOW SOLVED. I CAN RECOMMEND THE MBS-8K TO FOLKS HAVING COMPUCOLOR II AND EPSON MX-80 INTERFACE DIFFICULTIES.

For more information and price on that gem of an interface board please feel free to contact us.

We have found another problem in the CCII version V6.78 that seems to affect some printer interfaces. The problem occurs when using the CCII disk drive and the printer. The printer output becomes undependable. The correction requires cutting lands and therefore, we will not print the required correction. The information is available upon request by CCII owners who are able to make such a repair. For a fee we will make the necessary corrections for all others.

We have added to our list of computers the NEC PC-8000 Computer system. This is a very fine machine. It offers many features that brought you to the CCII in the first place, but it offers those features that you wish your CCII could have. The display is user selectable as 36, 40, 72, or 80 columns. The BASIC language is the complete Micro-soft set, including the USING command, sequential files, single, double precision, and TIME and DATE commands. The selection of monitors ranges from green without audio amplifiers to color complete with audio amplifiers. The disk drives are dual, double density with capacity of 160+K per drive. The BASIC ROM residing in lower 32K is switched out when the CP/M operating system disk is booted, and 32K RAM then provides a full 64K memory. The expansion slots can increase memory to 160K, bank addressable. All interfaces are built into the computer: RS232-C, parallel for printer, cassette, and disk through the I/O Unit. The screen display, and printer capabilities include upper, lower case letters, Greek alphabet, graphic symbols. The NEC PC-8023A printer completely complements this computer to provide a very professional machine at PC prices. Please write for complete PC-8000 guide and prices.

Multi–Digit Accuracy

by Neil Brandie
(Reprinted by permission
from CUVIC, Dec. 1982)

Recently whilst writing a program that involved the addition of large dollar values, the answer of course resulted in scientific notation. This was very unsatisfactory, but I was pulled out of my predicament by an article from a very early COLORCUE by D. Woods. [eds note: See also COLORCUE, Vol. II, No. 7, p. 18; Vol. II, No. 8, p. 6; and Vol. III, No. 4, p. 8.] I found that it cleared my problem, and I include it here for others in the hope that it may help them. The routine was designed to give accuracy to 9 digits. The algorithm adds digits from right to left and handles positive numbers only.

Include a line in your program (after a CLEAR statement and before any calls to the subroutine) to define the zeros. Set the number of zeros equal to the number of digits accuracy you want. For example, Z$ = "000000000" gives 9 digit accuracy. Send numbers to the subroutine held in the variables N1$ and N2$ as shown in listing 1. The answer is returned in AN$.

How The Subroutine Works

Lines 1000 to 1040 remove the decimal point in the input dollar amounts and represent the number as if it had been multiplied by 100. The number 123.45, for example, would be converted to 12345. The numbers are then padded out to the left with zeros until the whole number occupies 9 digits. This is done so that the algorithm can handle numbers of different lengths. If you will be inputting numbers with more than two or fewer than two digits after the decimal point, you will have to write a routine to loop through the input number in order to remove its decimal point and associated "+" sign (which will be assumed if the number is input as a numeric and later converted to a string).

Line 1050 sets the carry and the total of the two numbers equal to zero to start with. (The carry is the spillover from the addition of two digits. It is either a "0" or "1".) The index J in line 1060 moves through the digits of the numbers being added from right to left. D1 and D2 are the digits to be added at the current J position. These are both numeric variables.

Line 1090 adds these two digits plus the carry from the previous addition, while C in line 1120 is set equal to the left hand digit. C then becomes the carry for the next addition of digits. Because the value for the variable in A is numeric, it is converted into a string in line 1110. T$ is a temporary holder of this number and has as its first character the sign of the addition. In the second half of line 1110 this sign is removed and the digit alone is stored as the left-most character of the string AN$. When the loop is completed, AN$ will contain the answer to the addition but not including the decimal point. The decimal point is reinserted into the string in line 1140, and the final answer is printed in line 50.

```
 5 CLEAR 1000
10 Z$ = "000000000"
20 INPUT "FIRST NUMBER";N1$
30 INPUT "SECOND NUMBER";N2$
40 GOSUB 1000
50 PRINT AN$
```

```
  60 GOTO 20
1000 L1 = LEN(N1$): L2 = LEN(L2$)
1010 L1$ = LEFT$(N1$,L1-3): R1$ = RIGHT$(N1$,2)
1020 N1$ = L1$+R1$: L1 = LEN(N1$)
1025 N1$ = LEFT$(Z$,9-L1) + N1$
1030 L2$ = LEFT$(N2$,L2-3): R2$ = RIGHT$(N2$,2)
1040 N2$ = L2$+R2$: L2 = LEN(N2$)
1045 N2$ = LEFT$(Z$,9-L2)+N2$
1050 C = 0: AN$ = ""
1060 FOR J = 9 TO 1 STEP -1
1070 D1 = VAL(MID$(N1$,J,1))
1080 D2 = VAL(MID$(N2$,J,1))
1090 A = (D1+D2+C)
1100 B = A-10*INT(A/10)
1110 T$ = STR$(B): AN$ = RIGHT$(T$,1)+AN$
1120 C = INT((D1+D2+C)/10)
1130 NEXT J
1140 AN$ = LEFT$(AN$,7)+"."+RIGHT$(AN$,2)
1150 RETURN
```

Dollar Formatting Subroutine

by Keith Ochiltree
(Reprinted by permission
from CUVIC, Sept., 1982)

This is a handy subroutine to emulate the PRINT USING function of the TRS80 when dealing with money. The routine keeps the cash amount right justified and places the dollar sign in front. The routine is naturally limited to the six digit accuracy of the Compucolor computer and will give you rounding off errors if you use it to calculate values above $9999.00 or $999999.

The program splits the input value TT into TL and TR and makes them into strings. It then checks if the TL is greater than 0 (or a six digit number); if so, it jumps to line 7120; if not, it continues on to read the length of the string and establish the value RX, which is used to print the output right-justified.

```
7000 REM ** CONVERT VARIABLE TO MONEY **
7010 REM TT GIVES STRING DOLLARS (TT$)
7020 REM          RIGHT JUSTIFIED,
7030 REM          FLOATING '$' SIGN
7040 TL = INT(TT/100000)
7050 TR = INT(TT-TL*100000)
7060 TR$ = STR$(TR)
7070 TL$ = STR$(TL)
7080 IF TL<>0 GOTO 7120
7090 RX = LEN(TR$)
7100 TT$ = RIGHT$(TR$,RX-1)
7110 GOTO 7170
7120 RX = LEN(TL$)
7130 TL$ = RIGHT$(TL$,RX-1)
7140 RX = LEN(TR$)
7150 TR$ = RIGHT$("0000"+RIGHT$(TR$,RX-1),5)
7160 TT$ = TL$+TR$
7170 TT$ = "         $"+TT$
7180 TT$+RIGHT$(TT$,10)
7190 RETURN
```

Assembly Language Programming

by Joseph Norris
19 West Second Street
Moorestown, NJ 08057

Part X: Disk Operations

This introduction to disk operations in assembly language makes extensive use of specific routines in ROM and specific locations in RAM (below user space) which are part of the computer operating system, "FCS".[1] The source code names referring to these routines will be those used in the Compucolor System Listing, and the addresses will be given for both V6.78 and V8.79/V9.80 versions.[2] You should keep in mind that these routines are written in 8080 code, just as the programs which utilize them are, and that the CALLs to these routines are no more than shorthand ways of implementing the various functions required; they could each be written out "longhand" if we desired.

FLAGS

RAM addresses that will command our special attention are those that hold "flags" which direct the way certain routines perform. A flag is analogous to the lights in Old North Church ("one if by land and two if by sea...") using numbers instead of lights. A flag, then, is a hexadecimal number in a ROM-specified location in RAM (that is, a location reserved for this flag only) that instructs a routine to behave in a certain way. A set of default values for each of these flags is determined by the operating system at power-up. While flag values are often determined for us in BASIC or FCS>, we must assign them for ourselves in assembly language programming. The magic of these RAM locations is that the routines in ROM and the user both may write to them, giving the user some measure of control over the operation of the computer.

Consider one such flag at address 81F9H called **LOFL** (LO-FLAG). This flag tells the operating system where to direct the printable portion of the response to an FCS> command (such as DIR, DEV, etc.), and directs the output routine, **LO**, accordingly. Table 1 shows how the flag determines the print destination. If we put the value 00H in location 81F9H before issuing a directive to the file control system, for example "DIR", then the computer will print the directory on the CRT.

FCS Command Interpreter

The actual routine that makes this happen is called **FCS** [3], the File Control System Interpreter, located at 25ECH for V6.78 and 0A95H for V8.79/V9.80. The specific instruction to be executed is placed in a "command string", composed by the programmer, whose address is passed

Table I Print Destination; LOFL (81F9H)

FCS COMMAND RESPONSE

OPTION	FLAG VALUE
NO PRINT	0CH
SCREEN PRINT	00H
SERIAL PORT	0EH

to the routine.

Try the program in Listing 1 on your computer. It is operated, after assembly, by typing RUN XFCS from FCS> and will list the directory of the disk in the drive at the moment. Since assumption and ambiguity are the most vicious enemies of learning, we will painfully annotate this first listing and save the abbreviations and jargon for later on. I have borrowed a short program to return you to BASIC after the directory printout--just to keep things interesting. Macroassembler users will have to modify for ORG and END statements as usual. From now on in this series of articles the V8.79/V9.80 addresses will follow the V6.78 in parentheses. You will use one or the other, but not both.

If the program has worked, you can hope that the miracle of assembly-controlled disk operation is at hand. If you put an expendable .BAS program on the disk containing XFCX.PRG, you can resort to useless fun by changing the command string to:

```
CMSTR: DB   'DEL 0:XXXXXX.BAS;01',00H
```

(Note that not all 6 characters in the name are required.)

Since we can perform any FCS> command with **FCS** and **EMESS**, we can load and execute a .PRG file (for example, the MLDP) with the command string:

```
CMSTR:  DB 'RUN MLDP',00H
```

It should be apparent that we are contriving an experiment in program chaining (one program calling another) and that the procedure suggests a method for writing "MENU" programs in assembly language. (Homework?!)

The File Parameter Block

To make use of data files, which are in the form of "sequential files"[4], we must first understand the structure of a byte string called the File Parameter Block (FBP), a data string of 38 contiguous (connected) bytes which contain the parameters necessary for system routine file operations. The data in the FPB are contributed in part by both the programmer and ROM routines. Table II lists the code name, size and location of FPB data within the string. The values of some of these data are usually supplied by the programmer, as follows.

FPB: Value = 0 if program exists on disk, Value = 1 if creating new file. Note that this is the first byte and therefore describes the function FPB (old or new file) as well as the starting address of the entire string--a dual reference and therefore a source of confusion. The remaining parameters in the string are usually retrieved by measuring from this first byte.

FDRV: Drive number containing or to contain the file.

FNAM: Maximum of 6 ASCII characters which are the file name.

FTYP: 3 ASCII characters designating file type (.PRG, .SRC, etc.).

FVER: The file version number; one byte (two hex digits).

FBUF: The starting address of the programmer- assigned disk block buffer for data files, i.e., the place in memory where the data will be loaded and acted upon.

FXBC: The size of the above block buffer.

Table II The File Parameter Block

BYTE NUMBER	1	2	3	9	12	13	15	17	18	20	22
NAME	FPB	FATR	FNAM	FTYP	FVER	FSBK	FSIZ	FLBC	FLAD	FSAD	+
LENGTH	1	1	6	3	1	2	2	1	2	2	1

BYTE NUMBER	23	24	25	27	29	30	31	33	35	37
NAME	FDBK	FDEN	FAUX	FHAN	FFCN	FDRV	FBLK	FBUF	FXBC	FPTR
LENGTH	1	1	2	2	1	1	2	2	2	2

+ SPARE

The above parameters are usually installed by "parsing"[5] a "command string" with a special routine which extracts the data and inserts it in the correct location; they may also be installed by "poking" each value individually through routines you write yourself. After these parameters are installed, the file may be "opened" by another routine, which will fill in many of the remaining parameters, such as:

FDBK: The disk directory block in which the file is located.
FDEN: The entry number in that block for this file.
FATR: The file attribute type (1=Free Space, 2=Permanent File, 3=User File).
FSBK: The starting block on disk of the file.
FSIZ: The size of the file in bytes.
FLBC: The number of bytes used in the last file block on disk.
FLAD: The loading address for an image file.
FSAD: The starting address for an image file.

The address of one or more FPBs (one for each file to be opened; equivalent to FILE "R",1,2, etc. in BASIC) is designated by the programmer in the form of a code name. The FPB will be used to open, edit and close the file, and if the file is new, the routines will update the disk directory automatically. With the FILE Parameter Block data properly installed, only elementary programming is required for normal file functions.

In order to experiment with disk files, we must be able to create a file, enter data in it, write it to disk, open it, read it from disk, display the data (and perhaps print the data)--all steps being required to "close the loop". We must also understand what we are doing as we go. We will begin by examining the containing only printable ASCII characters, including positioning characters such as LF (linefeed), CR (carriage return), etc. The program SOURCE.PRG will use the input routine published in **COLORCUE**, June/July, 1982 to input data. Once stored on disk, your screen editor should permit you to recall the file and view and edit it. When the program is completed you will have constructed an option line which will permit you to (1) create (2) call (3) edit (4) close (5) print (6) end a .SRC file with a length of 124 bytes.[7] (This program may readily be modified for longer files.)

In the meantime you may examine the input routine and think how it might be modified for entry of text, file parameters for opening and creating (will you put file parameters in a different place than text? How will you do that? Do you want error detection for improper file parameters?) and a means of storing the various parameter inputs into the File Parameter Block. Given a **FPB** address, how will you access the various sections? Where will you place your file buffer? How will you display the contents of the file buffer on the screen? Will you want to save code other than the actual ASCII contents of the file data to assist in display functions? It will be helpful self-instruction to consider these things and experiment with them before the next issue of **COLORCUE.**[8] Since the input routine TEST.PRG will form the kernel of our program, you can also prepare an edited copy, renamed SOURCE.PRG on which you can build as we proceed.

Notes

[1] Other operating systems carry such names as CP/M, UNIX, OASIS, etc., trademarks of Digit Research, Bell Telephone and Phase One Software, respectively. The operating system is software, differentiated because of its function of controlling overall computer operations, as opposed to specific utility programs. "FCS" is in ROM because its length permitted storage on currently available ROM chips at the time it was written, and because ROM is always there at power-up. CP/M is much longer and therefore had to be delivered on disk. Since ROM chips now have larger capacities, even CP/M might soon become available in ROM. [On the other hand, having the operating system in ROM means that you can't change it here and there to suit your own needs. --Eds.]

[2] By "source code name" we mean designations like "KBDFL" and "KBRDY", which do not appear in ROM, of course, but are used by the programmer for legibility when composing source files. The assembler translates these names into addresses.

[3] Since a number of things are labelled with the letters "FCS", these articles will refer to the operating system as "FCS", the FCS computer mode as "FCS>" and the routine in ROM as **FCS.**

[4] The file type .RND is a reserved extension of BASIC, and we will not be using it here. In assembly language we can duplicate it, of course, and make it even more flexible if we desire. We have the power, in our program, of reserving another extension, say .RSV, to be handled in a prescribed way by our program.

[5] "Parsing" is the procedure by which a routine separates the parts of a file specification by looking at the spacing and punctuation in the string. That is the only way the routine can tell where a drive number ends and a file name begins, and why FCS> is so unmerciful if you enter it incorrectly! You routinely "parse" data in everyday life, such as this name and address string: John B. Adams, 223 Westover St., North Hills, NJ 08058.

[6] 3651 owners will find a file called BASLST.SRC on their Sampler Disk which contains a clear example of file routines in assembly. It is a long file and will not fit all at once on a screen editor. The best way to examine this file is to assemble it with a simultaneous printout. (With the MacroAssembler, type BASLST.SRC-L and watch with glee!)

[7] If you are eager to proceed now, a copy of the completed listing for SOURCE.SRC may be obtained by sending me your name, address and a check for

LISTING I; XFCS - Calling the DIRECTORY

```
;First list locations of flags and subroutines

EMESS   EQU     262DH(0AD6H)  ;a subroutine. Have faith.
FCS     EQU     25ECH(0A95h)  ;another subroutine
LOFL    EQU     81F9H         ;a flag location

        ORG XXXX              ;adjust to taste

XFCS    LXI     H,LOFL        ;We put the address of the flag in the HL register
        MVI     M,00H         ;and place 00H in that Ram slot.

        LXI     H,CMSTR       ;We give HL the address of the command string in
                              ;our program, because the FCS routine is coded to
                              ;look for it in HL.

        CALL    FCS           ;We actuate the FCS Command Interpreter which
                              ;takes our command, 'DIR' from the command string
                              ;at program address CMSTR and initializes the
                              ;proper code steps to retrieve directory
                              ;information from the disk, placing it where the
                              ;number in LOFL says to put it - in this case,
                              ;on the screen. FCS, then, requires two inputs:
                              ;a flag in LOFL and a command string pointer in
                              ;HL. When FCS has completed its work, it will
                              ;present us with some parameters. If a disk error
                              ;has occured, the B register will hold an error
                              ;code reference number ( 0=no error). The data in
                              ;the A, DE, and HL registers will be gone and the
                              ;8080 flags will be: Z-set if no errors, Z-reset
                              ;if an error has occured, C-set if a comma was
                              ;found in the command string.
```

$3.00 to cover printing and mailing costs. The listing will be printed in **COLORCUE** as we examine it in the coming installments.

[8] Recommended texts: For descriptions of disk and utility routines: Dewey, Dale, **Advanced Programmer's Manual.** D2 Engineering, 7284 High View Trail, Victor, NY 14564. $15, looseleaf. This is not a tutorial, but an elegant outline of routines in ROM with a guide to their use. For the intermediate and experienced programmer.

For 8080 utility routines: Findley, Robert and Edwards, Raymond, **Scelbi "8080" Software Gourmet Guide and Cook Book**, 2nd ed., 1978. Scelbi Computer Consulting, Inc., Milford, Connecticut 06460. $10.95. May be out of print, but worth a try! Review of instruction set and special routines involving stack pointer, sorting, conversion of data, tables, I/O processing and floating point operations.

For instruction set and assembler operations: **INTEL 8080A Assembly Language Programming** manual. Literature Dept., Intel Corporation, 3065 Bowers Avenue, Santa Clara, CA 95051. About $5.00. A very transparent and ordered presentation.

For computer organization, utility routines, I/O hardware and software procedures: Rony, Peter R., **8080A Microcomputer Interfacing and Programming**, 2nd ed., 1982. Howard W. Sams and Co., Inc., 4300 West 62nd Street, Indianapolis, Indiana 46268. $17.95. A revision of the "8080A Bugbook", this volume has one of the best software annotations I've seen. In spite of their use of octal code, the beginning programmer will have many critical questions answered in these pages.

```
   CALL EMESS                  ;You might as well know now about this error
                               ;message generator, the priest who always speaks
                               ;in red! This routine takes the number FCS left in
                               ;the B register and generates and prints an error
                               ;message if B<>0. If there was no error the
                               ;routine just RETurns. EMESS will print according
                               ;to LOFL.

;Now exit the program graciously with a jump to BASIC

BASJMP: LXI     H,8299H
        XRA     A
        MOV     M,A             ;This short and very useful routine
        INX     H               ;is taken from an article by
        SHLD    80D4H           ;M.A.E. Linden of Toronto, Ontario
        MOV     M,A             ;and published in FORUM INTERNATIONAL
        INX     H               ;VOL I, 5-6, p74
        MOV     M,A
        INX     H
        SHLD    80D6H
        LXI     H,81DFH
        JMP     0046H(1F2CH)

CMSTR:  DB      'DIR',00H       ;Our command line may contain any valid FCS>
                                ;command (just one per line!) but the line must
                                ;end in 00H as shown or an error will occur.
                                ;Unlike OSTR you may not place PLOT codes in the
                                ;command string, such as those to change color or
                                ;erase the page.

        END
```

FREPOST COMPUTERS, INC. 431 East 20th Street 10-D
New York, New York 10010 PHONE 212-673-6476
Source TCI251 Micronet 70210,374
PRICES FOR COMPUCOLOR AND ISC COMPATIBLE EQUIPMENT & SOFTWARE
MARCH, 1983

ADD ON 16K RAM BOARD (INCREASE A 16K CCII OR 3621 TO 32K RAM)
ASSEMBLED AND TESTED WITH RAM CHIPS 109.00
UPGRADE 8K OR 24K TO 32K CALL OR WRITE

8K PROM BOARD (INTEL 2716 TYPE EPROMS FOR USE IN 4000H-FFFFH)
ASSEMBLED AND TESTED-NO PROMS INCL. 49.00

64K BANK SELECTABLE ROM BOARD
SELECTS VIA SOFTWARE CONTROL UP TO 56K (D) OF EPROM IN 8K SEGMENTS. PLUGS INTO THE ADD-ON ROM SOCKETS INSIDE THE COMPUTER OR WITH THE EXTERNALIZER BOARD IT WILL OPERATE OUTSIDE THE COMPUTER. IT UTILIZES TI 2532 TYPE EPROMS. BUILT IN IS A SOCKET FOR ADDITION OF AN 8K BOARD E.G.,8K SINGLE BANK BOARD OR DEVLIN R A M BOARD. THIS COMBINATION GIVES YOU THE FULL 64 K OF PROM. 50 PIN BUS CONNECTOR ALLOWS FOR INSTALLATION WITH NO SOLDERING TO LOGIC BOARD.
ASSEMBLED AND TESTED---WITHOUT PROMS 249.00
ABOVE IN KIT FORM 199.00
50 PIN BUS CONNECTOR 10.00

BUFFERED EXTERNALIZER BOARD FOR CCII, 3621 OR 3650
THIS BOARD AND CABLE COMBINATION ALLOWS USE OF YOUR PLUG IN BOARD ON THE EXTERIOR OF THE MACHINE. IT IS MANDATORY FOR THE 3650 AND 3621.
ASSEMBLED AND TESTED ONLY 59.95

'THE' BASIC EDITOR (SEE FORUM VOL 2 NO 1 PP 11-12 FOR REVIEW)
PROM VERSION IN 2532 OR 2716 PROMS 89.00
PURCHASED WITH 8K PROM BOARD (A&T) 109.00
PURCHASED WITH 64K BANK BOARD (A&T) 269.00

LOWER CASE PROM WITH STANDARD CCII OR ISC GRAPHICS
(CAN USE CAPS LOCK SWITCH OR ADD ON TOGGLE SWITCH)
EPROM WITH LOWER CASE 35.00

ENHANCED OPERATING SYSTEM ROM FOR 6.78
ADDS 4 NEW JUMPS TO ACCESS EPROM AREA WITHOUT POKES TO USER VECTOR. PERSONALIZED WITH THREE INITIALS AT NO EXTRA CHARGE
EPROM WITH NEW OPERATING SYSTEM 29.00

THE FOLLOWING PROGRAMS CONSIST OF ONE BANK OF 2532 OR 2716 EPROMS
COM-TRONICS (tm) SOFTWARE IN NEW EPROM VERSIONS
1)TERM II COMMUNICATIONS PACKAGE 89.95
2)CTE, FORMATTER, SPEED 144.95
3)NEWBUG, CTA 109.95
4)CRC,DFM 79.95
5)PRINT2, CLIST,LLIST,LDAFIL 109.95
6)TERMII, SRC/BAS, BAS/SRC, FILMRG 154.95
7)CTA 69.95
8)NEWBUG, SORT, RENUM 109.95

BILL GREENE SOFTWARE IN EPROM
1)SUPER MONITOR PLUS 79.95

JIM HELMS SOFTWARE IN EPROM VERSIONS
1)EDITOR/ASSEMBLER 90.00
2)WISEII 8080 EMULATOR 75.00
3)DISK EDITOR 60.00
4)GENERAL LEDGER SPREADSHEET 110.00
5)SOURCE DISASSEMBLER 130.00

FREPOST COMPUTERS, INC (tm) EPROM VERSIONS

1)AT LAST! DIRECTORY PROGRAM BY BILL POWER	59.95
2)DISK BASED VERSION	39.95

RICK TAUBOLD & BILL GOSS' NEW REAL TIME STAR TREK

FEATURING GAME SAVE AND ALL NEW GRAPHICS	25.00

A FIRST FOR THE COMPUCOLOR/INTECOLOR! DOUBLE PRECISION MATH! CAN BE USED TO GIVE UP TO 16 DIGIT PRECISION MATH OPERATIONS ON ADD, SUBTRACT, MULTIPLY AND DIVIDE. TRANSCENDENTAL FUNCTIONS TO COME. AVAILABLE IN ROM FOR YOUR BANK SELECT OR 8K PROM BOARD. CAN BE ADDED TO MOST PROM PACKAGES. THIS FAST MACHINE LANGUAGE MODULE IS CALLABLE FROM BASIC, DO AWAY WITH THE PENNY ERRORS FOREVER!

STANDALONE IN ROM PACK	59.95
INCLUDED IN ROM PACK WITH OTHER PGMS	39.95

THE QUADRAM LINE OF PRINTER SPOOLERS FREE YOUR COMPUTER FROM NEEDLESS WASTE OF TIME WHILE EVEN FAST PRINTERS PRINT. DUMP YOUR DATA AT 9600 BAUD AND LET THE MICROFAZER HANDLE THE PRINTING CHORES WHILE YOU CONTINUE CRUNCHING. STANDALONE UNITS POWERED FROM YOUR PRINTER'S PARALLEL INPUT, ALLOW RESET AND RECOPY FROM FRONT PANEL.

64K SERIAL IN/PARALLEL OUT	269.95
8K SERIAL IN/PARALLEL OUT	189.95

ALL CONFIGURATIONS OF INPUT AND OUTPUT MODE ARE AVAILABLE. CALL!

THE ANGEL IS ANOTHER PRINT SPOOLER, AND IS LIKE THE MICROFAZER, BUT HAS UNIVERSAL INPUT/OUTPUT, MORE FRONT PANEL CONTROLS ALLOWING PRINT INTERRUPT, REPRINT FROM PAGE X, AND MUCH MUCH MORE

64K SERIAL OR PARALLE IN AND OUT	289.95

MULTI COMPUTER USERS NOTE :
ALL QUADRAM, STB, AST, PRINCETON GRAPHICS, AMDEK, TECMAR, AND RELATED MANUFACTURERS PRODUCTS ARE AVAILABLE AT GREAT SAVINGS FROM FREPOST FOR OTHER TYPE SYSTEMS.

WE SAVED THE BEST FOR LAST.....

NEW FOR 1983!! THE OKIDATA MICROLINE 92 9X9 LETTER QUALITY PRINTER COMES WITH THESE STANDARD FEATURES ---->

2000 BYTE BUFFER, PARALLEL INTERFACE, 7 CHARACTER FONTS PLUS CORRESPONDENCE FONT (NOT JUST A DOUBLE STRIKE DP FONT). ALSO, A SMART VERTICAL FORMAT UNIT, REAR OR BOTTOM PAPER FEED, FRICTION OR 9 1/2" PIN FEED, 6 CHARACTER WIDTHS, ENHANCED PRINT MODES AND

**** DOT ADDRESSABLE GRAPHICS INCLUDED AT NO EXTRA CHARGE! ****

YOU CAN ALSO CREATE YOUR OWN FONT AND DOWNLOAD IT TO THE M/L 92! 160CPS PRINT SPEED IN DP MODE, SHORT LINE SEEKING BIDIRECTIONAL PRINTING.

MICROLINE 92 80 COL 9X9 MATRIX PARALLEL	540.00
MICROLINE 92 AS ABOVE, SERIAL INTERFACE	620.00
MICROLINE 93 132 COLUMN 9X9 MATRIX PAR'LL	900.00
MICROLINE 93 AS ABOVE, SERIAL INTERFACE	980.00
MICROLINE 80 80 COL 7X9 MATRIX 80CPS	350.00
FREPOST SCREEN DUMP PROGRAM (CCII GRAPHICS CHAR SET)	39.95

FOR INFORMATION ON THESE AND OTHER PRODUCTS, PLEASE CALL OR WRITE TODAY. DELIVERY ON MOST ITEMS IS FROM STOCK.

INSTALLATION ASSISTANCE AND SERVICE IS AVAILABLE AT MODEST COST FOR YOUR COMPUCOLOR OR ISC COMPUTER.

Compucolor Disk Drive Improvements

by John Newman
PO Box 37
Darlington,
Western Australia 6070

While it is not practical to make major improvements (such as capacity) to the Compucolor disk drives, there are a number of useful, low cost modifications. Three of these are: (1) write protect switch, (2) motor run-on, and (3) dual speed switch. (Note: all of these require complete removal of the disk controller board, cutting of circuit tracks and soldering of new components.)

Write Protect Switch

This addition provides hardware protection against accidental writing of any data to the currently loaded diskette. Deletion of files is also inhibited. With the switch in the Write Protect OFF position, all normal reading and writing can occur. With the switch in the Write Protect ON position, only reading can occur. If you, too, have hoards of children playing Space Invaders, this switch should help prevent those mysterious disk erasures. Attempts to write to the disk with Write Protect ON will produce FCS errors: EVFY, EFWR or EDEL.

Parts required:

1. Miniature SPDT toggle switch.
2. 4700 ohm, .25 watt resistor.
3. Length of hookup wire.

The logic and component location diagrams are given in Figures 1 and 2. The switch should be fitted to a 1/4" hole on the front panel of the disk drive.

Motor Run-On (V8.79 only)

This modification is the most difficult to install but probably has the most value. When the disk is selected for a read or write, the drive motor starts up. A delay of up to a second occurs before the motor reaches correct speed and data is transferred. As soon as the data transfer is complete, the drive motor stops. Subsequent reads or writes require the same motor startup delay. An easily demonstrated example of this is listing of a long directory. There is a distinct pause between reading of each directory block.

With the motor run-on circuit added, the drive motor will continue for three seconds. If any read or write occurs within this time there is no startup delay. Listing of long directories and loading of .LDA programs with the run-on fitted take about half the normal time. The reason that this modification currently applies only to V8.79 is because V6.78 FCS has a built in delay of one second. This is being looked into right now. Hopefully a simple update to V6.78 FCS PROM will remove this problem.

Parts required:

1. 1000uF 6.3V electrolytic cap.
2. Signal diode 1N4148 or similar.
3. 4700 ohm .25 watt resistor.

Installation involves cutting the track between pins 2 and 3 of UA3, drilling three small holes and soldering the components as shown in Figures 3 and 4.

Dual Speed Switch

Those Compucolor owners who regularly buy or trade software are aware of compatibility problems between drives. They also know that reading a disk at a

slightly slower speed than the original writing speed overcomes many of the reading problems. This is usually achieved by drilling a hole in the disk cabinet, giving screwdriver access to the speed control potentiometer. The trouble with this is the difficulty of resetting to correct speed.

This modification is simply a switch to set the disk speed at one of two fixed speeds. By connecting a 2200 ohm resistor across half of the speed pot, the RPM drops by five. Use the SPEEDO program or the strobe indicator to set the normal speed (300). Reading speed for "foreign" disks can then be set to 295 RPM at the flick of a switch. By reducing the value of the resistor, the difference between the two speeds increases. A 1000 ohm resistor gives a difference of 8 RPM. The switch should be fitted to a 1/4" hole on the front panel of the disk drive.

Parts required:

1. Miniature SPDT switch.
2. 2200 ohm .25 watt resistor.
3. Length of hookup wire.

Install the parts as shown in Figures 5 and 6.

Figure 1

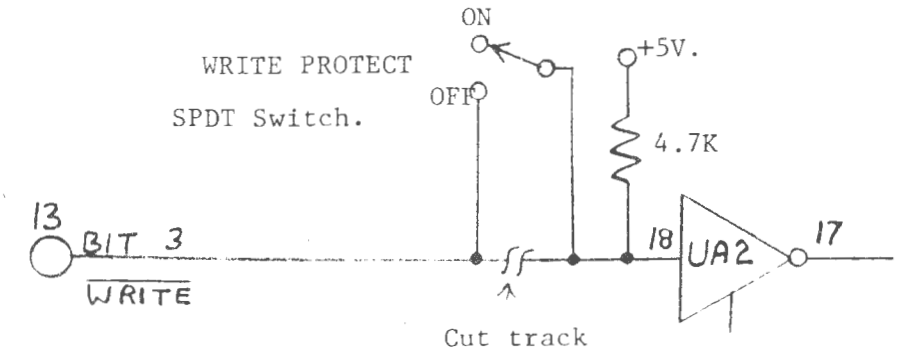


Figure 2

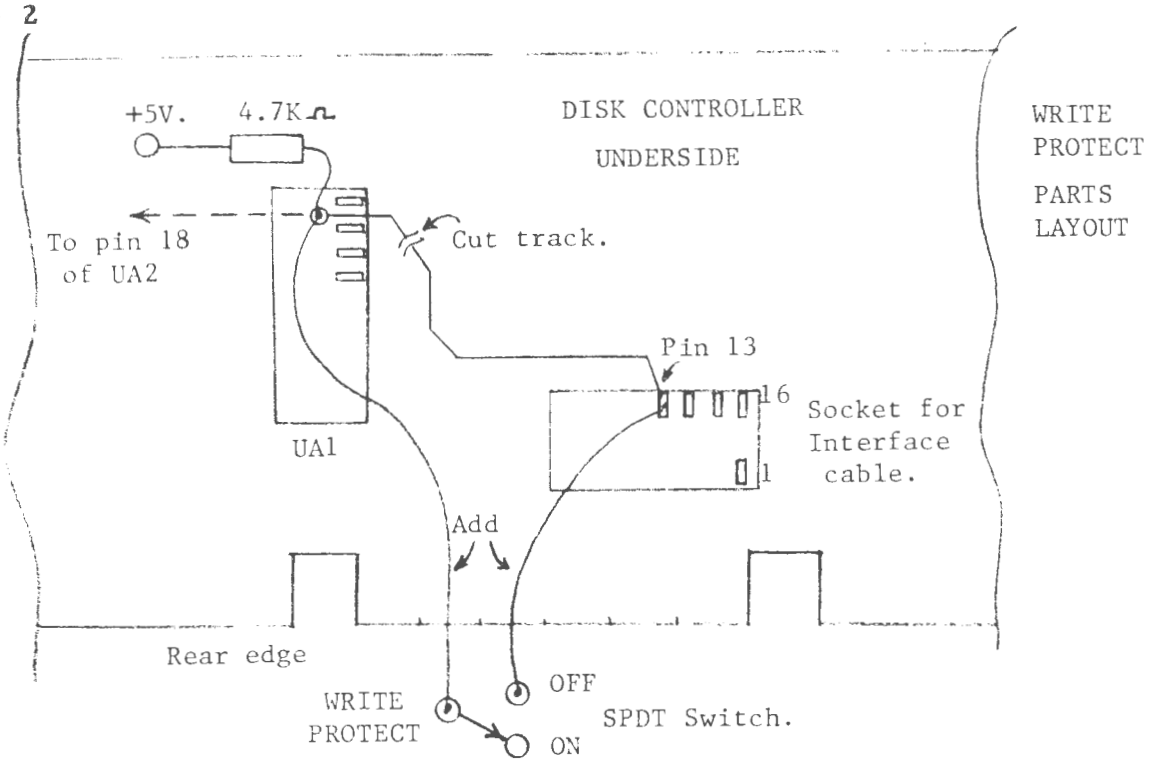


Figure 3

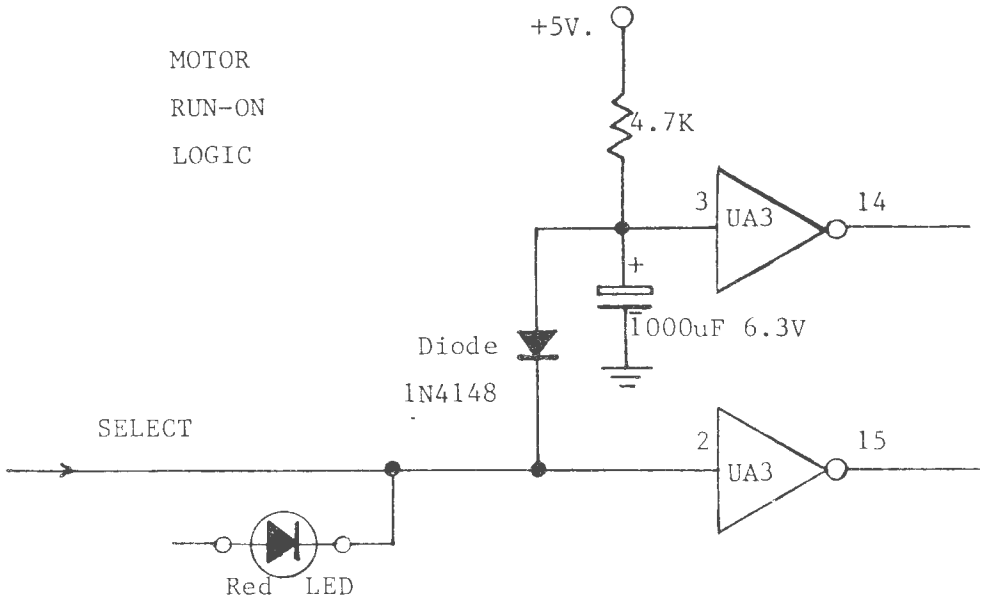


Figure 4

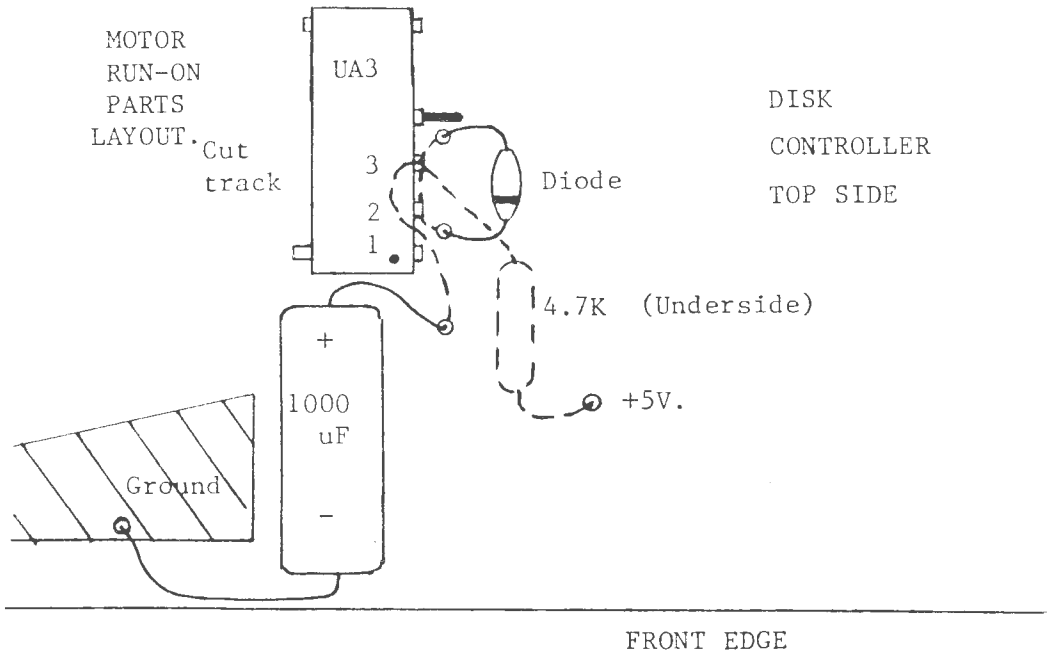


Figure 5

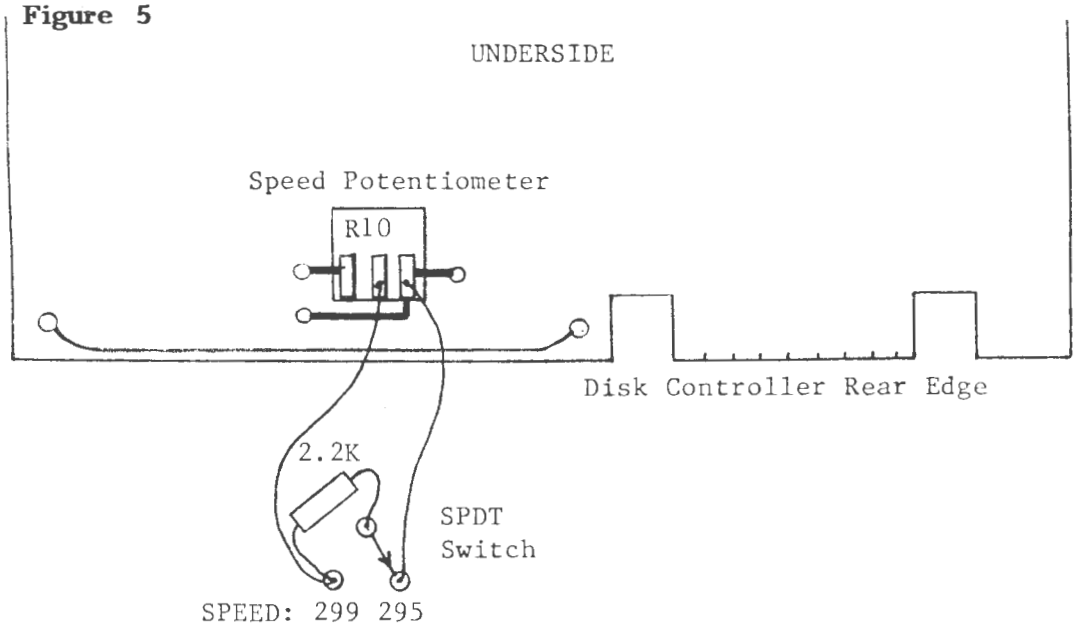
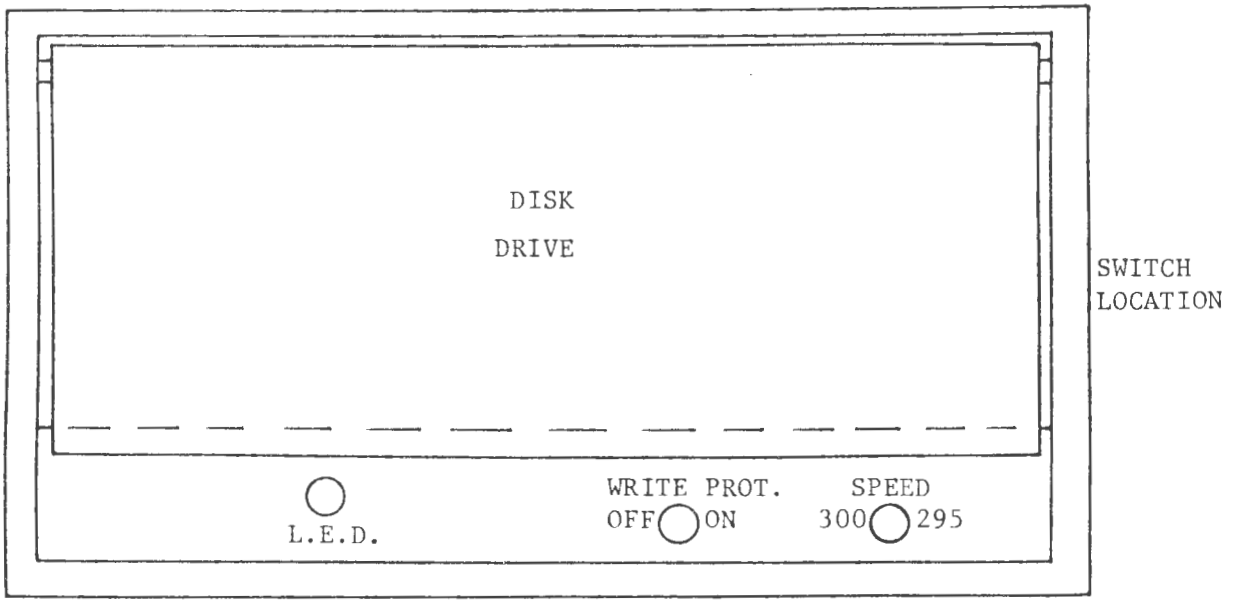


Figure 6



Cueties

by Steve Smith
498 Brown Street
Napa, CA 94558

```
190 PLOT 12
200 FOR I=0 TO 120
210 FOR T=-5 TO 5+I
215 C=INT(7*RND(1))+I
217 PLOT 6,C
220 X=INT(10+T*RND(1))+1
230 Y=INT(10+T*RND(1))+1
240 PLOT 2,X,Y,255
250 NEXT T:NEXT I
```

The Okidata Microline 84A Printer

by James L. Helms
1121 Warbler
Kerrville, TX 78028

I love the new Okidata Microline 84A. The maximum baud rate is 4800 with a serial interface, but when it is going at 200 characters per second you would never know it. It has true logic seeking character positioning, where the print head does not have to return to its home position prior to printing the next line. This really increases throughput.

The ribbons are typewriter styled and travel in both directions like a typewriter ribbon does. You could even use an ordinary ribbon except that it would void the 90 day warranty. The ribbons which Okidata supplies are impregnated with a head lubricant, so I wouldn't advise using plain ribbons.

The printer is capable of dot graphics, underlining, proportional spacing and user defined character sets. Unfortunately, the instruction manual leaves a lot to be desired. They do not, for example, explain how to download your own character set. Nor do they fully explain the printer's graphics capabilities. I was only after much playing around that I got the graphics to work. I still haven't figured out how to use my own character set. The manual also assumes that the user understands a lot of esoteric terms, especially those involved in the command sequences. It takes a while to figure out what's supposed to be going on. There is a section on setting what they call "channels". As far as I can make out, channels are a kind of vertical tab, the positions of which can be set by a command sequence giving line numbers. Vertical tabbing is then done by commanding vertical movements according to the channel numbers instead of the line numbers. Weird.

"N.L.Q." is another term which some users might not be familiar with. It stands for "Near Letter Quality". When in this mode, each print line is printed twice, once in the usual way, and the second time shifted by one half dot. It gives less of a dot matrix look to the letters, but it slows throughput to about 100 characters per second. (Still pretty fast.)

One of the big mysteries in their manual is the reference to "Single CSF Exhaust". If you can figure that one out, let me know.

There are a few things I don't like about the physical design of the printer. The power switch is located in the rear. I would have preferred the front console. And I don't like having to unscrew the top cover in order to change the dip switch settings. Usually, though, this can be overcome by downloading your own command sequence.

All in all, I am very pleased with the printer; I think it's worth every penny I paid for it. (Anyone want to buy a used Base 2 printer, cheap?)

Dip Switch Settings

Switch	Setting	Use
Ø1	—	odd/even parity
Ø2	ON	no parity
Ø3	OFF	8 bit word
Ø4	OFF	4800 cps
Ø5	ON	"
Ø6	ON	"
Ø7	OFF	line feed on LF
Ø8	ON	(always)
Ø9	ON	simplex
1Ø	ON	"
11	ON	"
12	OFF	(always)
13	OFF	"mark" when busy
14	OFF	(always)
15	OFF	RS232
16	ON	two wire

Cable Connections for CCII and Microline 84A

CCII pin	Microline pin	Signal		
3	3	RD	(XMITD DATA)	
14	7	SG	(SIGNAL GROUND)	
15	11	SSD	(BUSY)	
	4	RTS	(REQUEST TO SEND)	JUMPER 1
	5	CTS	(CLEAR TO SEND)	JUMPER 1
	6	DSR	(DATA SET READY)	JUMPER 2
	20	DTR	(DATA TERMINAL READY)	JUMPER 2

6 LPI TEST
LINE NUMBER 1 line number 1
LINE NUMBER 2 line number 2
LINE NUMBER 3 line number 3
LINE NUMBER 4 line number 4
LINE NUMBER 5 line number 5

8 LPI TEST
LINE NUMBER 1 line number 1
LINE NUMBER 2 line number 2
LINE NUMBER 3 line number 3
LINE NUMBER 4 line number 4
LINE NUMBER 5 line number 5

6 LPI UNDERLINE TEST
THIS IS UNDERLINING ... this is underlining
THIS IS UNDERLINING ... this is underlining
THIS IS UNDERLINING ... this is underlining
THIS IS UNDERLINING ... this is underlining
THIS IS UNDERLINING ... this is underlining

EMPHASIZED TEXT
TESTING ... testing
TESTING ... testing
TESTING ... testing
TESTING ... testing

SUPERSCRIPT / SUBSCRIPT

SUPERSCRIPT SUPERSCRIPT SUBSCRIPT SUBSCRIPT
SUPERSCRIPT SUPERSCRIPT SUBSCRIPT SUBSCRIPT
SUPERSCRIPT SUPERSCRIPT SUBSCRIPT SUBSCRIPT
SUPERSCRIPT SUPERSCRIPT SUBSCRIPT SUBSCRIPT
SUPERSCRIPT SUPERSCRIPT SUBSCRIPT SUBSCRIPT

5, 10.5, 12, 8, 17 CPI TESTS

1234567890
1234567890
1234567890
1234567890
1234567890
1234567890

FASBAS – A Basic Compiler

a mini-review

by D. B. Suits

Anyone with contacts in any other user groups, through newsletters or friendships, or who subscribes to COLORCUE, knows that there is a BASIC compiler loose. (A compiler is a program which takes another program (in BASIC, in this instance) and converts it into one that runs, or runs faster than an interpreted version. An interpreter is a .... this is going nowhere .) Called FASBAS and written by Peter Hiner of Great Britain, it produces code that runs from 2 to 5 times faster than the original BASIC versions. The price is $20 (apparently cheaper when user groups buy in bulk).

One evening we popped into ye olde computer roome to take a look at FASBAS and what it could do. The FASBAS disk comes with a demo BASIC program on it which you can run and then compile using FASBAS in order to appreciate the difference in speed. We thought, though, that we'd give it a "real life" test. Suits's BOUNCE program seemed a good choice. We read the FASBAS documentation and made note of the various restrictions. There are some things that FASBAS doesn't like to see in BASIC programs which it compiles (improperly exited FOR-NEXT loops, for example). Fortunately, BOUNCE did not violate any of the restrictions, so we loaded the FASBAS disk, ran FASBAS (it takes a few seconds to get itself organized), and told it to please compile BOUNCE. The first pass which FASBAS makes produces a quasi-assembly language source file and stores it on disk. The source file is very long indeed--so large that for a 10K or 16K BASIC program, the generated source code will take up most of the disk space. While the source code is being generated, FASBAS tells you whether it has encountered any illegitimate instructions in your BASIC program. These might be synax errors or else instructions (such as random file handling instructions) which FASBAS can't accommodate.

In the case of BOUNCE, all went well. Next we invoked the FASBAS assembler, a non-standard assembler (because the source code which FASBAS generates is non-standard) which makes two passes over the source code in a manner similar to the Compucolor assembler. Part way through this assembly, an error was reported in red. The error message, however, was a bit cryptic, so we had to look it up in the documentation manual. Oops! No list of error messages! Oh, well, it looked like a standard assembler error message indicating an ambiguous label reference. (If you don't know Compucolor assembler errors, you'll be out of luck here.) So. An ambiguous label? How can that be? WE didn't write the source code, FASBAS did. Why was FASBAS generating ambiguous labels? Back to the original BOUNCE program to see what we could see. It took us a while, but we finally discovered that FASBAS was indeed generating some ambiguous labels. Specifically, BOUNCE has an array called SC(). FASBAS labels that with a prefix "A" (for "array", I suppose). Unfortunately, the BASIC keyword "ASC" looks just like the label for array SC, and this was the source of the assembler's confusion. We got out THE BASIC editor, changed all references in BOUNCE from SC() to CS(), submitted the new version to FASBAS, ran the FASBAS assembler again, and.... Oops! Still an error. This one concerned the BASIC line "LOAD MENU:RUN". The manual says that's OK, and FASBAS didn't flag an error there, but the assembler did. Strange. So we took that line out of BOUNCE, re-compiled and reassembled, and.... Lo! It worked!

The assembler generated a .LDA file on disk which we loaded (.LDA files take a while) and then saved back out as a .PRG

(i.e., a true machine language) file. Then we ran BOUNCE.PRG. Yep! There was the program, up and running, and it was considerably faster than the interpreted BASIC version--so much faster, in fact, the the little ball which bounces around was now bouncing a bit too fast; it would have been appropriate to go back to the original program and put a few small delay loops in and recompile. So FASBAS has a few restrictions and a few bugs. Big deal. It's a working BASIC compiler for only $20! That's nothing to sneeze about. If you're interested in obtaining a copy, write to Peter Hiner, 11 Penny Croft, Harpenden, Herts AL5 2PD, England.

Classified Advertisements

For Sale Compucolor II, V 6.78, 32K, 117 key keyboard. Soundware, disks include games, BASIC Editor, Assembler, Personal Data Base. Manuals included. Excellent condition. Asking $1300 or best offer.

Harry Trueheart
8 Old Farm Circle
Pittsford, NY 14534
(716) 586-7906

For Sale Compucolor II, V8.79, 32K, purchased 11/81. Includes programming and maintenance manuals, Text Editor, Compucalc, Personal Data BAse, Fredi, Assembler, Format, games, and a file of **Colorcue**. $1500.

Also - new 16K RAM module $75.

Steven Forshay
1321 Webster Street D114
Alameda, CA 94501
(415) 522-5935

User Group Notes

We are trying to build a Compucolor software library for exchange of programs on a low cost reciprocity basis. We would like to receive offers regarding programs you have created and wish to share. For information, please write:

Bernard Lohman
Deurloostraat 103
1078 HW Amsterdam
The Netherlands

COMING NEXT

Focus on your screen with:

- Screen memory problems
- 4096 colors?!
- Adjustment advice
- A bar cursor

Turn to your disk drives with:

- Handy disk utilities in BASIC
- Assembly language routines

And more.

BULK RATE
U.S. POSTAGE
PAID
Rochester, N. Y.
Permit No. 415

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

Colorcue

April/May 1983

$2.

Colorcue

A bi-monthly publication by and for
Intecolor and Compucolor Users

Editors:
Ben Barlow
David B. Suits
Compuserve: 70045,1062

April/May, 1983
Volume 5, Number 5

COLORCUE is published bi-monthly. Subscriptions are US$12/year in the U.S., Canada and Mexico, and US$24 (includes air mail postage) elsewhere. Some back issues are available. All editorial and subscription correspondence should be addressed to **COLORCUE**, 161 Brookside Dr., Rochester, NY 14618, USA. All articles in **COLORCUE** are checked for accuracy to the best of our ability, but they are NOT guaranteed to be error free.

Editors' Notes

Several recent letters have inquired about the meaning of the mysterious number on the mailing label of their **Colorcue**. That number used to be the issue number of the last issue valid on your subscription. Unfortunately, that number was never explained or used anywhere in the magazine, so you have a right to be puzzled. Having explained that number now, you'll notice that it's changed. Since its purpose is to tell you how many issues you have to go, and a computer knows what the current issue is and what your last issue will be, it seems that the computer should subtract and tell you, simply, how many issues are left on your subscription. That's what the number does now, and that's what "to go" means.

If your "to go" is zero, you'd best renew to continue with **Colorcue**; you've received your last issue. When you renew, rest assured that your investment is secure. **Colorcue** maintains a bank balance large enough to refund subscribers' unused subscription dollars one for one, should the magazine cease publication. In simpler terms, you'll get your money's worth, or you'll get it back.

Still on the topic of subscriptions, let us issue a call for renewals. If you'll remember last year, when we did this, we waited until we had enough renewals in hand to make sure we still had a viable operation, and then began to work on the Aug/Sept issue. Because many of you, like at least one of our editors (guesses?) were late with their renewals, that issue was delayed (about 2 months). We've never gotten caught up, even though our intentions are superb. Please help us avoid that situation this year, and renew early. We promise not to get any further behind, and promise that you won't lose any money in the deal.

REMINDER

Beginning with Volume 6 (Aug/Sept), the subscription rate for **COLORCUE** will be US$18 in North America and US$30 elsewhere.

FASBAS Update

Peter Hiner has improved his FASBAS BASIC compiler. It no longer has all the previous limitations mentioned in last issue's review. And apparently all the bugs have been corrected. I have been using it successfully for several months now, and I can report that I am thoroughly delighted.

The price of FASBAS was erroneously given as $20 (US). Please note that the true price is $25 (US). Peter has graciously filled some orders even when the purchaser sent only $20. Let's be fair to him and his creation, though: if you bought FASBAS for only $20, please send him the remaining $5. After all, the more we encourage Peter, the more likely we are to see something else wonderous emerge from his computer room. Besides, you'll have to agree that $25 is CHEAP!

Speaking of FASBAS, have you been wondering how Peter came to write it in the first place, and are you curious to learn how it works? Then don't miss our next issue, wherein Peter begins a multi-issue article which answers these questions.

ISC News

Years ago many Compucolorists found a friend in Gene Boughey, who both gave us advice and produced programs for us on his own or in association with ISC. Now Gene has been promoted to manager of graphics systems sales at Intecolor. Congratulations, Gene!

No doubt one of Gene's duties will be to promote Intecolor's new VHR19 graphics terminal. It has 1024H by 1024V bit mapped color display (1024H by 768V viewable). Eight of 4096 colors may be displayed. The VHR19 has graphics commands such as point, line, polyline, rectangle, circle, arc, polygon fill, copy, zoom and pan. Four sizes of Tektronix character sets are included, plus two graphics character sets, one of which is user-definable. The detached keyboard has 113 keys; 36 are programmable function keys. Serial port, DMA channel, auxiliary I/O and printer ports are also included. The introductory price (until October 31) is $3995.

Assembly Language Programming

by Joseph Norris
19 West Second Street
Moorestown, NJ 08057

Part XI: Program Design and Parsing File Names

In the last issue we discussed the FCS Interpreter and introduced ourselves to the File Parameter Block. We now begin a three part series of articles that describe the construction of a program to create, open, edit, print and close a source file (.SRC). We have chosen a .SRC file because it contains only "printable" data, that is, hex numbers which may be displayed on the screen in the form of alpha-numeric and associated "typewriter" characters--all these commonly referred to as "ASCII" characters. In addition to this set is a subset, with hex numbers below 20H, representing, by standard agreement, such necessary printing procedures as a carriage return, line feed, form feed, etc. [1] All these characters are simple to insert and remove from a disk file, and are not likely to "run amok" when handled improperly. They make a good starting point for disk file experience.

Program Design

Of importance equal to specifics on the use of file routines in ROM, in this series, is the procedure by which our program is designed. Initial assembly language programming, proceeding from a background in BASIC, confronts a seductive insecurity; how can I manage without line numbers? Where on earth do I begin and with what? The traditional answer to this kind of questioning is "the flow chart", which, for many of us, is somewhat like being advised that the best way to extinguish a fire is to swallow it.

We can begin by acknowledging that the primary dynamic between computer and operator is communication. The CPU places most exacting requirements on the format for giving it instructions and data; the operator is entitled to the same specificity, and this is determined by the computer's messages to the operator, primarily through a sequence of option lines. [2] When this sequence is complete, the programmer has a "flow chart" of a very useful kind, containing a synopsis of questions to be asked and answers to be given, and formatted screen displays on which this exchange will take place. At this point the "outline" program may be assembled and run, stepping through each stage to evaluate order and completeness. One may then proceed to fill in the operations prescribed from this outline. Equally important, the programmer should now have a very clear idea as to which details of communication will be included with the program, and which need explanation in an "operator's manual". Your programming is not complete without such a manual, and its preparation is an essential part of "professional" program generation. Listing 2 is an example of an outline "flow chart" for our entire program. It will "run" as shown, and you may step through the option lines as though you were operating the final program. Note that portions of the program, yet unwritten, have a place reserved for them in the listing (marked '---------'). You may use these locations as a guide for inserting the "program modules" as we construct them throughout these articles. The listing includes the modules we will need from David Suits's input routine (somewhat modified for our use). I suggest you construct this routine, as shown, and test it, before entering the modules described in these articles.

```
;ERROR CORRECTION -

;Please make the following corrections to
; your listing:

; 1) Change label EDSA1 to EDS1A
; 2) Modify the last line to DSP3 to make
; two lines as follows:

;                DB      '   TEXT'
;     DSP3A:     DB      6,7,3,0,12,239

; 3) In module PARSE, change the first
; line to read -

;       LXI  B,DFLT ;set default type

; NOTE: The default type may be any three
; letters you wish, except for reserved
; file types such as LDA, PRG, BAS, RND,
; COM, etc. You may also create a 'null'
; file, with no type at all. Why not set
;    DFLT:    DB     '  '  and try it!



;LISTING II
;
;SOURCE.PRG;A program to Open, Enter
;text, Close and Print a .SRC file
;with parsing added

;=======================================
;INTERIM OPERATING INSTRUCTIONS:

;For this program as it is so far, select
;option to view corresponding display and
;see synopsis of program. Press RET to
;proceed. Exit with CPU/RESET. Many calls
;to GETCHA will be replaced by operating
;routines. For now they permit viewing
;displays before moving on. At end of the
;PRINT function you will view the error
;message displays. Keep pressing RET until
;the master option line reappears. Areas
;in the listing bordered by (;------) are
;routine insertion areas for later use.

;=======================================
;EQUATE AREA - put all EQU's below: v8.79
;    & V9.80 shown - v6.78 in parentheses

        KBCHAR  EQU     81FEH   ;(ram)
        LO      EQU     17C8H   ;(3392H)
        OSTR    EQU     182AH   ;(33F4H)
        STACK   EQU     8FFFH   ;(choose)

;=======================================
;ENTRY POINT -

        ORG     8200H   ;or your own spot

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

SOURCE: ;Initial steps explained later!

        LXI     H,0     ;clear HL
        DAD     SP      ;move in FCS
        SHLD    FCSSP   ; stack and save
        LXI     SP,STACK;our stack!

        MVI     A,0C3H  ;*input routine
        STA     81C5H   ; jump to CHRINT
        LXI     H,CHRINT
        SHLD    81C6H
        MVI     A,1FH
        STA     81DFH

SETUP:  LXI     H,CLR   ;clear page, page
        CALL    OSTR    ; mode,small char
        MVI     B,42    ;clear FS,MCHAR,
        LXI     H,FS    ; and INBFPR
XX1:    MVI     M,0
        DCR     B       ;decr counter
        INX     H       ;incr pointer
        JZ      CLBF    ;when finished
        JMP     XX1     ; back for next

CLBF:   MVI     B,129   ;clear INBUF
        LXI     H,INBUF
XX2:    MVI     M,' '   ;put 'space'
        DCR     B       ;down counter
        JZ      OPTION  ;when done
        INX     H
        JMP     XX2     ;clear next

;=======================================
;MAIN PROGRAM - complete!

OPTION: CALL    BLLN    ;border blue
        LXI     H,DSP1  ;point to text
        CALL    OSTR    ;and print it
        CALL    GTCHA   ;wait for select.
        CPI     49      ;  (ASCII '1')
        JZ      OPENA
        CPI     50      ;  and vector
        JZ      TEXT    ;  program to
        CPI     51      ;  selected
        JZ      CLOSEA  ;  routine
        CPI     52
        JZ      PRINT
        CPI     53      ;  (ASCII '5')
        JZ      ENDIT
        JMP     OPTION  ;if illegal entry

;=======================================
;BEGIN SUBROUTINES - insert below as we
;derive them (I use alphabetical order)

BCKSP:  MOV     A,C     ;*input routine
        ORA     A
        JZ      XX3
        MVI     A,1AH   ;doesn't erase
        CALL    LO      ; character -
        DCX     H       ; nice for
        DCR     C       ; editing text
XX3:    RET

BLLN:   LXI     H,DRLNB ;option border
        CALL    OSTR    ; blue
        RET

BOX:    LXI     H,BXDSP ;draw text window
        CALL    OSTR
        RET

CHRINT: PUSH    PSW     ;*input routine
        XRA     A
        STA     81FFH
        POP     PSW
        RET

CLOSEA:                 ;close file, one
        CALL    BLLN    ;routine for new
        LXI     H,DSP4  ;and one for old
        CALL    OSTR
;---------------------------------------
        CALL    GTCHA   ;routine here
;---------------------------------------
        JMP     SETUP   ;restart

ENDIT:                  ;'END' routines
        CALL    BLLN
        LXI     H,DSP6
        CALL    OSTR
```

```
        CALL    GTCHA   ;wait for key
;---------------------------------------
;vector to appropriate subroutine, in
;this space
;---------------------------------------
        JMP     OPTION

ERROR2: CALL    RDLN
        LXI     H,EDS2  ;'no file'
        CALL    OSTR
        CALL    GTCHA

ERROR4: CALL    RDLN
        LXI     H,EDS4  ;'file in buffer'
        CALL    OSTR
        CALL    GTCHA

ERROR5: CALL    RDLN
        LXI     H,EDS5  ;'FCS problem'
        CALL    OSTR
        CALL    GTCHA
;---------------------------------------
;remainder of routine in this space
;---------------------------------------

ERROR6: CALL    RDLN
        LXI     H,EDS6  ;'bad file name'
        CALL    OSTR
        CALL    GTCHA
        JMP     OPTION  ; - all done!

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

GTCHA:  XRA     A       ;get 1 kbrd char
        STA     KBCHAR  ; see COLORCUE
XX4:    LDA     KBCHAR  ; v2#8 p9 and
        ORA     A       ; Jun/Jul 82 p24
        JZ      XX4
        RET

INPUT:  LXI     H,INBUF ;*input routine
        SHLD    INBFPR
        MVI     C,0
INPUT1: CALL    GTCHA
        CPI     0DH     ;(CR)
        JZ      INPUT4
        CPI     1AH     ;left arrow?
        JZ      INPUT2
        CPI     19H     ;right arrow?
        JZ      INPUT5
        CPI     ' '     ;space?
        JC      INPUT1  ;note 'JC'
        CPI     129     ;maxKey+1
        JNC     INPUT1  ;ignore
        MOV     B,A     ;save character
        PUSH    H       ;WHAT?! Look at
        MOV     A,C     ; maximum chars
        LXI     H,MCHAR ; allowed by the
        CMP     M       ; caller - OK?
        POP     H       ;get pointer
        JNC     INPUT1  ;no! ignore
        MOV     M,B     ;yes, print it!
        INX     H
        INR     C
        MOV     A,B
        CALL    LO
        JMP     INPUT1

INPUT2: CALL    BCKSP
        JMP     INPUT1

INPUT4: LDA     MCHAR   ;look for 128
        CPI     128     ; if so -
        RZ              ; return, else
        MVI     M,0DH   ; add CR for name
        RET             ; and return

INPUT5: MOV     A,C     ;right arrow
        CPI     128     ;end of line?
        JZ      INPUT1  ;yes, ignore!
        MVI     A,19H
        CALL    LO      ;do it! adjust
        INX     H       ; counts
        INR     C       ; and
        JMP     INPUT1  ; return

OPENA:                  ;open file: will
        LXI     H,CLR   ;branch to one
        CALL    OSTR    ;routine for new
        CALL    BLLN    ;files, and one
        LXI     H,DSP2  ;for old files
        CALL    OSTR
;---------------------------------------
;File open routine in this space.
        CALL    GTCHA
;---------------------------------------
        CALL    BOX
;---------------------------------------
                        ;display file here
;---------------------------------------
        JMP     OPTION  ;and return

PRINT:                  ;print to RS232
        CALL    BLLN
        LXI     H,DSP5A
        CALL    OSTR
;---------------------------------------
        CALL    GTCHA   ;baud routine
;---------------------------------------
        CALL    BLLN
        LXI     H,DSP5B
        CALL    OSTR
;---------------------------------------
        CALL    GTCHA   ;'prepare' routine
;If you want to send setup string to the
;printer, do it here, with LXI H,SETPR
;and S1OUT.
;---------------------------------------
        CALL    BLLN
        LXI     H,DSP5C ;signal operator
        CALL    OSTR
;---------------------------------------
        CALL    GTCHA   ;printing routine
; replaces this area: will normally JMP
; OPTION when finished, but go on to view
        JMP     ERROR2  ;error messages
;---------------------------------------

RDLN:   LXI     H,DRLNR ;paint border red
        CALL    OSTR    ;draw it
        RET

TEXT:                   ;enter text
        CALL    BLLN
        LXI     H,DSP3
        CALL    OSTR
;---------------------------------------
        CALL    GTCHA   ;routine here
;---------------------------------------
        JMP     OPTION  ;and return

;=======================================
;BEGIN STRING STORAGE -


BXDSP:  DB      6,2,3,0,12,11,3,0,13,11
        DB      2,0,82,242,127,82,255
        DB      2,0,68,242,127,68,255,239

CLR:    DB      6,2,12,27,24,15
        DB      'SOURCE.PRG - SOURCE FILE'
        DB      '  PROGRAM',239

DRLNB:  ;see COLORCUE v2#7 p16 & Manual
        ;for repeat string with OSTR here:
```

```
        DB      6,4,3,0,2,30,237,8
        DB      '________',238,3,0,4,30
        DB      237,8,'eeeeeeee',238,29,6
        DB      2,3,0,3,11,3,0,3,239

DRLNR:  DB      6,1,3,0,2,30,237,8
        DB      '________',238,3,0,4,30
        DB      237,8,'eeeeeeee',238,29,6
        DB      1,3,0,3,11,3,0,3,'ERROR  '
        DB      6,8,'      ',3,52,3,6,72
        DB      ' RET ',6,1,'  ERROR'
        DB      6,7,239

DSP1:   DB      20,'OPTION ',6,72,'>>>>>'
        DB      6,3,'      1',22,'OPEN ',19
        DB      '2',22,'TEXT ',19,'3',22
        DB      'CLOSE ',19,'4',22,'PRINT '
        DB      19,'5',22,'END      ',6,72
        DB      '<<<<<<',6,4,' OPTION',3,64
        DB      3,239

DSP2:   DB      20,'OPEN   ',6,8,'      '
        DB      6,3,'      ENTER FILE NAME'
        DB      '  ',31,'>> ',6,35,'       '
        DB      6,67,' << ',6,8,3,52,3
        DB      ' RET ',6,4,'   OPEN',6,35
        DB      3,37,3,239

DSP3:   DB      20,'TEXT   ',6,8,'      ',6,3
        DB      3,23,3,'EDIT FILE BELOW -'
        DB      3,52,3,6,72,' RET ',6,4
        DB      '   TEXT',6,7,3,0,12,239

DSP4:   DB      20,'CLOSE  ',6,8,'      ',6,3
        DB      3,25,3,'CLOSING FILE',3,52,3
        DB      6,8,'      ',6,4,'  CLOSE'
        DB      3,64,3,239

DSP5A:  DB      19,'BAUD ',6,72,'>>',6,3
        DB      ' SELECT:  ',17,'1',22,'110 '
        DB      17,'2',22,'150 ',17,'3',22
        DB      '300 ',17,'4',22,'1200 ',17
        DB      '5',22,'2400 ',17,'6',22
        DB      '4800 ',17,'7',22,'9600  '
        DB      6,72,'<<',6,3,' BAUD'
        DB      3,64,3,239

DSP5B:  DB      20,'PRINT  ',6,8,'      ',6,3
        DB      3,23,3,'PREPARE PRINTER -'
        DB      3,52,3,6,72,' RET ',6,4
        DB      '  PRINT',3,64,3,239

DSP5C:  DB      20,'PRINT  ',6,8,'      ',6,3
        DB      3,27,3,'PRINTING -',3,52
        DB      3,6,8,'       '
        DB      6,4,'  PRINT',3,64,3,239

DSP6:   DB      17,'END    ',6,72,'>>>>>'
        DB      6,7,3,20,3,'EXIT TO ',17,'1'
        DB      23,'FCS ',17,'2',23,'BASIC '
        DB      17,'3',23,'CRT',3,52,3,6,72
        DB      '<<<<<<',6,1,'    END',3,64,3
        DB      239

EDS1:   DB      3,0,8,239

EDSA1:  DB      6,7,3,12,3,'           ERROR'
        DB      ' MESSAGE BELOW          '
        DB      6,72,'  RET ',6,0,3,64,3,239

EDS1B:  DB      6,2,3,0,8,11,3,0,9,11,239

EDS2:   DB      3,20,3,'NO FILE IN BUFFER - '
        DB      'OPEN',3,64,3,239

EDS4:   DB      3,18,3,'FILE IN BUFFER - CL'
        DB      'OSE FIRST     ',3,64,3,239

EDS5:   DB      3,15,3,'DISK PROBLEM. WANT'
        DB      ' FSC ERROR? (Y/N)  ',6,72
        DB      '  ?  ',3,64,3,239

EDS6:   DB      3,25,3,'BAD FILE NAME'
        DB      3,64,3,239

EXSTR:  DB      6,2,12,239

;=======================================
;BEGIN DATA STORAGE - order is important!

FCSSP:  DW      1       ;FCS stack pointer
FS:     DS      1       ;File open/cl flg
MCHAR:  DS      1       ;*max char allowed
INBFPR: DS      2       ;*input routine
FPB1:   DS      38      ;File para block!
INBUF:  DS      128     ;*input routine

        END     SOURCE
```

Parsing

Any use of a disk file involves a file specification which must always contain, at least, a file name of one to six characters. The additional parameters, which you have already used many times, are optional; these are the drive number (CD0:, etc.), the file extension (.BAS, ;0A, etc.). When these additional parameters are not supplied by the programmer, the operating system supplies "default" values. These parts of the file specification are needed by many other file handling routines, so they must then be completed into a place easily identified--the File Parameter Block.

Entry of a file specification is the first step to file access. The parts of the file specification may be "poked" into the FPB one part at a time, or you may type in any or all parts of the file specification string (such as 0:NEW.SRC;04 but always at least the file name!) and let the operating system "parse" it and place each parameter into the FPB. The guide to accurate parsing is the order and punctuation of the specification. In fact, if the order and punctuation are incorrect, the parsing will fail, which is why FCS> is so ruthless if you don't do it right.

There are several system routines that will parse a file specification string, supplying default values where they are

missing. These are **PFSPC**, which parses any file type with a default type of your choosing; **PSFSP**, which looks for a ".SRC" file; **PPFSP**, which looks for ".PRG"; and **PNDSP**, which looks for a 'null', or ". " in the file specification. **PFSPC** is the all-purpose routine.

Before calling **PFSPC**, you need to prepare the following:

a) A file specification string, located at a specific address in memory.
b) A string containing the three letter file default extension (no period!) located at a specific place in memory.
c) A File Parameter Block, located at a specific place in memory.
d) The address of the file specification string in the HL registers.
e) The address of the default file extension in the BC registers.
f) The address of the File Parameter Block in the DE registers.

If NAMST is the address of our specification string ("NEW;02"), DFLT the address of our default type string ("SRC"), and FPB1 the address of our File Parameter Block, Listing 1 shows how it would work. If the parsing has been successful, the Carry and Zero flags will be reset (i.e. = 0). Carry will be set if an error occurred, and Zero flag will be set if no version number was included in NAMST. A missing version number need not produce an error, however. The BC registers will hold any FCS error code, DE will still contain the FBP1 address, HL will be unpredictable, and the accumulator will hold the version number (the default ";01" if none was given in NAMST).

The File Parameter Block will now have the following slots filled:
FDRV, the drive number; **FNAM**, file name; **FTYP**, file extension or type; **FVER**, version number, and **FHAN**, file handler routine. This latter is the system routine selected by the computer for the appropriate kind of disk drive named. [3]

At this point the computer has done nothing but verify a valid file specification and place the parameters in the FPB. No access to the disk directory has yet been made, and the computer does not yet know if the file exists, or if there is room for it on the disk currently in the drive.

Entering the File Name

We must provide a means for entering the file name, and we may use the input routine and INBUF to hold it, with the address of INBUF acting as NAMST in our example above. If we want to limit the entry to the file name only, we might choose to limit the input to six characters ("user-friendly"?) and let parsing supply the rest of the file specification. This is an elegant procedure but not without its complications.

When the file specification is parsed by one of the system routines, it will examine for drive number, name, extension and version, using the punctuation as a guide and inserting default values as it goes. If all the parameters are not given in the specification string, the file parameters that are given, however few, must be marked at their end by a carriage return, or by another character whose value is less than 20H, which will signal the parsing routines to end their search. Otherwise they will continue forever. So if our file specification is to consist of the file name only, some provision must be made for adding a CR to the end of the name, and so our storage space must have room for at least seven characters. (Note, too, that upper case characters must be used for the file name.)

In our program, INBUF is used for the disk file buffer and the text buffer, both. As a text buffer it serves to hold the file name entry and the text file we have created. In order to differentiate between file name and text, we use the data storage address, MCHAR (Maximum CHARacters) to flag the difference: 6 for a file name entry, and 128 for text entry. No carriage return is entered in the buffer if text is being stored. But look at INPUT4 in the listing. If MCHAR does not hold 128, the maximum text characters, this subroutine will add a CR to the string, assuming it is a file name. This way our parsing will be valid. You will add some lines to subroutine OPENA that put 6 into MCHAR before parsing. (See Listing 3.)

Program Modules

When you have assembled and tested Listing 2 and are certain it is running properly, insert the modules of Listing 3 as directed. Unfortunately, they will not access any disk files yet, but we will be ready to do that when we continue. You

may run the program and enter a file name, which will return you to OPTION. ERROR6 is a file error trap of your own. It may be displayed by entering a bad name, such as "$TEST" or ".....". (The file name cannot begin with non-alpha-numeric characters. Can it be all numbers? Try some different combinations and see.) INPUT will not accept a seventh character, but it doesn't abort; rather, it waits to see if you have any further editing to do. Only pressing RETURN will terminate the name entry.

The modifications to the input routine are for fun--not necessarily improvements. The cursor movement procedure here is handy for correcting only one letter, early in the text, while permitting you to see the entire buffer contents. Some of the material in Listing 2 is not yet used, but will be accessed later. It will do no harm to enter the entire listing now.

As a final word, if you have had difficulty getting started in writing assembly language programs, either in formatting them or learning to apply the instructions, it is invaluable to set up a routine such as Listing 2. After typing all those LXIs and CALL OSTRs with a few other instructions to round then out, your proficiency will increase rapidly. For many of us, it has been true that the initial "terror" has given way in an unexpected moment, and suddenly we were programming with great ease.

In the next article, we will proceed with file opening routines and learn how to create a new .SRC file on disk.

NOTES

1 If you examine the ASCII listing in your programming manual you will find only the carriage return listed, the others being assigned to colors, plot functions and so on. This assignment is meaningful to your CRT display only. A printer will interpret these numbers differently, as we shall see later on.

2 Such non-computational material is sometimes referred to as "gingerbread", meant as a disparaging term. It is, nevertheless, what makes computers useful. An option line lets the operator select or escape from a function offered by the software. Its design is very critical for operator acceptance and efficiency, often determining--alone--whether a program sells or not.

3 If you have a 3651 computer, this might be a 5" or 8" floppy disk (MD, DM, FD or DF, single or double sided disk types).

```
LISTING III: PARSING MODULES

1) Add to EQU section:

        PFSPC   EQU     14ADH   ;(3077H)

2) Add following to replace 'CALL GTCHA'
in module labelled OPENA:

        LXI     H,MCHAR ;set maximum char
        MVI     M,6     ; to six for name
        CALL    INPUT
        CALL    PARSE

3) Add this module:

PARSE:  LXI     H,DFLT  ;set parameters
        LXI     D,FPB1
        LXI     H,INBUF
        CALL    PFSPC   ;parse file name
        JC      ERROR6  ;print error message
        RET

4) Add to STRING STORAGE space:

DFLT:   DB      'SRC'
```

```
;LISTING IV: Add to SOURCE.SRC

;For EQUATE area - v9.80/V8.79 shown,
;                  v6.78 in parentheses

        CLSEQO  EQU     156CH   ;(3136H)
        EMESS   EQU     0AD6H   ;(262DH)
        GTBYT   EQU     1662H   ;(322CH)
        INSEQO  EQU     151DH   ;(30E7H)
        OPEN    EQU     11E1H   ;(2DABH)
        PTBYT   EQU     1680H   ;(324AH)
        RESET   EQU     0B48H   ;(26A5H)
        RWSEQI  EQU     14FCH   ;(30C6H)

;Add these modules after routine OPTION -


CLOSEA: CALL    FSO     ;check if file open
        CALL    BLLN
        LXI     H,DSP4
        CALL    OSTR
;---------------------------------------
        LXI     H,FPB1  ;initialize new
        CALL    INSEQO  ; file
        CALL    WRITE   ;write data
        LXI     H,FPB1
        CALL    CLSEQO  ;close new file
        XRA     A       ;change flag=0
        STA     FS
;---------------------------------------
        JMP     SETUP   ;restart

CREATE: CALL    RESET   ;reset drive
        LXI     H,FPB1  ;put "new" file
        MVI     M,1     ; code in FPB1
        CALL    OPEN
```

```
        JC      ERROR5  ;FCS error
        LXI     H,FS    ;set 'file open'
        MVI     M,1     ; flag
        CALL    NMDSP   ;display name
        CALL    BOX     ;draw text window
        JMP     CLBF    ;erase buffer

ERROR1: LXI     H,EDS1  ;position cursor
        CALL    OSTR    ; to display
        CALL    EMESS   ; FCS error
        LXI     H,EDS1A ;print new
        CALL    OSTR    ; message
        CALL    GTCHA
        LXI     H,EDS1B ;erase error
        CALL    OSTR
        JMP     OPTION

;In ERROR2 and ERROR4, add a last line to
; read 'JMP OPTION'.

ERROR5: CALL    RESET   ;expanded routine
        CALL    RDLN
        LXI     H,EDS5  ;'FCS problem'
        CALL    OSTR
        CALL    GTCHA   ;wait for decision

;---------------------------------------
        CPI     89      ;is it 'Y'?
        JZ      ERROR1  ;yes! print it
        JMP     OPTION  ;no! return


FSC:    LDA     FS      ;check file closed
        CPI     0
        JNZ     ERROR4
        RET

FSO:    LDA     FS      ;check file open
        CPI     1
        JNZ     ERROR2
        RET

NMDSP:  LXI     H,WRFL  ;print file name
        CALL    OSTR
        LXI     H,INBUF ; from buffer
XX5:    MOV     A,M
        CPI     0DH     ;cr?
        RZ              ;yes, return
        CALL    LO      ;no, print it
        INX     H
        JMP     XX5

OPENA:  CALL    FSC     ;check file closed
        LXI     H,CLR
        CALL    OSTR
        CALL    BLLN
        LXI     H,DSP2
        CALL    OSTR
;---------------------------------------
        LXI     H,MCHAR ;set max chars
        MVI     M,6     ; to six
        CALL    INPUT   ;get file name
        CALL    PARSE   ; and parse it
        LXI     H,FPB1  ;set 'existing'
        MVI     M,0     ; file mode and
        CALL    OPEN    ; try to open
        JC      CREATE  ;doesn't exist
        CALL    NMDSP   ;OK! display name
        LXI     H,INBUF ;put address of
        SHLD    FPB1+32 ; buffer in FBUF
        LXI     H,FPB1+34
        MVI     M,128   ;length in FXBC
        LXI     H,FPB1
        CALL    RWSEQI  ;set file pointers
;---------------------------------------
        CALL    BOX     ;paint text box
;---------------------------------------
        LXI     H,FS    ;set file status
        MVI     M,1     ; flag to 'open'
        CALL    READ    ;file into buff
;---------------------------------------
        JMP     OPTION  ;for next command

;In routine PRINT, change the last line
; from 'JMP ERROR2' to 'JMP OPTION'

READ:   LXI     H,DSP3A ;position cursor
        CALL    OSTR    ; in text box
        LXI     H,FPB1
        MVI     B,128   ;counter
XX9:    CALL    GTBYT   ;process one byte
        CALL    LO      ;print it
        DCR     B
        RZ              ;when done
        JMP     XX9     ;for next one

TEXT:   CALL    FSO     ;check file status
        CALL    BLLN
        LXI     H,DSP3
        CALL    OSTR
;---------------------------------------
        LXI     H,MCHAR ;set maximum chars
        MVI     M,128   ; to 128
        CALL    INPUT   ; and get data
;---------------------------------------
        JMP     OPTION  ;then return

WRITE:  MVI     B,128   ;count bytes
        LXI     D,INBUF ;point to source
        LXI     H,FPB1
XXA:    LDAX    D       ;put byte in Acc
        CALL    PTBYT   ;write it
        INX     D       ;adjust pointer
        DCR     B       ; and counter
        RZ              ;all done!
        JMP     XXA     ;go for next one

;---------------------------------------
;For data storage area, add -

WRFL:   DB      6,4,3,38,0,'FILE OPEN: '
        DB      6,7,239
```

```
;LISTING V: Add to SOURCE.SRC

;For EQUATE area - v9.80/v8.79 shown.
:                 v6.78 in parentheses

        ADHLA   EQU     194EH   ;(3518H)

        CRTSET  EQU     01EBH   ;v9.80
                                ;(020FH) v8.79
                                ;(37C0H) v6.78

        ESC1    EQU     053AH   ;(2420H)
        S1OUT   EQU     17F9H   ;(33C3H)

;Add these modules after routine OPTION -


BASJMP: LXI     H,EXSTR ;clear screen
        CALL    OSTR    ; green color &
        LXI     H8299H  ; exit BASIC, by
        XRA     A       ; M.A.E. Linden,
        MOV     M,A     ; FORUM INT'L
        INX     H       ; V1,#5-6 p74
        SHLD    80D4H
        MOV     M,A
        INX     H
        MOV     M,A
        INX     H
        SHLD    80D6H
        LXI     H,81DFH
        JMP     1F2CH
```

```
CRTJMP: LXI     H,EXSTR ;exit CRT mode
        CALL    OSTR
        JMP     CRTSET

ENDIT:  CALL    FSC     ;chk file closed
        CALL    BLLN    ; for 'end'
        LXI     H,DSP6
        CALL    OSTR
        CALL    GTCHA   ;wait for key
;---------------------------------------
        CPI     49      ;'1'=FCS
        JZ      FCSJMP
        CPI     50      ;'2'=BASIC
        JZ      BASJMP
        CPI     51      ;'3'=CRT
        JZ      CRTJMP
;---------------------------------------
        JMP     OPTION

FCSJMP: LXI     H,EXSTR ;exit FCS
        CALL    OSTR
        LHLD    FCSSP   ;restore stack
        SPHL
        MVI     H,044H
        JMP     ESC1

PRINT:  CALL    FSO     ;check file open
        CALL    BLLN
        LXI     H,DSP5A
        CALL    OSTR
;---------------------------------------
        CALL    GTCHA   ;get baud number
        SUI     030H    ;convert to hex
        LXI     H,BAUDTB;point to table
        CALL    ADHLA   ;index pointer
        MOV     A,M     ;move code to A
        OUT     5       ;put it in baud
                        ; rate generator
;---------------------------------------
        CALL    BLLN
        LXI     H,DSP5B
        CALL    OSTR
;---------------------------------------
        CALL    GTCHA   ;prepare routine
;If you want to send setup string, do it
; here with LXI H,SETPR & S1OUT. See text
;---------------------------------------
        CALL    BLLN
        LXI     H,DSP5C
        CALL    OSTR
;---------------------------------------
        MVI     C,64    ;set chr/line ctr
        MVI     E,0AH   ;send cr & lf to
        CALL    S1OUT   ; set printer hd
        MVI     B,128   ;set total chars
        LXI     H,INBUF ;point to text
XX6:    MOV     E,M     ;move in char &
        CALL    S1OUT   ; print it
        DCR     C       ;less 1 per line
        MOV     A,C
        CPI     0       ;line full?
        CZ      XX8     ;yes! index paper
        DCR     B       ;no,text finished?
        JZ      XX7     ;yes! finish up
        INX     H       ;no, point to next
        JMP     XX6     ; char and get it

XX7:    MVI     E,0DH   ;carriage return
        CALL    S1OUT
        MVI     E,0AH   ;line feed
        CALL    S1OUT
        MVI     C,64    ;for next line
        RET             ;go get it!

;---------------------------------------
;Add to string storage -

BAUDTB: DB      00H,01H,02H,04H,08H
        DB      010H,020H,040H
```

Author! Author!

Keep them article submissions acomin', folks. You can't imagine what it means to us to have the postman bring us some material for **Colorcue.** We encourage you to put your work on disk, if you can, and if you wish. We can handle Compucolor disks and also CP/M 8" (single or double sided, single or double density). Plain ASCII text files are just fine. Wordstar files are OK, too. CompuWriter files might present a little problem, but if that's all you've got, send them in and let us worry about it. We'll return your disk, of course. (Although sometimes there is an embarrassing delay in doing so.)

Two Handy Disk Utilities

by Tom Napier
12 Birch Street
Monsey, NY 10952

I have two utility programs that have gotten me out of more tight spots than I care to remember. Since both are quite short, the printed page rather than the disk seems to be the medium for sharing them.

The first enables me to display the contents of any block on the disk, edit it, and rerecord the block on the disk. I have found this to be invaluable whenever a disk read error occurs, since, after loading the program, one can keep on reading the faulty block until an error-free version turns up. Rewriting the block will cure the read error in most cases that aren't caused by physical damage to the disk. (Most EDCS errors - ed.)

I also use the program to locate files when a file erasure has aborted, and the directory no longer correctly indicates the disk content. Its other major use is for minor editing of text or source code. If one has typed a colon in place of a semicolon, it is much quicker to edit a long source file by editing the block directly than to reload the editor, read the file, correct, and rewrite it. It also saves disk space, since the entire file is not rewritten.

The program (Disk Block Examiner and Editor, **Listing 1**) works as follows. It asks if you want the display to be "ALL HEX?". Type "Y" and it will display every byte in the block in hexadecimal notation. Type "N" and the output format will show all printable ASCII characters; a period before a character indicates that it is lower case, unprintable characters are displayed in hex form.

Next the program asks for the block number. Enter this in hex form as it appears in the directory. After you type a valid block number, the program reads and displays the block. You then have the option of typing another block number, rewriting the block displayed (type "W"), or editing the block (type "E"). Beware! No trapping is present to stop the user from typing "W" before any block has been displayed.

On entering the Edit mode, you have the option of changing the character at the current cursor position (two red carets under a byte in the block). Typing any printable character other than RETURN will cause the editor to replace the character over the cursor, and advance it one position. Typing two characters and then RETURN will cause the two characters to be taken as a hex byte, and this will replace the indicated original. You can move the cursor by typing "MM", and answering questions for X and Y coordinates of the new cursor location. X runs from 0 to 15, and Y runs from 0 to 7. The columns of the 16 by 8 byte array are labelled in hex, but the question should be answered in decimal form. Y counts down from the top (0) row. After entering coordinates, the cursor is placed under the desired location, and the program returns to Edit mode. Press RETURN to return to the block number mode, and "W" to rewrite the modified block to disk.

A practical note - this program uses 197 bytes of RAM, a 128 byte buffer, a 1 byte flag, and a 68 byte machine code subroutine. In the program as printed, these occupy locations in RAM between

5F00 and 5FFF. If you don't have RAM available at these addresses, you will have to allocate part of BASIC's RAM area as buffer space. This program uses the BASIC CALL instruction to call a fast decode and display routine.

Variable A points to the CALL routine, B points to the CALL vector, C to a one byte flag location, and D points to the block buffer. A, C, D, and E will need to be changed if RAM between 4000H and 5FFFH is not available. (Or write Tom Devlin for an 8K add-on RAM board - ed.)

How often have you had trouble reading a disk and on checking the directory found things like a start block of FFEF and despaired of ever being able to read the disk again? Have you copied a disk and wanted to change its title? Have you ever wanted to change the start address of a program?

The Directory Edit Program (**Listing 2**) gives you a complete directory editing facility. You can even hide a file by changing the free space location. The program is very easy to use. It asks for a file number (files are numbered consecutively from the top - it's up to you to count them from a DIR command). A file number of zero selects the disk title. After selecting a file number, the screen will display the existing directory entry and request changes, parameter by parameter. To change a value, type the new value in hex form, or just RETURN if there is to be no change. The corrected entry will be written automatically and you then have the option of selecting another entry or returning to BASIC.

```
50 REM  DISK BLOCK EXAMINER AND EDITOR
60 REM  COPYRIGHT 1983  T. M. NAPIER
70 REM  VERSION 6/1/83
100 A= 24064:B= 33282:C= 24570:D= 24320:GOSUB 63000
110 POKE B,195:POKE B+ 1,0:POKE B+ 2,94
120 POKE C,1:REM  HEX FLAG
125 X0= 0:Y0= 0:X= 0:Y= 0:REM  EDITOR CURSORS
130 PLOT 6,2,12,15,10,10,10
135 PRINT "DISK BLOCK EXAMINER":PRINT :PRINT
140 INPUT "ALL HEX? ";Q$
150 IF Q$= "Y"THEN POKE C,0
160 GOSUB 900:INPUT "BLOCK NO., EDIT (E) OR WRITE (W) ";N$:IF N$= "E"THEN 300
165 IF N$= "W"THEN 240
167 IF N$= "B"THEN END
170 PLOT 12,27,4:PRINT "REA "N$" 5F00-5F7F":PLOT 27,27,10,10
180 X= 0:Y= 0:M$= N$
190 GOSUB 800:REM  DISPLAY
230 GOTO 160
240 PLOT 27,4:PRINT "WRI "M$" 5F00-5F7F":PLOT 27,27
250 GOSUB 900:PRINT "BLOCK REWRITTEN"
260 FOR I= 0TO 1000:NEXT
270 GOTO 160
300 REM  BLOCK EDIT ROUTINE
310 GOSUB 900
320 INPUT "NEW BYTE OR MOVE CURSOR (MM) ";Q$
330 IF Q$= "0"THEN 160
340 IF Q$= "MM"THEN GOSUB 900:INPUT "NEW CURSOR (X,Y) ";X,Y:GOTO 550
350 L= LEN (Q$)
360 IF L= 1THEN NB= ASC (Q$):GOTO 500
370 B1= ASC (LEFT$ (Q$,1)):B2= ASC (RIGHT$ (Q$,1))
380 IF B1= 46THEN NB= B2+ 32:GOTO 500
```

```
390 B1= B1- 48:IF B1> 9THEN B1= B1- 7
400 B2= B2- 48:IF B2> 9THEN B2= B2- 7
410 NB= 16* B1+ B2
500 REM  UPDATE BLOCK
510 AD= D+ X+ 16* Y
520 POKE AD,NB
530 X= X+ 1:IF X= 16THEN X= 0:Y= Y+ 1
540 IF Y= 8THEN Y= 0
550 GOSUB 800
580 GOTO 300
800 REM  UPDATE DISPLAY
810 PLOT 3,0,3
820 PRINT " 0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F"
830 PLOT 3,0,5:Q= CALL (0)
840 PLOT 6,1,3,4* X0,2* Y0+ 6,32,32,6,2
850 PLOT 6,1,3,4* X,2* Y+ 6,94,94,6,2
860 X0= X:Y0= Y
890 RETURN
900 PLOT  3,0,22,11:RETURN
1000 REM  BLOCK DISPLAY ROUTINE
1010 DATA 21,00,5F
1020 DATA 3A,FA,5F
1030 DATA A7,7E,CA,3E,5E
1040 DATA FE,80,D2,3E,5E
1050 DATA FE,20,DA,3E,5E
1060 DATA FE,60,3E,20,DA,1E,5E
1070 DATA 3E,2E,CD,92,33
1080 DATA 7E,DA,27,5E
1090 DATA E6,DF,CD,92,33
1100 DATA CD,B3,34,CD,B3,34
1110 DATA 2C,7D,E6,0F,CC,8B,33
1120 DATA 7D,FE,80
1130 DATA C2,03,5E
1140 DATA C9
1150 DATA CD,9B,33
1160 DATA C3,2A,5E
1190 DATA XX
63000 I= 0
63010 READ HX$:IF HX$= "XX"THEN RETURN
63020 C1= ASC (HX$)- 48:IF C1> 9THEN C1= C1- 7
63030 C2= ASC (RIGHT$ (HX$,1))- 48:IF C2> 9THEN C2= C2- 7
63040 POKE A+ I,16* C1+ C2:I= I+ 1:GOTO 63010
```

```
*150 REM  DIRECTORY EDIT PROGRAM 11/10/82
155 REM  COPYRIGHT 1983  T. M. NAPIER
160 CLEAR 200
190 PLOT 12
200 INPUT "SELECT FILE NUMBER ";F
205 PLOT 12,10,10
210 B= INT (F/ 6)
220 N= F- 6* B
230 IF B> 9THEN B= B+ 7
250 B$= CHR$ (B+ 48)
260 PLOT 27,4:PRINT "REA "B$" 5F00-5F7F":PLOT 27,27
265 IF F= 0THEN 1000:REM  EDIT DISK TITLE
270 FF$= "":FT$= ""
```

```
280 S= 21* N+ 24321
290 FOR I= 2TO 7
300 FF$= FF$+ CHR$ (PEEK (S+ I))
310 NEXT
320 FOR I= 8TO 10
330 FT$= FT$+ CHR$ (PEEK (S+ I))
340 NEXT
350 P= 1:GOSUB 1900:AN$= X$
360 P= 11:GOSUB 1900:VN$= X$
380 P= 12:GOSUB 1800:SB$= Y$
400 P= 14:GOSUB 1800:SZ$= Y$
420 P= 16:GOSUB 1900:LB$= X$
440 P= 17:GOSUB 1800:LA$= Y$
460 P= 19:GOSUB 1800:SA$= Y$
490 PRINT :PRINT "ATTRIBUTE       "AN$
500 PRINT :PRINT "FILE NAME       "FF$
510 PRINT :PRINT "FILE TYPE       "FT$
520 PRINT :PRINT "VERSION NO.     "VN$
530 PRINT :PRINT "START BLOCK     "SB$
540 PRINT :PRINT "SIZE            "SZ$
550 PRINT :PRINT "LAST BLOCK      "LB$
560 PRINT :PRINT "LOAD ADDRESS    "LA$
570 PRINT :PRINT "START ADDRESS   "SA$
575 PRINT :PRINT
580 INPUT "CHANGE THIS ENTRY? ";Q$
590 IF Q$= "Y"THEN 620
595 PLOT 28,11
600 INPUT "READ ANOTHER ENTRY? ";Q$:IF Q$< > "N"THEN 190
610 END :REM  **********
620 INPUT "CHANGE ATTRIBUTE ";Q$:IF Q$< > "0"THEN AN$= Q$
630 INPUT "CHANGE NAME? ";Q$:IF Q$< > "0"THEN FF$= Q$
640 INPUT "CHANGE TYPE? ";Q$:IF Q$< > "0"THEN FT$= Q$
650 INPUT "CHANGE VERSION? ";Q$:IF Q$< > "0"THEN VN$= Q$
660 INPUT "CHANGE START BLOCK? ";Q$:IF Q$< > "0"THEN SB$= Q$
670 INPUT "CHANGE SIZE? ";Q$:IF Q$< > "0"THEN SZ$= Q$
680 INPUT "CHANGE LAST BLOCK? ";Q$:IF Q$< > "0"THEN LB$= Q$
690 INPUT "CHANGE LOAD ADDRESS? ";Q$:IF Q$< > "0"THEN LA$= Q$
700 INPUT "CHANGE START ADDRESS? ";Q$:IF Q$< > "0"THEN SA$= Q$
705 FF$= LEFT$ (FF$+ "      ",6)
710 FOR I= 1TO 6
720 POKE S+ I+ 1,ASC (MID$ (FF$,I,1))
730 NEXT
735 FT$= LEFT$ (FT$+ "   ",3)
740 FOR I= 1TO 3
750 POKE S+ I+ 7,ASC (MID$ (FT$,I,1))
760 NEXT
770 P= 11:X$= VN$:GOSUB 1700
780 P= 12:Y$= SB$:GOSUB 1600
790 P= 14:Y$= SZ$:GOSUB 1600
800 P= 16:X$= LB$:GOSUB 1700
810 P= 17:Y$= LA$:GOSUB 1600
820 P= 19:Y$= SA$:GOSUB 1600
830 P= 1:X$= AN$:GOSUB 1700
900 PLOT 27,4:PRINT "WRI "B$" 5F00-5F7F":PLOT 27,27
910 PRINT :PRINT
920 GOTO 600
999 REM  **********
1000 REM  DISK TITLE
1010 DT$= ""
```

```
1020 S= 24321
1030 FOR I= 2TO 11
1040 DT$= DT$+ CHR$ (PEEK (S+ I))
1050 NEXT
1060 PRINT :PRINT "DISK NAME      "DT$
1070 PRINT :PRINT
1080 INPUT "CHANGE DISK NAME? ";0$
1090 IF 0$< > "Y"THEN 595
1100 INPUT "NEW DISK NAME ";0$:IF 0$< > "0"THEN  DT$= 0$
1105 DT$= LEFT$ (DT$+ "          ",10)
1110 FOR I= 1TO 10
1120 POKE S+ I+ 1,ASC (MID$ (DT$,I,1))
1130 NEXT
1140 GOTO 900
1599 REM  ** HEX TO WORD **
1600 X$= RIGHT$ (Y$,2):GOSUB 1700
1610 P= P+ 1
1620 X$= LEFT$ (Y$,2):GOSUB 1700
1630 RETURN
1699 REM  ** HEX TO BYTE **
1700 X1= ASC (LEFT$ (X$,1))- 48
1705 IF X1> 9THEN X1= X1- 7
1710 X2= ASC (RIGHT$ (X$,1))- 48
1715 IF X2> 9THEN X2= X2- 7
1720 POKE S+ P,16* X1+ X2
1730 RETURN
1799 REM  ** WORD TO HEX **
1800 X= PEEK (S+ P):GOSUB 1910
1840 Y$= X$
1850 X= PEEK (S+ P+ 1):GOSUB 1910
1860 Y$= X$+ Y$
1870 RETURN
1899 REM  ** BYTE TO HEX **
1900 X= PEEK (S+ P)
1910 X1= INT (X/ 16)
1920 X2= X- 16* X1
1930 IF X1> 9THEN X1= X1+ 7
1940 IF X2> 9THEN X2= X2+ 7
1950 X$= CHR$ (X1+ 48)+ CHR$ (X2+ 48)
1960 RETURN
```

Cueties

```
10 C(0)= 1
20 C(1)= 7
30 C(2)= 4
40 FOR Y= 0TO 127
50   PLOT 6,C(Y- 3* INT (Y/ 3))
60   PLOT 2,250,0,Y,127,255
70 NEXT
```

Screen Memory – Problems and Cures

by Tom Devlin
3809 Airport Road
Waterford, MI 48095

I have noticed a few references to screen memory problems recently and while these problems are usually simple to fix information on how to diagnose and repair them has been lacking. The following will give a quick overview of the operation of the screen memory as well as examples of some of the more common problems and their solutions.

As most of you know, the screen memory has only 4K of RAM but occupies 8K of memory space. This 8K area runs from 6000-7FFFH and is divided into two halves, 'fast' (6000-6FFFH), and 'slow' (7000-7FFFH). If the 'slow' memory is addressed the CPU waits until the 5027 CRT Controller chip is finished with a character before accessing the screen memory; addressing the 'fast' area lets the CPU gain access at once. This is probably only of academic interest; I mention it only to point out that there is in fact a difference.

The screen memory proper consists of 8 4K by 1 4027 dynamic memory chips (UD4-UD11). Each chip handles 1 bit of the 8 bit character/CCI data word.

The most common problem is random dots of color appearing on the screen. This is usually caused by the 5 volt power supply being a little low. The 4027 memory chips are very fussy about the 5 volt level and setting the supply to exactly 5 volts with a digital voltmeter should clear things up.

A bad memory chip can cause some weird things to happen. Characters can switch colors or change to an entirely different character. You may have blocks of color that will not erase or a character may start (or refuse) to blink. This is more than just annoying, it disables your **DEL**ete, **COP**y and, on V6.78, **DUP**licate functions because these commands use the screen memory as a temporary buffer.

Finding the chip responsible requires nothing more than a knowledge of the ASCII character values and the CCI code. If, for example, a green character (PLOT 6,2) changes to cyan (PLOT 6,6) it must have added blue (PLOT 6,4). Since binary bits add up in 1,2,4,8,16,32,64,128 order within the data word, you know that the third bit has turned on. Which chip controls the third bit? Simple; the chips are mounted on the logic board in the same order as the bits in the data word (UD4 is bit one, UD5 is bit two and so on). A little counting makes it obvious that UD6 is the culprit.

Characters changing can be diagnosed the same way; if a space (ASCII 32) suddenly becomes an exclamation point (ASCII 33) UD4 (bit one) is at fault.

Once you have identified the malfunctioning chip you will have to find a replacement. This isn't easy; the 150ns 4027 memory chips originally used are hard to find and even harder to pay for. I have discovered that the more readily available (and far cheaper due to higher production volumes) 150ns 4116 (16K by 1) will plug right in. The 4116 chips are also not as sensitive to the 5 volt supply as the 4027 part. (At this point some of you are thinking "if we have all that extra memory in the 4116 chips is there any way of using it?". Sure is, assuming that there is any interest (drop me a line) next time I'll show you how to add another page of screen memory.)

Most of the problems due to bad memory

chips will show up as one bit (of the possible eight) being erratic. Another less common problem involves entire rows, columns or sections of the screen going berserk. This is usually caused by one of the three 74S153 address multiplexers UE8, UE9 or UD12. There is no handy one-to-one relationship between these chips and the problems they cause; unless you can read schematics and have access to a scope the best bet would probably be to replace them one at a time until the problem goes away.

UnClassified Advertising

We are offering for sale the following machines:

(1) Intecolor 8063 (I) CP/M, hi-res, dual 250K floppies plus word processing software. Tube & keyboard need work. SN 20384. Original cost: $9200.

(2) Intecolor 8001G with dual floppies, modem port. Fair condition. SN 18974. (Floppies SN 41229.) Original cost $1550.

(3) Intecolor 8900 with floppies and word processing. Mint condition. SN 400644. (Floppies SN 41838.)

(4) 2 NEC printers. Condition unknown.
SN 541005053-7812
SN 541006301-7902

(5) ADDS 580 console. Needs work. SN A25519.

No responsible offer will be refused.

Bonnie Krypel
Inside Sales Coordinator
Apache Electronic Systems, Inc.
900 Jorie Blvd., Suite 124
Oakbrook, Illinois 60521-2211

****

WANTED: Genesis III program. Will buy or trade.

Maurice Adams
3621 Buffalo Rd.
New Albany, NY 14513

****

FOR SALE: Compucolor II V6.78
Two disk drives, deluxe keyboard, Assembler, FORTRAN, Text Editor, Formatter. Cost $3000, will sacrifice for $1500 or best offer.

Andy Mau
5 Eldridge Street, Store North
New York NY 10002 (212) 431-1277

Tech Tip

by John Newman
PO Box 37, Darlington
Western Australia 6070

Since the publication of the article on disk drive improvements in the Feb/Mar issue, a couple of improvements have been suggested. One is to reduce the 1000uF capacitor to 470uF in the motor run-on circuit (Figure 3). The reasons are that most 1000uF caps are too large to fit under the disk drive cover, and it has been found that some drives intermittently misread, causing the usual track 0 reset.

The other suggestions have to do with the speed control switch. One user has fitted a multi-turn pot to the front panel which is connected in circuit only when the switch is set to the low-speed position. The normal speed is left at 300RPM. A related modification is to use a spring loaded switch so that it cannot inadvertently be left in the low-speed position.

Tech Tip

by Alexander V. Pinter
P.O. Box 230
Columbus, GA 31902

The Compucolor Assembler program (#990014) doesn't need the 20 disk blocks allocated to it on the disk. The last five blocks are "DS" type variable spaces that are filled in by the program itself at run time. To make the assembler take up less room on disk, do this:

```
FCS>LOA ASM.PRG 8200
FCS>SAV ASM.PRG 8200 D7D
```

Now the program is smaller by 5 blocks (and will load more quickly, too).

Tid–Bits for Compucolor

by Howard Rosen
Box 434
Huntington Valley, PA 19006

From my experience servicing the CCII I have found an improvement over the Maintenance Manual's procedure for producing sharp color images. Enter CRT mode with the key sequence ESC (CRT), BG ON. The computer is in the CRT mode for changing background color. Press CONTROL (and hold) and Q (for red). Press ERASE PAGE to produce a full red screen. Review the screen for no other color. If required, make adjustment, provided you are experienced. If you do not know how and feel that you wish to try, please contact me for instructions. This step is not for the inexperienced. Repeat GREEN, YELLOW, BLUE, MAGENTA, CYAN and WHITE backgrounds. Once satisfied that each screen was only one bright color, then continue. Set background to BLACK and then press FG ON for foreground color. Press CAPS LOCK key to put the computer in graphics characters. Set color to GREEN and press the following sequence of keys:

ESC Y G

You will see green horizontal lines. These lines are your reference lines. Green, red and blue are individual colors. The other colors are mixtures of them. Now we are going to review and trim up the colors as best we can. Set the color to yellow. Yellow is red and green, therefore we are going to adjust the red to combine with the green to produce the best yellow possible. First observe the center portion of the screen, approximately a four by four inch square. The controls for moving the red up/down are on the neck of the picture tube. Don't touch anything other than those parts I tell you. For that matter, please think twice if you're not too sure. The red knob on the white housing will move the red up/down. Again, this adjustment is only for the center of the screen. Press the ESC Y N keys for vertical lines. The adjustment is the round dial immediately beneath the red knob we just used. Repeat the above for the following sequence of colors: CYAN, MAGENTA, WHITE. Did you use the blue knob in the above procedures when adjusting the blue color? Now that the center of the screen is pretty well adjusted, we are concerned with the edges. Please refer to your maintenance manual for the adjustment of the upper/lower/right/left edges. The main point I wanted to get across was the change described above to get the job done better. A BASIC program to test the colors all over the screen is given in Listing 1. I call it "COLOR". ■

Listing 1

```
5 PLOT 12,15,6,3
10 FOR I = 1 TO 1E6
15 FOR K = 1 TO 7
20 FOR J = 0 TO 63
30 PLOT 3,J,M,6,K,K + 48
35 PLOT 6,K
40 NEXT J
42 M = M + 1
43 IF M = 64 THEN M = 0
45 NEXT K
50 NEXT I
```

Blue Sky Dept.

by David B. Suits

Is it possible to enhance the color capabilities of the Compucolor II? Not being very far advanced in analog electronics, I will allow myself to speculate on matters I know little about. The result will be, I hope, interesting entertainment, even if more knowledgeable readers find these ideas impractical.

Consider: The CRT color is maintained by illuminating zero, one, two or all three of three different colored dots on the screen. We specify which dots will be intensified by controlling three electron guns. Figure 1 shows a simplified diagram. If a magenta dot is showing on the screen, that means that, at a particular location on the screen, the red and blue dots are being intensified, which means that there are signals travelling to both the red and blue guns. If you were to reach inside the computer and rip out the blue line, all magenta colors would instantly change to red. (And any other color which incorporated blue would change: blue to black, cyan to green, and white to yellow.) But it is inelegant to change screen colors by ripping out wires. A more sophisticated procedure would be to place a switch in each of the lines. Now, if those switches could be activated under software control, we would have a nifty new addition to the Compucolor's color graphics capabilities. (Some such on-off control is used, by the way, in some Intecolor computers.) Specifically, it would provide for fast (practically instantaneous) changes of some colors or color combinations. You could, under software control, turn off the red gun. Then anything you drew in red would not show on the screen. Turn the red gun on, and--poof!--there it would be.

Let's extend the possibilities. Some color CRTs (such as your TV set) allow for more than eight colors. But how do we get more than eight colors if we have only three control lines? Easy. Open up your computer and you will find some trimpots, or variable resistors, one for each color. These determine the intensity of each color. Turning up the red pot, for example, will brighten the red and consequently slightly change those colors in which red plays a part: yellow, magenta and white. All we need, then, is to put these trimpots under software control. We'll still be able to display a maximum of eight colors at one time, but these may be any eight colors we wish.

My idea is to build two parallel output ports which will provide 16 control bits in all: 1 bit each for the on-off control for each of the three guns; 4 bits each for controlling the intensity of each gun; and 1 unused bit. (See

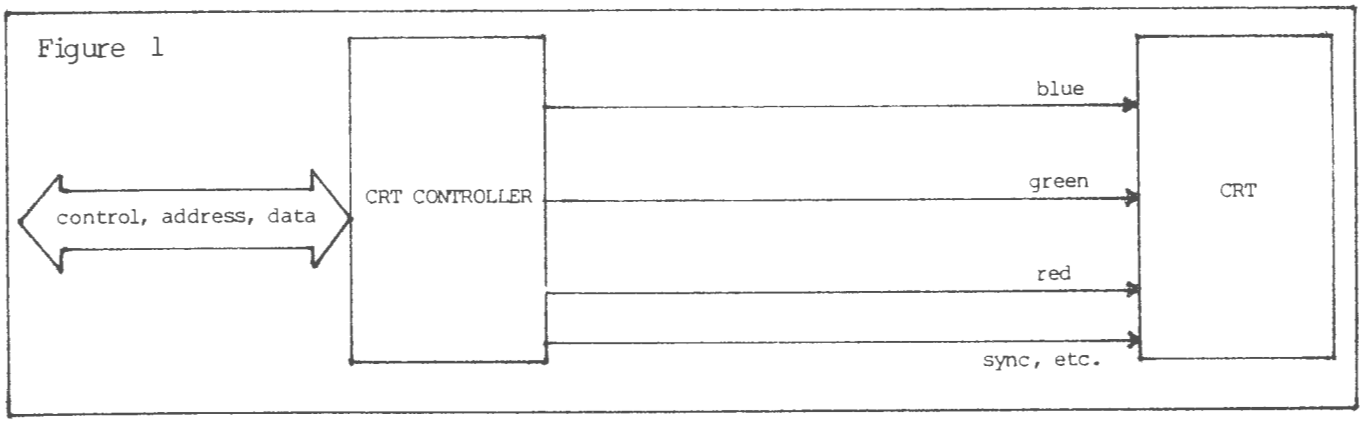


Figure 2.) This gives us 16 intensity levels for each primary color. That's a choice of eight colors out of a palette of 4096!

Will it be easy to implement? Well, the on-off control for each primary color might be arranged as an AND gate just beyond the CRT controller chip. (Figure 3.) The control of the intensity is beyond my abilities, but I wouldn't be surprised to learn that there is some simple chip that could select the 16 levels. On the other hand, the voltage levels there might be too high to be easily dealt with. Or perhaps the scheme suffers from some more fundamental flaw.

There's your challenge for today: either built it, or else tell the rest of us why it's impractical.

Figure 2

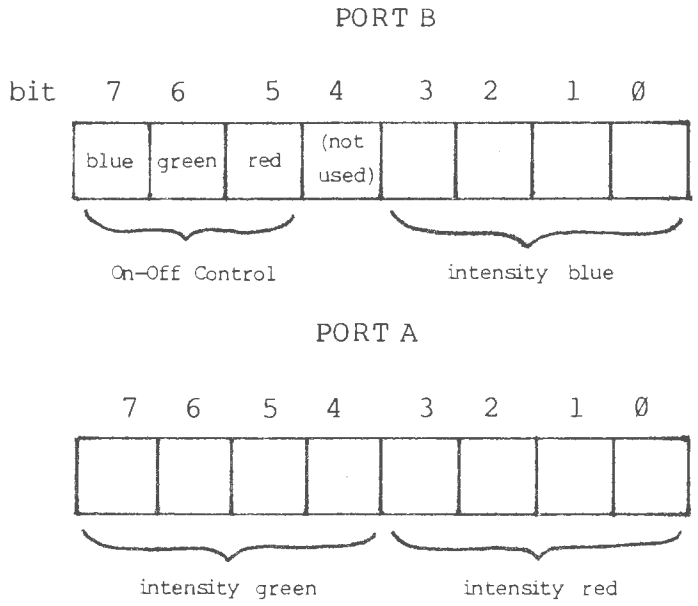
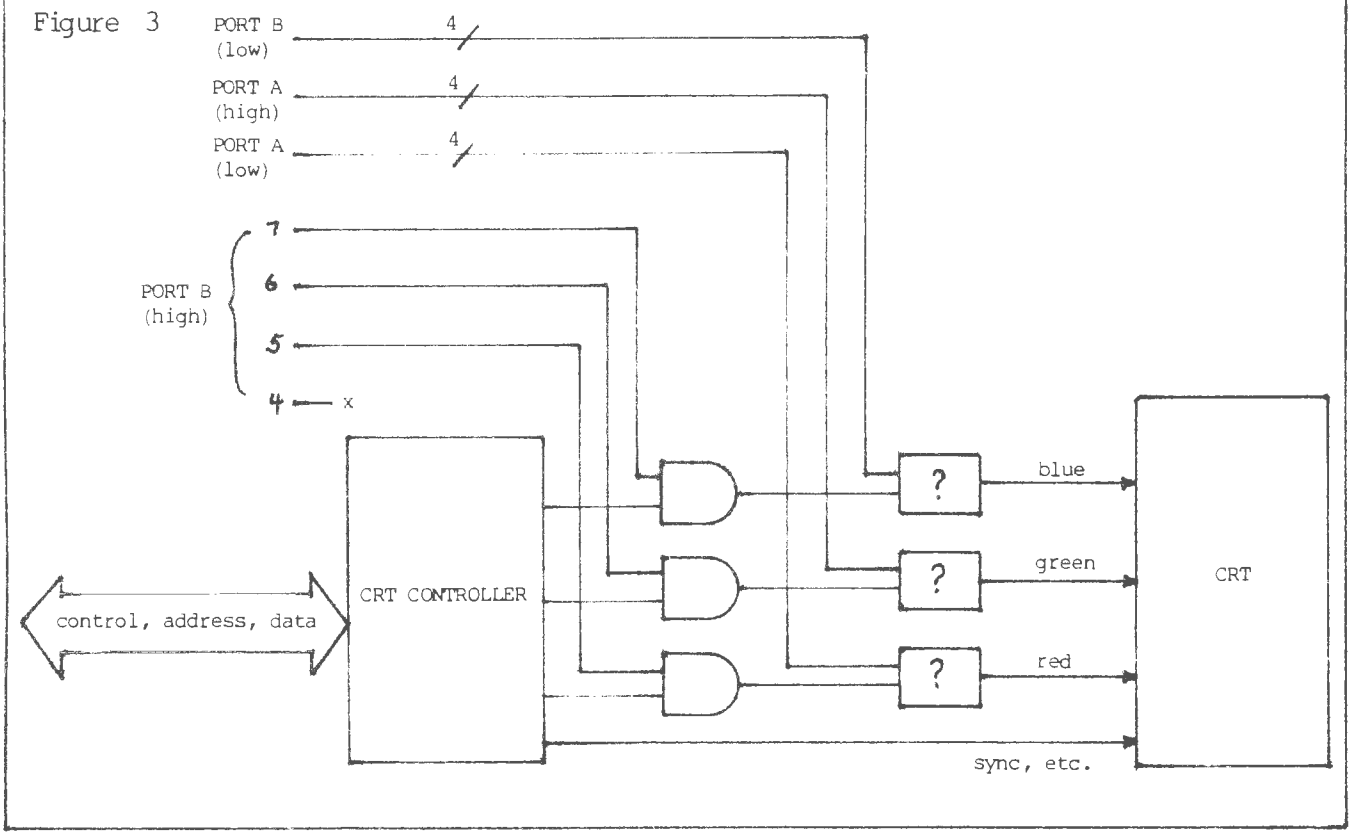


Figure 3



Bar Cursor

by F M Good
(Reprinted by permission from CUWEST, Aug., 1982)

The BAR program is a machine language program which allows input to a BASIC program to be done via a coloured bar which the user moves up or down across a list of selections printed out by the BASIC program. When the user has positioned the bar on the selection he chooses, he presses the RETURN key and program execution returns to the BASIC program.

The subroutine is accessed by the function:

Y = CALL(X)

where X is the top left hand memory position of the list, and Y is the number of the chosen selection.

Before executing the Y = CALL(X) instruction, the program must POKE into the third byte of the subroutine the number of selections in the list. The printed list must follow the following format:

(1) For best results, it should be printed in white.
(2) There must be four spaces in front of each selection in order to allow for the arrow.

```
            ORG     0D000H

START:      DI
            MVI     B,20    ;No. of selections in list.
            DCR     B       ;B=counter for maximum no.
                            ;of movements down.
            SHLD    REGTMP1 ;Store HL.
            XCHG
            SHLD    TOPPOS  ;Store D - position of the
                            ;top selection.
            XCHG
            XRA     A
PUTBAR:     MOV     H,D     ;Get position of bar
            MOV     L,E     ;from DE into HL.
            MVI     M,'-'   ;Put in red arrow.
            INX     H
            MVI     M,1
            INX     H
            MVI     M,'-'
            INX     H
            MVI     M,1
            INX     H
            MVI     M,'>'
            INX     H
            MVI     M,1
            INX     H
            INX     H
            INX     H
            STA     ACCTMP  ;Temporarily store A
                            ;A=no. of current selection.

LOOPPUTBAR1:MVI     C,3     ;Put in cyan background (bar).
LOOPPUTBAR: INX     H
            MOV     A,M
            ORI     00100000B       ;Set background to cyan.
```

```
                MOV     M,A
                INX     H
                MOV     A,M
                CPI     32
                JNZ     LOOPPUTBAR1
                DCR     C       ;Continue bar until 3 spaces in
                                ;a row.
                JNZ     LOOPPUTBAR

LOOPGETKEY:     LDA     KEYFLAG ;Put KEYFLAG
                MOV     C,A     ;in reg. C.
                LXI     H,5000
LOOPSLOWGETKEY: DCX     H
                MOV     A,H
                ANA     A
                JNZ     LOOPSLOWGETKEY

                **  CHECK FOR KEYS  **

                MVI     A,2
                OUT     7       ;Check for return
                IN      1       ;by direct accessing
                CPI     0EFH    ;of keyboard.
                JZ      GOTSELECTION    ;Yes, return pressed.

                MVI     A,3
                OUT     7       ;Check for up arrow.
                IN      1
                CPI     0DFH
                JZ      UP      ;Yes, up pressed.

                MVI     A,5
                OUT     7       ;Check for down arrow.
                IN      1
                CPI     0EFH
                JZ      DOWN    ;Yes, down pressed.

                MVI     A,7
                OUT     7       ;Check for home.
                IN      1
                CPI     0EFH
                JZ      HOME    ;Yes, HOME pressed.

                XRA     A       ;Key not pressed,
                STA     KEYFLAG ;so clear KEYFLAG
                MOV     C,A     ;and reg C.
                JMP     LOOPGETKEY
UP:             CALL    PAUSE
                MVI     C,1     ;Set C to 1.
                MOV     A,C     ;I.e. key pressed.
                STA     KEYFLAG ;Store at KEYFLAG.
                LDA     ACCTMP  ;Restore bar position.
                ANA     A       ;At top of list?
                JZ      LOOPGETKEY      ;Yes, then don't move bar.

                CALL    CEARBAR ;Clear current bar.
                LDA     ACCTMP  ;Restore A.
                DCR     A       ;Move bar up.
                STA     ACCTMP  ;Store new pos.
                LXI     H,-128
                DAD     D       ;Get new bar position.
                XCHG            ;Put new pos in DE.
                JMP     PUTBAR  ;Put new bar.

DOWN:           CALL    PAUSE
                MVI     C,1     ;Set C to one.
                MOV     A,C     ;I.e. key pressed.
                STA     KEYFLAG ;Store at KEYFLAG.
                LDA     ACCTMP  ;Restore bar position.
                CMP     B       ;At bottom of list?
                JZ      LOOPGETKEY      ;Yes, then don't move bar.

                CALL    CLEARBAR        ;Clear current bar.
                LDA     ACCTMP  ;Restore A.
                INR     A       ;Move bar down.
                STA     ACCTMP  ;Store new pos.
                LXI     H,128
                DAD     D       ;Set new pos.
                XCHG            ;Put new bar pos in DE.
```

```
                JMP     PUTBAR  ;Put new bar.

HOME:           XRA     A       ;Clear KEYFLAG.
                STA     KEYFLAG
                CALL    CLEARBAR        ;Clear current bar.
                LHLD    TOPPOS  ;Restore top bar position.
                XCHG            ;Put into DE.
                XRA     A       ;Bar position = 0.
                JMP     PUTBAR  ;Put new bar.

GOTSELECTION:   LDA     ACCTMP  ;Get position of bar.
                INR     A       ;Convert to selection no.
                MOV     E,A     ;Put value into DE
                MVI     D,0     ;for return to BASIC.
                LHLD    REGTMP1 ;Restore HL.
                EI              ;Enable interrupts.
                RET             ;Return to BASIC program.

                ;This routine checks the value of KEYFLAG
                ;and depending on its value either waits
                ;for a short while or returns to the program
                ;immediately.
                ;  This allows the user to either step slowly
                ;down the list by pressing and releasing the
                ;key or move rapidly down the list by keeping
                ;the key pressed.
                ;
                ;  KEYFLAG indicates whether the key is kept
                ;depressed or not ;
                ;       if KEYFLAG=0 then not depressed.
                ;       if KEYFLAG=1 then depressed.
                ;
                ;  Thus us KEYFLAG=1 then routine will not
                ;execute a pause and will return to the
                ;program immediately and if KEYFLAG=0 the
                ;routine will pause before returning.

PAUSE:          LDA     KEYFLAG ;Get value of KEYFLAG.
                ANA     A       ;Is it zero?
                RNZ             ;No, return without pause.

        ;Pause while HL counts down to zero.

                LXI     H,20000
LOOPPAUSE:      DCX     H
                MOV     A,H
                ANA     A
                JNZ     LOOPPAUSE       ;Not zero, decrement again.
                RET             ;Pause ended, return to program.

CLEARBAR:       MOV     H,D     ;Get bar position from
                MOV     L,E     ;DE into HL.
                MVI     M,32    ;Clear arrow.
                INX     H
                INX     H
                MVI     M,32
                INX     H
                INX     H
                MVI     M,32
                INX     H
                INX     H
                INX     H
                INX     H

LOOPCLEARBAR1:  MVI     C,3     ;Clear bar.
LOOPCLEARBAR:   INX     H
                MOV     A,M
                ANI     00000111B       ;Set background to black.
                MOV     M,A
                INX     H
                MOV     A,M
                CPI     32
                JNZ     LOOPCLEARBAR1
                DCR     C       ;Continue to clear bar
                JNZ     LOOPCLEARBAR    ;until 3 spaces in a row.
                RET             ;Finished clearing.

REGTMP1:        DS      2       ;Temporary storage for HL.
TOPPOS:         DS      2       ;Position of top selection.
```

```
ACCTMP:         DS      1       ;Temporary storage for A.
KEYFLAG:        DS      1       ;Flag indicating whether key
                                ;is kept pressed (=1) or key
                                ;pressed then released (=0).
                END     START
```

The Freepost 64K Bank Board:

a review

by Christopher J. Zerr
14741 N.E. 31st Unit 1-C
Bellevue, WA 98007

ITEM: Freepost 64K Bank Board (kit form)
SUPPLIER: Freepost Computer Systems
431 East 20th Street 10D
New York, NY 10010
Price: $199.00

Most CCII owners are presently not utilizing their full memory capacity. ISC left an 8K address space in memory (4000H -- 5FFFH) for use with EPROM (Erasable Programmable Read Only Memory) chips as desired by the end user. First we saw Tom Devlin's RAM board for this area which gives you an extra 8K of RAM. Shortly thereafter we had the announcement of a multi-bank ERPOM board from Freepost Computers in New York. This board can be purchased in two ways: assembled and tested for $249.00 or in kit form for $199.00. I purchased the kit form, so this review will pertain to that.

Assembly of the board is quite simple. All you need is a pencil type soldering iron, some solder, and about three hours. First you solder all the sockets into their respective holes. Be alert, though, because sockets are not supplied for UF6 (the 7447 BCD decoder/driver). The parts I got were confusing because I got a 16 pin plug and two 16 pin sockets, and there just happen to be three 16 pin areas for sockets, and the plug is not one of them. The plug is to be used in case you install the Devlin RAM board: the plug is supposed to replace the IC in UF8. But why Freepost sent a 16 pin plug for a 14 pin socket I don't know.

Once the bank board is assembled, you may install the Devlin 8K RAM board if you have one, or you may use another ROM board. I am using the Devlin board, which requires eight wires to be connected to the logic board. Two come from the RAM board to their places as specified in Devlin's documentation. The other six come from the bank board to the logic board. With only eight wires to connect, I saw no reason to purchase Freepost's optional card edge connector ($10). There are places on the logic board for soldering all of the wires except one, which I soldered directly to the 50 pin bus edge.

There were a few problems. First, the component layout drawing is incorrect. (Some ICs were mislabelled.) Second, it wasn't clear how to set one of the RAM card jumpers. One connection is from jumper J on the card to a spot on the logic board. But nothing was said about the pre-wired jumper on the ram card from J to K. When I called Freepost about this, they said to remove to jumper.

Third, the 7447 IC (UF6) is wired incorrectly. If you use a common anode for the LED (by the way, this option was not documented very well), then pins 3, 4 and 5 of the 7447 must be connected to +5 volts. I pulled the three pins out of the socket and wire-wrapped them together and then attached them to +5 volts. Also, pins 6 and 8 must be pulled to ground. (This option, by the way, shows you which bank has been selected.)

After all this I was ready to power up the CCII. I turned it on and...on it came with no problems. I typed "OUT 255,7" to set up the RAM card and then ran a memory test. Everything worked just fine.

Upon power up, the bank board is set to some random bank. To correct this, pull pin 1 of the 74161 (UF2) out of the socket and run a jumper from pin 26 of the keyboard (CPU reset) to the pin you just pulled out. Now whenever you power up or hit CPU reset, it will automatically place you in Bank 0.

In all, the small trouble and confusion was worth saving the $50 for the kit form. If you have at least a little electronics knowledge then you should try the kit.

REMINDER

Beginning with Volume 6 (Aug/Sept), the subscription rate for **COLORCUE** will be US$18 in North America and US$30 elsewhere.

MORE DISK STORAGE FOR $24.95!

Store 50% more ASCII data.

Works on both V6.78 and V8.79.

Supplied for 8200H and 4000H.

Uses CD0: and/or CD1:

PACK.PRG packs with Huffman Codes, UNPACK.PRG restores.

All ASCII codes accommodated, use for ASM.SRC, Text Editor, CTE files, etc.

Delay for personal checks, Send Postal Money Order for same day shipment of program disk and user instructions to:

VANCE PINTER
P.O. BOX 20
COLUMBUS, GEORGIA 31902

THE FINAL FRONTIER

YOUR MISSION: Rid Federation Space of the Klingons.
YOUR ENEMY: The Klingon Deep Space Fleet, allied forces from the Romulan Empire, and time.
YOUR ALLIES: Only 1 to 3 Starbases (Don't let the Romulans destroy them).
YOUR SHIP: The mightest ship in space...The U.S.S. Enterprise with Phasers, Photon Torpedoes, Regenerative warp power, limited impulse power, and more. You must allocate energy to these systems as you see fit.

This innovative implementation of a classic computer game is complete with many new features and graphics. AND there is more to come as adventure scenarios to link with this program are on the drawing board.

Over 7900 repeatable games are possible or have the computer generate a random game. Variable galaxy size allows you to play from a few minutes to a few hours with "SAVE GAME" feature in case you can't finish at one sitting.

$25.00 including disk and detailed instructions from:
Rick Taubold
197 Hollybrook Road
Rochester, New York 14623

BULK RATE
U.S. POSTAGE
PAID
Rochester, N. Y
Permit No. 415

Colorcue
161 Brookside Dr.
Rochester, NY 14618

Colorcue

June/July 1983

$2

Colorcue

A bi-monthly publication by and for Intecolor and Compucolor Users

Editors:
Ben Barlow
David B. Suits
Compuserve: 70045,1062

June/July, 1983
Volume 5, Number 6

Editors' Notes

A Changing of the Guard

We have some good news and some bad news. The bad news is that our editorship of **Colorcue** comes to an end with this issue. It is a decision we have been thinking about for a while, and it was not an easy one to make. We have been putting **Colorcue** together for two years; and what a fine two years it has been! We have come to know many of you through your letters, articles and phone calls, and we hope to have generated at least a little bit of communication among Compucolor and Intecolor users.

The good news is that one of our most articulate and prolific contributors is to become the next editor: Joseph Norris. Joe has had (and continues to have) an on-going involvement with ISC equipment. He is very much concerned to keep **Colorcue** alive for another twelve months and, in the process, to help encourage Compucolor and Intecolor users to take note of the resources generally left available to them. He has some good ideas for the coming year's issues, so hang on to your hats. The new subscription rate (US$18 in the US, Canada and Mexico, US$30 elsewhere) will remain in effect; the publication schedule of every other month will be kept; and the guarantee that you'll get your money's worth or your money back will of course be honored.

Back issues of **Colorcue** will still be available (see the inside of the back cover for details) from our Rochester address. Requests for back issues of the issues under Norris' editorship, as well as all editorial and subscription correspondence, should now go to:

Joseph Norris
19 West Second Street
Moorestown, NJ 08057.

Although we are leaving as editors, we hope that we will not be leaving as potential contributors. There is still much to be said about these machines we've been pounding on for four or five years, and we will probably have more to say. By the way, **your** articles and contributions are needed just as much as ever. Don't be lazy. -- BCB and DBS

FASBAS Revealed

Speaking of FASBAS, this issue of **Colorcue** brings to you the first of a series on the FASBAS BASIC compiler by no other than the author himself, Peter Hiner. Even if you're not expert in assembly language, and even if you're not really the sort to go mucking about inside a compiler, you will probably learn a lot by reading his articles. They tell you a great deal about regular, interpreted BASIC along the way.

ISC News

Knox Pannill III, well known to readers of the Compucolor software manual, has been appointed to the position of manager of graphics systems support in Atlanta. Intecolor Corp. (now a subsidiary company of Intelligent Sytems Co.) announced the 2427D color graphics terminal incorporating a 16 bit Zilog Z8002 and an 8085, compatible with Tektronix' 4010, 4014, and 4027, and with DEC's VT100/VT52 terminals. The terminal features a readable bit map, 560H X 288V resolution, a 64 color palette with 8 colors concurrently displayable, and a $1,995 price (100 quantity). Intecolor is also offering its VHR-19 terminal for $3,995, which features a 19 inch high resolution screen (1024 X 768 displayed out of 1024 X 1024), high level graphics with ploygon fill, and 8 displayable colors selectable from a palette of 4,096. An NEC 7220 graphics processor, a seperate alphanumeric processor, and a Z80 team up to drive the machine. (Note: before rushing out to buy one, check out the NEC APC which offers the same graphics, and more. -- BCB)

Our congratulations go to Tom Devlin, the first reader in memory to mention the **Colorcue** covers! At our editorial conferences, we spend a long time discussing covers, and have thought at times of going to a standard format "table of contents" type cover. It' nice to know that someone likes them the way they are.

Big Money in Advertising

by Ben Barlow

If you're anything at all like me, you've hefted a recent copy of Byte magazine with all that advertising and wondered just how much money they take in. I've also wondered how many pages of ads they have (never wondered enough to count them, though). The other day I came across some information in a trade publication called Computer & Electronics Marketing that helped answer those burning questions. The summary is displayed for you in the table below. Just for fun, I added a line for **Colorcue** to see how we stack up against the "big boys", and got surprised - advertising in **Colorcue** costs more per reader than any of the less popular magazines, even Byte! For a moment, I felt that our advertisers would feel cheated. After all, our rate per reader is more than three times Byte's! But after reflection, I convinced myself that **Colorcue** is really a bargain for advertisers. Almost anyone in the country can afford at least a quarter page, and our readers represent a specialty market. Although Byte might reach 304,230 people per month, only a small fraction of those represent a market for any particular product. IBM PC owners won't buy software from an Apple ad; Apple people aren't interested in Z80 disassemblers. 100% of our readers, though, are a market for our advertisers. So considered on the basis of cost per applicable subscriber, **Colorcue** comes out on top.

Magazine	Circulation (monthly)	Ad pages per yr	Ad cost per page	Ad ratio ads/eds	Ad cost per subscriber	Revenue from ads
Compute	1,957,670	1,494	3,895	1.1	.00	5,819,130
Personal Computing	525,000	1,857	9,950	1.2	.02	18,477,150
Popular Computing	306,231	809	4,900	1.5	.02	3,964,100
Byte	304,230	4,001	5,950	1.5	.02	23,805,950
PC World	170,000	2,400	3,800	1.1	.02	9,120,000
PC	110,000	4,200	3,226	1.5	.03	13,549,200
80 Micro	108,160	N/A	2,395	N/A	.02	N/A
Creative Computing	108,007	2,082	4,995	1.1	.05	10,399,590
Infoworld	58,913	2,237	3,780	1.0	.06	8,455,860
Colorcue	401	18	30	.1	.07	540

Ad rates quoted for full page b&w. Eds refers to editorial (non-ad) pages.

Assembly Language Programming

by Joseph Norris
19 West Second Street
Moorestown, NJ 08057

(Note: This article refers to program listings which, for completeness, were printed in the April/May issue of **Color-cue.** In order to devote more space to articles, we are not reprinting them here. If you do not have a copy of the previous issue, we will gladly send a copy of the listing (and a refund of your postage) at your request. - eds.)

PART XII: Opening/Closing the File

As we begin this second of three articles, we know how to parse a data string for a valid file name, and we have placed some parameters in the File Parameter Block (FPB). We are ready to call upon the disk directory and OPEN the disk file.

The OPEN routine

The operating system will open files that exist already on disk, or a new file that we wish to create. There is a separate procedure for opening each of these types. In our program a "new" files is a file with a name unique to the disk in the drive, i.e., we cannot have "FIRST.SRC;01" existing and hope to open "FIRST.SRC;02". [1] To call the proper routine, we have only to enter the correct code number in the first cell of the FPB (FPB1 in our program). A "0" in this slot will cause the routine to look for an existing file name and version (default is ";01"); a "1" will ask it to open a "new" file. I have chosen to use unique file names because, with this feature, SOURCE.PRG can serve as a functioning mini-data base.

Since we eventually want to recall and edit previously created files, our program always looks for an existing file first. Here is the procedure (see OPENA in Listing IV).

We must give the OPEN routine the address of the FPB by placing it in the HL registers, place a "0" in the first byte of the FPB and call OPEN. At this point the OPEN routine will access the file directory and look for the file name we have parsed. The following parameters will follow the routine:

- -The "0" will be replaced in the first **FPB** slot by a code number (usually 06H or 20H);
- -any error code will be in BC (with the carry flag set if error),
- -DE will be unpredictable, and
- -the FPB pointer will reside, still, in HL.
- -If the carry is reset, no errors occurred in opening. It is possible to follow CALL OPEN with CALL EMESS, to print the error message.

In our program we assume, for the moment, that any error is EFNF (File Not Found) and proceed to open a new file at program module CREATE. But let's see what happens if the old file is accepted. Continuing in OPENA, we display the name of the opened file (NMDSP) and then prepare to read it into the disk buffer, which, for us, is the same as INBUF, the text buffer. (This dual usage of buffer space must be approached with care, but is quite secure in our program.) At this point, INBUF is still filled with spaces.

To read the file, we must supply the FPB with two more parameters: (a) the starting address of the text buffer (INBUF) in **FBUF**, and the length of the text buffer in **FXBC**. These may be referenced as FPB+32 and FPB+34. (See Part X of this series.) **FXBC**, the buffer length, must be a multiple of the standard block size of 128 bytes. We now "rewind" the file--that is, we set parameters so the FPB knows which byte is the first byte--by calling RWSEQI. This routine initializes **FBLK**, **FAUX** and **FPTR** by setting them to zero. With the FPB data complete as described up to this point, all RWSEQI needs is the FPB address in HL. This call must follow the call to OPEN. There are no status parameters when RWSEQI is complete. Now all the required data are in the FPB, and INBUF is still cleared with spaces.

We are now ready to transfer the text bytes from the disk file into INBUF. We draw our text box and then call READ, which uses the routine GTBYT to read data from the disk. A simple loop pulls all 128 bytes quickly. GTBYT has the following properties:

> To operate, it needs only the address of FPB in HL. Each accessed byte will appear in the accumulator at the end of the call. BC and DE are unaffected, and the FPB pointer remains in HL. The carry bit will be set if there is an error. If the carry and zero flags are both set, then the prescribed buffer length was filled without reading all the data bytes in the file.

As we "get" each data byte, we also print it with **LO**, in the text box on the screen. In routine READ (Listing IV) the B register is the counter, set to the number of bytes to be read and printed. In the File Parameter Block, the last entry, **FPTR** (pointer), begins with the value 0000H and increments by one each time GTBYT is called. At the end of the last counter decrement, **FPTR** will hold 0080H (=128 bytes). Now we can draw an analogy between BASIC's GET statement in random files, and the GTBYT routine. In the BASIC example: GET 1,3,5;G$[20]--**FPTR** would be loaded with the number of the byte to begin reading on--in this case "5"--and the B register counter would be loaded with 20--the number of bytes to be read. (We have not yet considered a System routine that uses the file-record-byte organization, but it exists.)

At this point we return to the option line for the next instruction. But suppose our file does not yet exist. What happens to create it? Going back to OPENA, we see that, after the instruction line CALL OPEN, a "jump on carry" instruction branches the program to CREATE. This branch will be effective if OPEN generates a file error, such as "file not found" mentioned before. (If a file error occurs, the carry flag is set.) It may be that a different file error has occurred, and, if so, we will have another opportunity to explore it, but our assumption is that the desired file does not exist and must be created on disk.

The CREATE routine

The possibility of a file error suggests that we call RESET, a System routine to reset the heads on our disk drives, as a precaution. We will now call OPEN again, but this time with the "new file" code in the first slot of **FPB1**. If an error again occurs, routine ERROR5 will display it for us; otherwise, we print the file name on the screen for reference, as before, draw the test box, and go back to the beginning of the program to erase the disk buffer for new text (but leaving the FPB data undisturbed!). Routine OPEN, this time, will verify that the file name does not exist, and will place reference parameters in the FPB. At this point, the directory is unchanged, so that, should the program be aborted, no damage to data on the disk will result. However, the FPB data is essential to closing the new file correctly and must not be changed before the closing routine has been called.

The choices in our option line are appropriate either to the program with an opened file or to a program with a closed file. For instance, we cannot print a file that has not been opened, neither can we edit its text. I have installed a byte in memory to hold a "1" if a file is open and a "0" if no file is opened. This slot is labelled FS: (File Status) in the data storage section of the listing. You will notice that several routines install the appropriate status byte in FS.

When any option is selected, the program routines FSO and FSC (File Status Open, File Status Closed) are used to test the FS flag. A jump to a likely error message results if the test fails.

The TEXT routine

The next likely option to be selected, after opening a file, is the entering or editing of text. Option 2 will branch to routine TEXT, which sets the buffer length to 128 bytes and uses INPUT (David Suits's routine) to enter or edit the 128 byte buffer, simultaneously showing changes on the screen. We are altering the disk buffer too, of course, since it is the same as the "text buffer". As long as the maximum buffer length is not exceeded (monitored by MCHAR) the FPB data will remain valid. With the text in its final form, the file may now be written back to disk and closed.

The CLOSEA Routine

We check FSO first to verify that an open file exists. The procedure which follows insures that new text or edited text will be written back into the existing version of the file, or to a new file if the new name was unique to the disk. **INSEQO** uses **RWSEQI** for another initialization, and continues by verifying that there is room on the disk for our file. Since we use only a single block of 128 bytes, we will never have trouble rewriting an edited file. Without this feature, we could never write back to the same .SRC file, if we went beyond 128 bytes, without danger of writing over a subsequent disk file. In the case of a new file, we need only one block to verify a good file write.

The next step is to use PTBYT to write the disk buffer to disk. (See routine WRITE.) It works much the same as GTBYT, moving **FPTR** as before, from 0 to 128. **PTBYT** requires the presence of the byte to be written in the accumulator and the FPB pointer in HL. The B register is our counter for 128 bytes; the DE registers keep track of the current buffer address. The following status will result:

The accumulator will contain the byte written, BC and DE are unchanged, and the HL registers still hold the FPB pointer. Carry flag reset indicates no errors. If both the carry and zero flags are set, the allocated file space was filled before all bytes were written.

With the file written to disk, we must now update the directory--add the new file if appropriate--and free the disk buffer space. **CLSEQO** does all this for us. It needs only the HL pointer to the FPB. All other parameters are already in the FPB for final closing. The status parameters are:

- -The accumulator contents are altered,
- -BC holds the error code, if any,
- -and DE is altered.
- -The carry flag set indicates a closing error, in which case the HL registers will be altered.

At this point we have "closed the loop". Listing IV contains all the necessary modules in their entirety. If portions of any routine from the last issue were previously installed, check them carefully to see that they now coincide with the expanded listing.

A WORD OF CAUTION: Since this program makes use of the disk directory, it would be wise to create files on a clean disk until you are certain the program holds no errors; otherwise the directory may be damaged and other programs on the disk will be inaccessible.

Next Time

In the final article in this series we will examine routines to print the text file to a printer through the RS232 port, and install the ENDIT routines. We will also examine, briefly, a few remaining disk file routines. While you are waiting, it could be a useful experiment to reformulate the 128 byte record into a data base page of an address book. You can print the field headings, one below the other, and apportion the 128 bytes into name, address, telephone, etc. In such an application, rather than printing each byte as it comes from GTBYT, you will want to transfer bytes in their field lengths from INBUF to the screen, field by field. Use OSTR to position the cursor after the field heading. You are invited to write to me if there are questions or comments.

NOTES [1] In other formats, a "new" file can also be one with a previously established file name, and a higher version number. To do this, we would open the file as version ;01, for instance, edit the text as before, but poke 02H into **FVER** before calling CLSEQO.

The Final Frontier – *a Review*

by John R. Bell
8300 Fourth Avenue
North Bergen, NJ 07047

We are all familiar with the Compucolor version of STAR TREK and many have come to know the updated version in the Rochester Users' Group library, but The Final Frontier, a new STAR TREK from Rick Taubold and Bill Goss, is a quantum leap above them both. For starters, when you run the menu on the The Final Frontier disk, you hear the STAR TREK theme song! Those of you without soundware will really miss something. The song drags a bit toward the end. All the notes are there, they just come out slower than you'd expect. An animation spectacular is the next treat. The ENTERPRISE and a Klingon battlecruiser glide onto the screen and exchange phaser fire. The ENTERPRISE is victorious (of course) and the Klingon ship shimmers into nonexistence. The two starships are superbly drawn.

As with most sophisticated games, the commands are numerous and a bit overwhelming at first. But not to worry, there's a HELP feature available and extensive instructions before the game. During the game, the screen lists the various items like energy in the shields, warp drive, and phasers, number of photon torpedoes available, and number of Klingons and Romulans still in the galaxy. Yes, there are Romulans to contend with, and they are nastier than the Klingons, because they don't show up in the Long Range Sensor scans. (As every trekkie knows, they have a cloaking device.)

At the start of the game, you can seed the random number generator with a number from 1 to 9, or have the machine generate a really random one based on both the realtime clock and the amount of time since you last pressed return.

Once having selected a seed, you can then define the size of the galaxy, from 3 (for a 3 x 3 galaxy) to 10 (10 x 10). This, coupled with the methods of seeding the random number generator, gives you a choice of 7,992 repeatable games and an unlimited number of random games. But what really sets this game apart from all others is the graphics. You have a large, functional viewscreen, and you can see the shields "flash" when you take phaser hits from the enemy. When you're in RED ALERT the screen changes color, and a RED ALERT signal flashes. Various items can appear in your viewscreen; stars, Klingons, Romulans, and your Starbase, and they change size depending on their distance from you! Up close, the images of the Klingons and Romulans are quite impressive. I won't tell you what happens when you go into hyperspace - I don't want to spoil the surprise.

The game has some interesting aspects to it. When you're all done, if your rating is high enough, you will be promoted to the rank of Starfleet Admiral. If you quit, you will be apprised of the consequences of your action and will be told what the universe thinks of you. The instructions say "BEWARE OF BLACKHOLES!" and you should believe it. Blackholes are present in this game, and can really alter your game plan. Also, the gravitational field of stars can affect your photon torpedoes. I tried to shoot a torpedo through some stars, and the deflection was such that the torpedo came back and destroyed the ENTERPRISE (with rather spectacular results).

In summary, I would say that this is undoubtedly **the** best STAR TREK program I've seen. Friends of mine with Ataris, Commodores, Apples, NECs, and TRASH-80's would rather play this STAR TREK than any other STAR TREK they've ever gotten their hands on. The programmers set the price at a low $25. so that everyone can afford it, without resorting to piracy. All in all, The Final Frontier is **the** best STAR TREK game available on any machine at any price.

The Final Frontier – *Another Review*

by Bill Barlow

Captain's Log, Stardate 8173.65. I just played The Final Frontier on my Compucolor II. It was fascinating with its STAR TREK introduction and the graphics that went with it. The first time I tried to play I got blown away by Romulans and Klingons. Since I thought it was going to be easy with the experience I've had I thought there was a bug in the program! This is kind of like a hit and wait for your punishment game. You hit the Klingons and wait for them to destroy you. It has good graphics and sound except that the music doesn't have a beat to it, and there's only sound during the intro. The commands are easy to remember. You have "t" for Torpedo and "m" for Move etc... The only thing that I didn't understand was the LONG RANGE SCAN. I didn't know that my ship's position was always in the center of the scanner, so I thought I was where I was supposed to be when really I was almost in front of a Klingon warship! If you understand the scanner you can find your Federation Starbase and get more torpedoes and impulse power. Overall this game is very good, if you like to be challenged! It's hard to win at this one, fellow crew members, so set your phasers to stun and go get 'em! (made for 32K)

Tech Tip

by John Newman
PO Box 37, Darlington
West Australia 6070

Since the publication of the article on disk drive improvements in the Feb/Mar issue, a couple of improvements have been suggested.

One is to reduce the 1000uF capacitor to 470uF in the motor run-on circuit (Fig.3 in the Feb/Mar 83 issue) - most 1000uF caps are too large to fit under the disk drive cover, and we have found that some disk drives intermittently misread, causing the usual track 0 reset.

The other suggestions relate to the speed control switch. One user has fitted a multi-turn pot to the front panel which is only connected in the circuit when the switch is set to the low speed position; the normal speed is left at 300RPM. A related modification is to use a spring loaded switch so that it cannot be inadvertently left in the low-speed position.

Compiling BASIC

by Peter Hiner
11 Penny Croft
Harpenden, Herts
AL5 2PD England

PART I: HISTORY

Dave Suits thought that the chance of some free publicity might induce me to write about my BASIC compiler (FASBAS), how it works and what I learnt during its development. He was right.

Winding the clock back about two years, it all started from a curiosity to find out what was hidden in ROM. I spent a lot of time vainly trying to follow the program step by step from address 0000 and got lost in the mysteries of cold start routines. I decided that a better approach would be to try to identify the purpose of the subroutines without worrying about details, and then see how they fitted together. So I borrowed a printer and a large pile of paper, and generated a complete listing of the disassembled contents of ROM.

After many weeks of labour, helped by some lucky guesses, the jigsaw started to fall into place. I suppose I had expected to find a lot of convenient BASIC subroutines that I could easily use in Assembly Language programs; this was not to be. The first problem with the BASIC routines was that they operate on floating point values, and I could not see how to handle them within the body of an Assembly Language program. Then I got the idea that if I wrote a rudimentary sort of compiler, which would take simple BASIC instructions and generate an Assembly Language SRC file, I could edit the file to add the rest of the program, somehow keeping the floating point routines separate and avoiding the difficult problem of interfacing with them. Although I did not eventually follow this route, the concept of generating an Assembly Language SRC file was retained, and I got started on the compiler.

The first major landmark was when I managed to compile a program like this:

```
10 A = 1
20 PRINT A
```

What a tremendous achievement!

I knew the addresses of the subroutines for COS, SIN, etc., so I was soon able to compile A = COS(B). The arithmetic functions were a bit more tricky, as they involved putting things onto the stack, but once I had mastered these, and included simple IF statements, I had an elementary language for writing BASIC programs and compiling them. The first program of any consequence that I compiled gave a spectacular ten times increase in speed, but I have subsequently realized that this was partly due to the rudimentary (and hence slow) nature of the BASIC program. For example, the statement:

```
100 IF A+B*C = 3 THEN GOSUB 500
```

had to be broken down into:

```
100 X = B * C
101 X = A + X
102 IF X = 3 THEN GOSUB 500
```

I went on to add further essential instructions such as INPUT, PLOT, etc., and at this stage I had achieved my ambition to be able to compile useful programs, even if they had to be written specially to match the limited set of

instructions available. I might have left it alone at this point, but at the insistence of fellow members of the UK User Group (and encouraged in particular by Dave Thomas) I continued towards the goal of being able to compile any BASIC program without imposing restrictions on the way it was written. And when at last I thought I had finished and I released FASBAS VER12.20, I got back not only bug reports, but also requests for removal of the few remaining limitations (and in this case I mention particularly the Australians as hard taskmasters!). With the current version (VER12.21) I hope I have reached a satisfactory level of transparency, and further versions will extend the scope of the program to include V9.80 BASIC, generating code for loading into EPROM, etc.

FASBAS BASICS

Enough of the history. How does the compiler work? First it loads from disk a table of Assembly Language instructions including a library of subroutines and the ROM addresses appropriate to the version of the machine for which the BASIC program is to be compiled. Then it assigns input and output buffer space for the BASIC and Assembly Language files. The library of subroutines is immediately transferred to the output buffer, to become the start of the compiled program. During these operations part of the FASBAS program itself gets overwritten, and that is why it can not be interrupted and restarted without reloading from disk. The library space will be overwritten by lists of variables, etc., but the table of ROM addresses is retained for use in generating the compiled program.

Now the BASIC program is loaded to the input buffer and FASBAS scans through it looking for DATA statements (also for multidimension numerical arrays, but these will be discussed later). If a DATA statement is found, a label Dnnnn: is generated (where nnnn is the BASIC line number), and written to output buffer together with the actual data. Thus all the data is gathered in order and suitably labeled so that the BASIC functions of READ and RESTORE can easily be simulated by the compiled program.

During the second scan through the BASIC program, the main body of the compiled program is written to output buffer. To describe this I will give examples, starting with the simplest.

Each line of BASIC starts with a two byte linking address (pointing to the start of the next line) and then the line number expressed as a two byte (16 bit) binary value. FASBAS skips over the linking address (which is not needed) and generates a label Lnnnn: for the line number.

Suppose that the first line is:

10 GOTO 500

FASBAS generates L10: JMP L500. At this stage we do not know what the memory address for line 500 will be, but it will be labeled (L500:) when we get to it, and the Assembler will insert the actual memory address later. Similarly GOSUB 500 would become CALL L500. To digress for a moment, let us consider the difference in operation between the compiled JMP L500 and the BASIC GOTO 500. The compiled version contains a direct jump instruction which will be executed almost instantaneously. The BASIC interpreter would operate on GOTO 500 as follows: The token representing GOTO would be looked up in a table to find the address of the appropriate routine, and then 500 would be converted from the three ASCII characters '5', '0', '0' to a 16 bit binary value. This value would be compared with the current line number to see whether the search for line 500 could be carried out by scanning forwards from the present position or whether the search had to be started from the beginning of the BASIC program. (It cannot search backwards.) The search is carried out by reading each line number and comparing it with 500. If not equal, then the linking address is used to enable the interpreter to skip directly to the start of the next line, to compare that line number with 500; and so on. Clearly, for this function the compiled program can operate hundreds or even thousands of times faster, depending on the length of the BASIC program which has to be searched through.

Now we will deal with a statement like A=5. FASBAS sees that the line begins with a variable and therefore knows that it has to deal with an equate statement. FASBAS keeps a record of variables used (for assigning memory space at the end), so first it looks through this list to see whether it has already recorded variable A (which will be given the label VA). FASBAS also stores VA in a temporary buffer for use in a moment. Then it skips

over the equal sign and evaluates the rest of the statement (in this case simply 5). Constants are handled very much like variables, and 5 will be called K5, will be recorded in a list, and at the end will be assigned memory space, into which FASBAS will put the 4 byte floating point value for 5.

```
LXI  H,K5     ; Point to location containing value 5.
CALL ....     ; Subroutine to move contents of
              ; identified location to the BASIC
              ; Accumulator.
LXI  H,VA     ; Point to location of variable A.
CALL ....     ; Subroutine to move contents of
              ; BASIC Accumulator to identified
              ; location.
```

Listing 1

The compiled program generated by the statement will be something like Listing 1. The BASIC Accumulator mentioned in the listing refers to a 4 byte store located at 80DEH to 80E1H, which is central to all BASIC mathematical operations. Again, we will digress to consider how the BASIC interpreter would handle the statement A=5, and you will see that it uses the Accumulator in a similar way, but has a lot of additional work to do. When the BASIC interpreter sees a line starting with a variable, it knows it has to deal with an equate statement. First it searches through its store of variables, looking for A, and if it cannot find A, it creates a new variable location for A. Variables are stored at the end of the BASIC program and are followed by storage space for arrays. So if it has to create a new variable location, it must first move all the arrays 6 bytes higher in memory to make space (2 bytes for the name of the variable and 4 bytes to store a floating point value). The interpreter pushes the address of the variable location onto the stack and then checks for the presence of an equals sign (SN ERROR if missing). Now it evaluates the rest of the statement, and since this is the constant 5, it has to convert 5 from ASCII to a floating point value, which it puts in the Accumulator. Then it pops the address of variable A from the stack to registers HL and moves the constants of the Accumulator to the location identified by HL. If the statement to be interpreted were A=B, then the second part of the operation would be like the first (search for variable B, etc., and then move the contents of variable B's location via the Accumulator to the location of variable A).

I hope you can now see why BASIC can often be speeded up by replacing constants with variables which have been used early in the program: the search for a variable location will then be faster than the complicated routine for converting a constant from ASCII to floating point. The compiled version of a program saves times by pointing directly to the location containing the required floating point value, and it does not make any difference whether this is a variable or a constant, or where it is first used in the program.

The mathematical functions such as SQR, LOG, COS, etc. are simple to deal with provided that we do not attempt to delve deep into the subroutines for handling them. If we consider A=COS(B), the first part is as described for A=5 above. To evaluate COS(B), FASBAS notes that it has to generate a statement to call the COS subroutine (it pushes onto the stack a pointer to the address of the COS subroutine in its table of ROM subroutines), and goes on to evaluate the parenthetical expression. The compiled program will follow the principle of putting a value (in this case variable B) into the BASIC Accumulator, calling a subroutine to operate on the contents of the Accumulator, and then expecting to find the result of that operation in the Accumulator. So the compiled version of A=COS(B) would be as in Listing 2.

```
LXI  H,VB     ; Point to location of variable B.
CALL ....     ; Subroutine to move contents of identified
              ; location to BASIC Accumulator.
CALL ....     ; Subroutine to generate COS(B) with
              ; result in Accumulator.
LXI  H,VA     ; Point to location of variable A.
CALL ....     ; Subroutine to move contents of
              ; Accumulator to identified location.
```

Listing 2

When we start looking into an example of simple arithmetic, the same principles for using the Accumulator will apply, but there is an added complication that we now have one more variable (or constant) to handle. For the statement A=B-C, we

know that we have to get the result of evaluating B-C into the Accumulator so that it can be moved to the location for variable A. As FASBAS works through the line, it finds variable B and generates instructions to move it to the Accumulator. Then it finds the minus sign, so it generates an instruction to PUSH the contents of the Accumulator onto the stack. Now it generates instructions to move variable C to the Accumulator, to POP variable B from the stack to registers BC and DE and to call the Subtraction subroutine. Now B-C is in the Accumulator. This principle could easily be extended to handle A=B-C+D, and the result of B-C would now be pushed onto the stack, variable D would be moved to the Accumulator and the Addition subroutine would be called, resulting in B-C+D in the Accumulator. The compiled program for A=B-C+D would look like Listing 3.

```
LXI  H,VB     ; Point to variable B.
CALL ....     ; Move variable B to Accumulator.
CALL ....     ; Push variable B onto stack.
LXI  H,VC     ; Point to variable C.
CALL ....     ; Move variable C to Accumulator.
CALL ....     ; Call subtraction routine (which includes
              ; instructions to pop variable B from stack
              ; to registers BC and DE).
CALL ....     ; Push result B-C onto stack.
LXI  H,VD     ; Point to variable D.
CALL ....     ; Move variable D to accumulator.
CALL ....     ; Call addition subroutine (which
              ; pops B-C from stack to registers).
LXI  H,VA     ; Point to variable A.
CALL ....     ; Move B-C+D from Accumulator to
              ; location of variable A.
```

Listing 3

No doubt you will comment that this is very long-winded. The final compiled version takes up 36 bytes of memory space compared with 7 bytes for BASIC (or 12 bytes if you include line number, linking address and line terminator). I am afraid this is the penalty you have to pay for getting an increase in speed.

The method described for evaluating and compiling an expression like A=B-C+D is fine as long as you are simply working from left to right. I will leave the delights of carrying out arithmetic operations in priority order until next time, when we will also look at some of the more complicated functions.

Meanwhile I should include a note of explanation about the SRC file, which FASBAS generates as an intermediate step towards the final compiled version of a BASIC program. I have described the output of FASBAS as if it were Assembly Language, and originally it was precisely that. But I found that the SRC file was typically 10 times the size of the original BASIC program, so I tokenized many of the instructions to save space. 'JMP' becomes lower case 'j', 'CALL' becomes lower case 'c', etc., but if you do not have a lower case character generator you will see a lot of graphic symbols. Also I omitted all spaces, commas, and carriage returns, and the H character denoting hexadecimal values. All of this serves to disguise what is really genuine Assembly Language, but the special version of Assembler (FBASM) sorts it all out.

NEXT TIME: Expression evaluation and string handling.

Cueties

```
5 PLOT 29:C= 0
10 PLOT 27,24,12
20 FOR Y= 0TO 31STEP 4
25 PLOT 6,C
30 FOR X= 0TO 62STEP 3
40 PLOT 3,X,Y,116,117,32
50 PLOT 3,X,Y+ 1,118,119,32
60 PLOT 3,X,Y+ 2,32,116,117
70 PLOT 3,X,Y+ 3,32,118,119
80 NEXT X:C= C+ 1:NEXT Y
90 GOTO 10
```

Go the Superior Way with your IRA

by Doug Van Putte
18 Cross Bow Drive
Rochester, NY 14624

This article is for you Compucolor/Intecolor users who have invested or plan to invest in an IRA. Is there one of you that would pass up the chance to earn thousands of extra dollars on your IRA investment using a safe and simple method? I think not! My goal is to persuade you that this can happen, and that it will require very little effort or risk on your part. It all begins with an IRA investment, but it goes one important step beyond. In carrying out this step, your computer can be a real profit-making partner.

Last year was a landmark in the evolution of government approved retirement plans. For the first time, individual retirement accounts became available to all workers. Regardless of other plans, you could set-up IRA's for a maximum of $2000 for both yourself and for your working spouse. And millions of Americans did just that. That act allowed them to:

- o Deduct the amount invested from their gross income before taxes - an immediate tax savings. And ..
- o Participate in a myriad of investment vehicles to accummulate tax deferred savings until retirement at a minimum age of 59 1/2 years.

It sounds like the American dream tax shelter, doesn't it? It certainly is, especially when you invest that IRA wisely.

The wise folks have their IRA's in investments that stress high interest or high dividends. Its a simple economic fact that one can profit immensely from the compounding of high yields. The preferred investments include money market funds, utilities, bonds, and high yielding stocks. If you are not investment wise, you can still have access to all these investment vehicles without brokerage fees by participating in a mutual fund. If you are already doing all of this and are wondering where all the extra thousands are coming from - hold onto your socks because I'm going to convince you that you can achieve even greater returns.

I propose to you a method that will allow you to participate in all of the above vehicles and skim off the cream from each one at low risk, without much effort, and all without much investment savvy. Sounds almost too good to be true, doesn't it? No, it's not against the law! Read on further...

Mutual funds not only offer all the investment vehicles I've talked about, but they allow you to switch from one vehicle to another with a toll-free telephone call - in other words, you are your own money manager. You can switch your IRA when the time is right to maximize your yields.

Why does this opportunity arise? Consider Figure I : Interest Rates vs. Stock Values. It is generally accepted that as

stock prices increase, interest rates decrease, and in the opposite manner, as stock prices decrease, interest rates increase. Wouldn't it be great if one could take advantage of both high stock values and high interest rates? Well, it is possible by making use of a strategy originally proposed by R. Fabian [1].

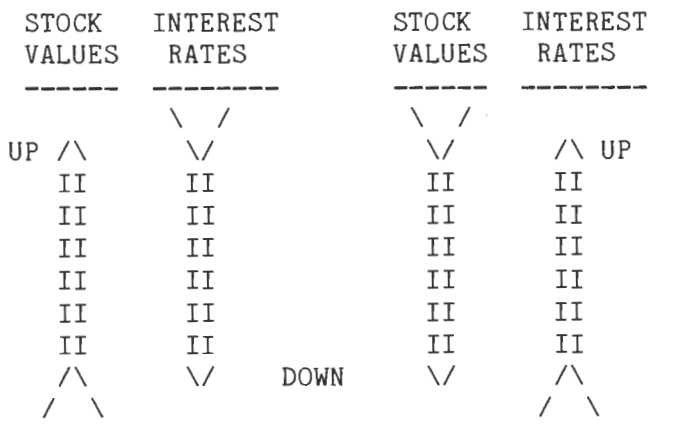


FIGURE I: STOCK VALUES vs. INTEREST RATES

What is the strategy? Consider Figure II, a plot of a typical stock value over a two year period. This plot illustrates the basic strategy in two axioms.

- On a rising stock market, you switch from an interest fund to a stock fund. Suppose your IRA is invested in a money market fund which has a nice interest rate, when the stock fund begins to appreciate in value. As the money market interest rate starts to drop, you'll begin to wish you could get some of the stock fund action. You can get that action by a simple telephone call. The mutual fund will sell your money market shares and invest all the proceeds into the stock fund you specify. Now you can profit from the appreciation in the stock fund shares and also collect the stock fund dividends.

- On a falling stock market, you switch from a stock fund to an interest fund. Sooner or later the stock fund shares will begin to depreciate in value. By making another telephone call, you convert your IRA back to money market shares. Now you can begin to proft from the increasing interest rates in your money market shares. Meanwhile, you are patting yourself on the back for taking that nice capital gain from the sale of your stock fund shares.

I know that you are thinking that it's all too neat. After all, Figure II is plotted after the fact, and everybody tends to be 100% accurate with hindsight. So...

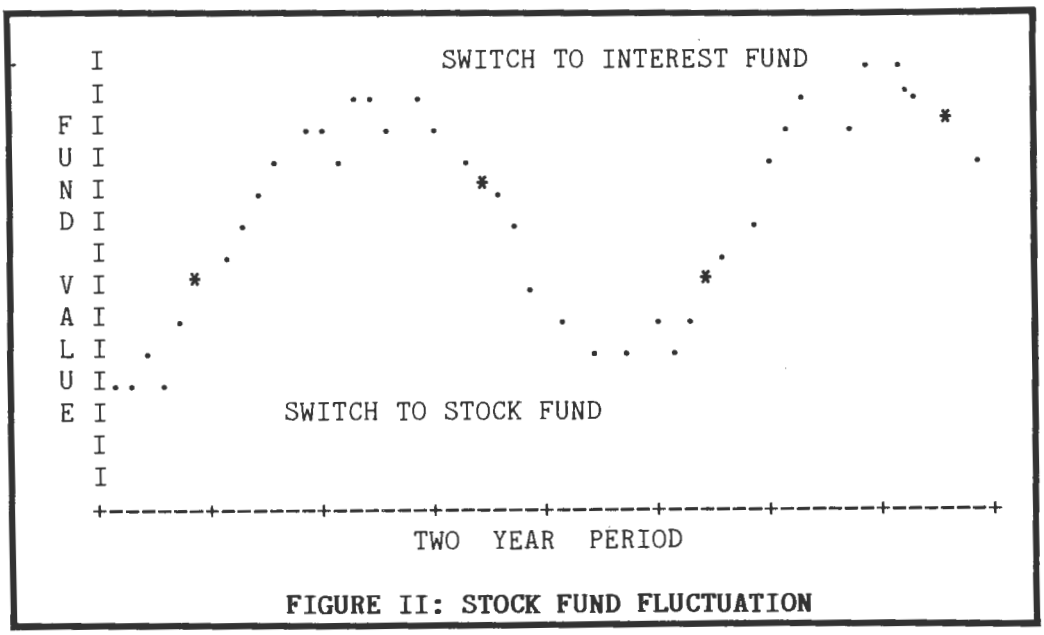


FIGURE II: STOCK FUND FLUCTUATION

Where does one get the switch signal in real time?

o One way relies upon my computer program, STOCK FUND SWITCH STRATEGY PROGRAM, to help you make the switch decision. The program allows one to calculate the direction of the stock shares each week based on data taken from his local newspaper. The data input required by the program are the week-ending values of the stock fund(s), and the Dow Industrial and the Dow Transportation indices. A comparison is then made between the week-ending values and their respective 39-week averages. The individual comparisons are then combined to compute the upward or downward direction of the stock fund shares.. A menu of the program functions is listed on Figure III. Contact me if you'd like more details [2] on the stock program.

STOCK FUND SWITCH STRATEGY PROGRAM

1. CREATE STOCK FILES.
2. ENTER WEEKLY VALUES.
3. PRINT DECISION TABLE.
4. PRINT & EDIT DATA FILES.
5. INPUT MULTIPLE WEEKS OF DATA.
6. PLOT STOCK & DOW FOR ONE YEAR.
7. ADD A NEW STOCK FILE.
8. EXPLANATION OF PROGRAM STRATEGY.
9. STORE DATA AND END PROGRAM.

FIGURE III: SFSSP PROGRAM MENU

o Another way is to subscribe to a financial newsletter [3] that will give you the switch signal.

Both the SFSSP Program ($39.95) and a typical newsletter subscription ($97/year) are tax deductable.

What are the potential gains of the dynamic switching method over the typical static holding method? Consider Figure IV which compares the methods for two funds. The switch method results in about 30% better return for the Nicolas Fund, while it yields a 560% greater return over the holding method for the Pennsylvania Mutual Fund. That is the result of compounding that extra return over a 10 year period! Now you can visualize the source of those extra thousands of dollars for your retirement.

You can manage your own IRA's aggressively, using a simple, relatively risk free method to earn thousands of extra dollars for your retirement. Are you ready to join me now in reaping the extensive benefits that are ours for the taking?

[1] Based on article in PERSONAL FINANCE, Vol. IX, No. 11, entitled "Beat The Market with No-Load Mutual Funds" by R. Fabian, an SEC-registered investment advisor.

[2] Send $1 to cover postage for copies of the SFSSP program documentation and the article in footnote 1:

[3] TELEPHONE SWITCH NEWSLETTER, P.O. Box 2538, Huntington Beach, CA 92647.

	PERCENTAGE GAIN		GROWTH RATE+ (%)	
	BUY AND HOLD	TELE-PHONE SWITCH*	BUY-AND HOLD	TELE-PHONE SWITCH*
NICHOLAS FUND	277	378	14.2	17.0
PENNSYLVANIA MUTUAL FUND	115	528	7.2	20.2

\+ COMPOUND ANNUAL GROWTH RATE.
* ASSUMES 6% YIELD ON MONEY MARKET FUND.

FIGURE IV: SWITCH vs. HOLD STRATEGY COMPARISON FOR 10-YEAR PERIOD ENDING 12/31/81

Transformers (not electrical)

by David B. Suits

Here's an interesting fact about numbers. Given any positive integer (i.e., 1, 2, 3, ...), if it is odd, then multiply it by 3 and add 1. Otherwise, divide it by 2. Repeat the process for the result. Continue this for as long as you please. Curiously, the result you get will eventually be 1. **Any** positive integer will "transform to 1". (I don't know whether anyone has actually been able to prove that yet.) You can test this to your satisfaction using the program in Listing 1.

Actually, this might just as well be called "transform to 4" or "transform to 2", since, if you don't stop at 1, the next step will be 4, and the next will be 2; and then you'll be back at 1, and you'll loop in this fashion forever. Perhaps it should be called "transform to the 4-2-1 loop".

You can enhance the transform to 1 program by printing out a list of numbers along with the number of steps it takes to transform to 1. The number of steps is usually surprisingly small. And there are a few peculiarities: 5 takes 5 steps. That's a nice symmetry. Let me invent a term. I'll call any number which is equal to its transform step count an **isostep.** Is 5 the only isostep? I don't know how to prove whether it is or not; but the program in Listing 2 will help you look around for others. I put variable TN into program ISOSTEP so that you can use 4 or 2 as the terminating number and look around for isosteps in transform to 4 and transform to 2 sequences.

Incidentally, can you think of other transforms which will necessarily lead to a final loop? I don't mean trivial transforms such as "If n is odd, then add 1, else divide by 2." I mean, rather, simple transforms which yield surprising results. And what's more, the transform ought to have the same result for any number. That is, the transform "If n is odd, then add 1, else subtract 1" will of course lead to a simple loop (and right away). But it won't be the same terminating condition for every number. Just to get you started, here is an alternative version of the transform to 1: "If n is odd, then N=3*N-1, else N=N/2." In this case, 1 is subtracted from 3*N instead of being added to 3*N, as it is in the original transform to 1. Will this work? And if so, will there be

Listing 1

```
10 REM Program TRANSFORM TO 1
20 PRINT
30 INPUT "Enter any positive integer: ";N
40 IF (N <> INT(N)) OR (N < 1) THEN 30
50 COUNT = 0
60 IF N = 1 THEN 120
70 PRINT N;
80 IF N/2 <> INT(N/2) THEN N = 3*N+1 : GOTO 100
90 N = N/2
100 COUNT = COUNT+1
110 GOTO 60 : REM Repeat until N = 1.
120 PRINT N : REM It will be 1 when it gets here.
130 PRINT COUNT" steps."
140 GOTO 20
```

Listing 2

```
10 REM Program ISOSTEP
20 TN = 1 : REM Terminating Number.
30 LO = 1 : MAX = 100 : REM Look at numbers LO .. MAX.
40 PRINT
50 FOR J = LO TO MAX
60     N = J
70     REM Transform it to TN and count the steps it takes.
80     COUNT = 0
90     IF N = TN THEN 160
100    IF N/2 <> INT(N/2) THEN N = 3*N+1 : GOTO 120
110    N = N/2
120    COUNT = COUNT+1
130    GOTO 90
140    REM Print the original number and its step count.
150    REM If the number is an isostep, print it in red.
160    IF J = COUNT THEN PLOT 17 : PRINT J,COUNT, : GOTO 180
170    PLOT 18 : PRINT J, : PLOT 23 : PRINT COUNT,
180 NEXT J
```

the same isosteps as in the original transform to 1?

All this is to whet your appetite. There's a lot to investigate here. For example, do prime numbers play any peculiar role in the transform to 1 (or 4 or 2)? Are there numbers whose step count is prime? Are there prime isosteps? Is there a relationship between any two numbers which have the same step count (e.g., 5 and 32)? Is there a relationship between any two isosteps? Is there any way to predict what a number's step count will be, or whether it will be an isostep? And.... Well, you get the idea.

Happy transforms.

UnClassified Advertising

****

FOR SALE: Compucolor II V6.78.
16K RAM. Analog board needs repair.
Sold as is. Money order only, please.
$450.

Roland Lundberg
1506 44th Ave. SW
Seattle, WA 98116

****

FOR SALE: Intecolor 3621 V8.79.
16K, standard keyboard. Includes
Programming and maintenance manuals.
Excellent condition. Asking $800.

Stephen Zehl
19 Courtenay Circle
Pittsford, NY 14534
(716) 586-3787

****

Repairing BASIC Line Numbers

by Mike Barrick
Valley Forge High School
9999 Independence Blvd.
Parma Heights, OH 44130

A short time ago a single bad memory location in my CCII damaged many of my programs. When parts of the program would scroll through the bad memory location, the ASCII value would change by sixteen. For example, the letter "T" would become "D" because the ASCII value would be reduced by sixteen.

After replacing the bad memory IC, I began to repair the programs. This was time consuming, but easy using FREDI. Unfortunately, some line numbers had been changed, and FREDI doesn't edit line numbers. I had repaired a few line numbers in the past using tips from Dennis Martin's article "How to Poke Without Getting Jabbed" in the Dec/Jan 1980 **Colorcue**, but this time there were too many to repair using that method.

I wrote the following program to append to each program to be repaired. The program is begun with a RUN 65000 and requests a START ADDRESS where a scan of memory is to begin looking for a line number. The starting address is usually the beginning of BASIC in RAM, but may be a best guess of the memory location where the line number that is to be repaired is located. Beginning at the starting address, the program scans memory looking for a zero which indicates the end of a line of BASIC coding. It then jumps to the memory location of the next line number which is contained in the third and fourth bytes beyond the location of the zero. The line number is converted from hex to decimal and displayed. If the line number is the one to be repaired, it allows a correction to be made. If there is still quite a span to the desired number, enter a zero and the program returns to START ADDRESS for a better guess at the location of the desired line number. If the second guess brings the user a few line numbers before the desired one, a move to the line number to be repaired can be made by entering the same line number as displayed and the program moves to the next line number. This procedure can be repeated until you reach the line number to be repaired.

```
1000 REM  ******************************************************
2000 REM  *                                                    *
3000 REM  *        CORRLN - REPAIR DAMAGED LINE NO.            *
4000 REM  *          M. P. BARRICK - VFHS - 07/83               *
5000 REM  *         ALL INTECOLORS - USE CORRECT AD             *
6000 REM  *                                                    *
7000 REM  ******************************************************
8000 REM
65000 REM  APPEND THIS PROGRAM TO REPAIR A DAMAGED LINE NO.
65050 REM  BASIC ON THE CC-II STARTS AT 33434
65100 INPUT "START ADDRESS - ";AD
65200 REM  LOCATE FIRST LINE NUMBER AFTER START ADDRESS
65210 IF PEEK(AD)=0 THEN GOSUB 65300
65220 AD=AD+1
65250 GOTO 65200
65300 REM  CONVERT LINE NUMBER TO DECIMAL
65310 LN=PEEK(AD+3)+PEEK(AD+4)*256
65330 GOSUB 65400
65340 AD=AD+4
65350 RETURN
65400 REM  DISPLAY LINE NUMBER AND CHANGE IF DESIRED
65410 PRINT
65420 PRINT "CHANGE LINE NO. ";LN;" TO- "; : INPUT "";CN
65430 IF CN=0 THEN GOTO 65000
65440 HB=INT(CN/256) : LB=CN-HB*256
65460 POKE AD+3,LB : POKE AD+4,HB
65490 RETURN
```

Animated Hourglass

by Tom Andries
815 W. Douglas Rd. Lot #1
Mishawaka, IN 46545

David Suits's book **Color Graphics for the Intecolor 3651 and Compucolor II Computers** impressed me with the speed and elegance of animation that could be achieved using the PRINT command. It also made me wonder how effective and impressive the various graphic plot modes incorporated in the CCII could be in an animated situation. At first it appeared that most of the plot graphics were too sluggish for most animation purposes, but I finally decided that what we needed was merely an appropriate subject. Sand flowing in an hourglass struck me as being a suitable compromise for speed and realism.

The appended program is well-remarked and should generally be easy to follow. However, a few words are in order about the routine and subroutines which move the "sand" from the top of the hourglass to the bottom.

Lines 1070 through 1500 cause the screen memory location where the "sand" is, and where it is going, to be PEEKed at in a predetermined sequence. The contents of a given location will then determine what is POKEd in by the sand movement subroutines. Only the locations actually being operated on are looked at, to keep the routine as speedy as possible. My first intention had been to sequence through the affected cursor positions, but it soon became clear that it would be more efficient to use plot positions and derive the respective X and Y coordinates and screen memory locations from them.

The term "pass" in the descriptive material refers to the manipulation of the hourglass: top-left, top-right, bottom-left and bottom-right, in that order.

The routine moves one "grain" of "sand" at a time in each of these quadrants until the right-center column (plot column 63) is reached. Then the starting pass values are reset for the next diagonal line in each of the four quadrants.

The subroutines to drop and heap the sand use a modified form of the PLOT 2,254,x plot submode. Those of you familiar with the plot submode know that each cursor position on the screen is composed of eight blocks, called plot blocks, which can be turned on or off depending on the value specified for x. Figure 1 shows a cursor position divided up into plot blocks and the value assigned to each block. To create a plot character using this submode, the desired plot blocks are filled in and the binary weights associated with each filled-in block are added together. This is the x value for that particular character. It should be obvious that any combination of plot blocks can be represented by values from 0 to 254. The value 255, which should turn all the blocks on, cannot be used as it is reserved to signal an exit from PLOT mode.

1	16
2	32
4	64
8	128

Figure 1

Actually the plot subroutine is not used at all; the desired x values and CCI codes are POKEd directly into screen memory. This is not only faster than using the PLOT mode, but has the added advantage of allowing us to use the 255 character, which of course is outlawed in PLOT mode. A plot character with a 255 value looks just like a space on the screen, but unlike the space, which is printed in the current background color, the plot character is printed using the foreground color.

found at the top, it has to be yellow, and at the bottom, cyan. When a space is found its location determines what replaces it. If the space is in the top-left quadrant, plot character 16 is used; if in the top-right quadrant, plot character 1; bottom-left, plot character 128; and bottom-right, plot character 8. From now on when we fill in any position with a solid color it will be done with the 255 plot character.

Referring again to Figure 2, and comparing it to the checks made in the

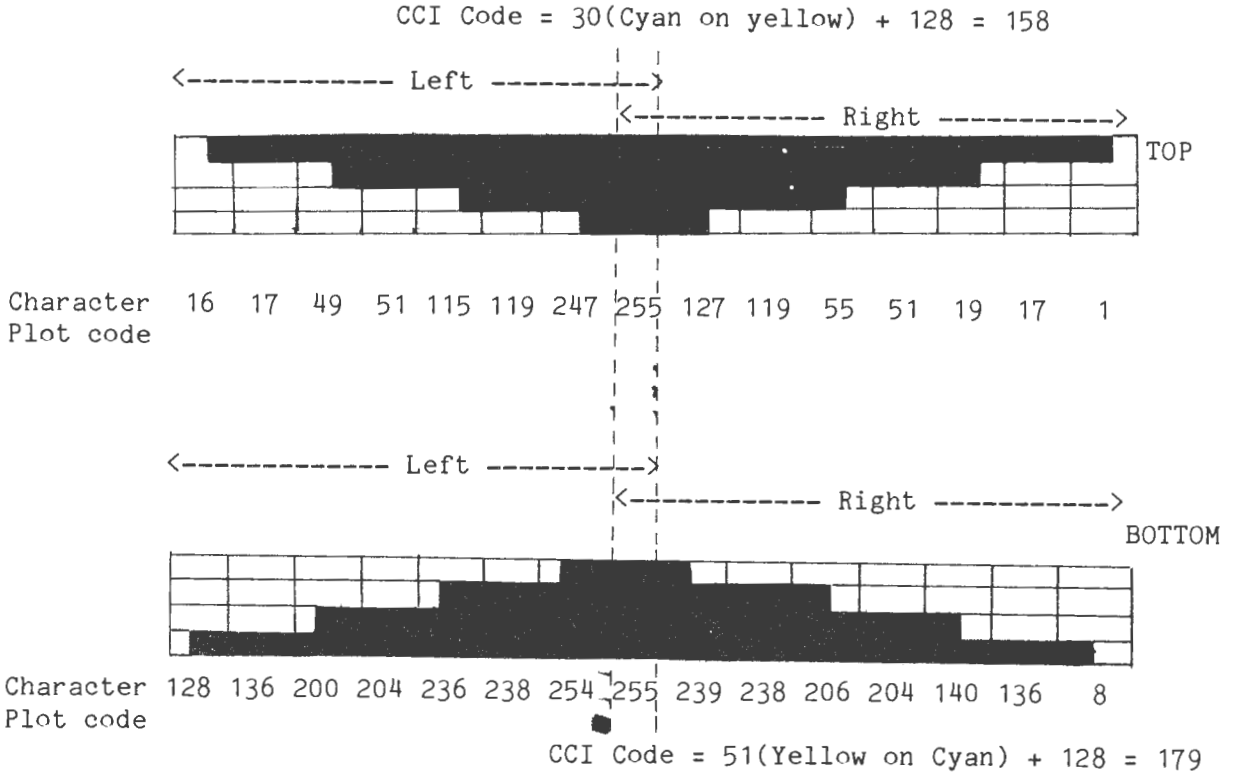


Figure 2

Figure 2 shows the sequence used to advance from one sand grain configuration to the next. The progression is from the outer edges toward the center. The very first thing the PEEKer will find in each virgin screen position is a yellow space (32 in the even-numbered screen memory location and 24 in the next higher screen memory location) at the top, and a cyan space (32 and 48) at the bottom. The CCI codes represent black on yellow and black on cyan, respectively, because that is what we used when the hourglass was first drawn. We don't have to check for the color, though. If a space is

subroutines (lines 1600 to 1750 for the top and 1830 to 1960 for the bottom), we can see that the plot character value found in a given screen location determines the next code to be inserted there. For instance, if either a 49 or 19 is found in a screen location at the top of the hourglass, a 51 replaces it. A 119 will change to either a 247 or a 127 depending on whether the X cursor value is less than 31, or greater than 30 (left or right side). We only have to POKE the CCI code in once, right after finding a space, because once we start painting a location either yellow on cyan

or cyan on yellow, it will never change to anything else.

You may have noticed that the CCI values are 158 and 179 for top and bottom respectively. At the top we want to end up with cyan on yellow which is CCI code 30, and at the bottom we want yellow on cyan which is CCI code 51. To each of these we must add 128. This is how screen memory knows to use a plot character. Without the added 128, we would get a character from the special character set.

The routine for the bottom of the display is similar but not as complex as the top; however, we must also check the bottom for the straight vertical yellow line that makes up the "sand stream" (character 110 from the special character set). Line 1850 takes care of this.

The subroutine composed of lines 1770 through 1810 provides the correctional data to keep our starting pass values accurate when we reach the special triangular characters which form the angled glass at the top. These are not plot characters and must be handled differently. Lines 1730 and 1740 make sure these change from yellow to cyan when the adjacent cursor positions are all cyan. We don't have to worry about this at the bottom as the sand never gets high enough to reach the angled glass. All we have to do there is freeze the X position value once we reach the edges, and increment the Y value as each row fills with sand. Line 1480 checks the last top location to be filled in. If it is cyan we're done except for erasing the remainder of the sand stream.

All in all, I think the action is impressive and realistic. I was pleasantly surprised to discover that the sand completes its movement from the top to bottom in 3 minutes and ten seconds, which is roughly the time it takes for a great number of egg and telephone-call timers of this type.

```
1 REM         * * * * * * * * * * * * * * * * * * * * * * * *
2 REM         *  'ANIMATED HOURGLASS' BY TOM ANDRIES     *
3 REM         *        315 W. DOUGLAS RD. LOT #1         *
4 REM         *         MISHAWAKA, INDIANA 46545         *
5 REM         *           SEPTEMBER 21, 1983             *
6 REM         * * * * * * * * * * * * * * * * * * * * * * * *
7 REM
8 REM               ***   DRAW THE HOURGLASS   ***
9 REM
10 PLOT 6,0:REM    SET COLOR, BLACK ON BLACK
20 PLOT 12:REM     ERASE SCREEN
30 PLOT 15:REM     SMALL CHARACTERS
40 PLOT 27,10:REM     WRITE VERTICAL
50 PLOT 29:REM     MAKE SURE FLAG IS OFF
60 REM                ***   DRAW LEFT UPRIGHT   ***
70 PLOT 6,56:REM    SET FRAME COLOR - BLACK ON WHITE
80 FOR X= 19TO 21:REM     CURSOR COLUMN FOR LEFT UPRIGHT
90 PLOT 3,X,1:REM     SET CURSOR STARTING POSITION
100 FOR Y= 0TO 30:REM     CURSOR ROW
110 PLOT 96:REM     PRINT CROSSHATCH CHARACTER
120 NEXT Y:REM     NEXT ROW
130 PRINT :REM     DON'T LET THE CHARACTER COUNTER GET YOU!
140 NEXT X:REM     NEXT COLUMN
150 REM                ***  DRAW RIGHT UPRIGHT   ***
160 FOR X= 41TO 43:REM     COLUMN
170 PLOT 3,X,1:REM     SET CURSOR
180 FOR Y= 0TO 30:REM     ROW
190 PLOT 96:REM     PRINT CROSSHATCH
200 NEXT Y:REM     NEXT ROW
210 PRINT :REM     WATCH THE CHARACTER COUNT!
220 NEXT X:REM     NEXT COLUMN
230 REM                ***   DRAW TOP CROSS-MEMBER   ***
240 PLOT 27,24:REM     SET PAGE MODE, WRITE LEFT TO RIGHT
250 FOR Y= 2TO 3:REM     TWO ROWS HIGH
260 PLOT 3,22,Y:REM     SET CURSOR
270 FOR X= 22TO 40:REM     19 COLUMNS WIDE
280 PLOT 96:REM     PRINT CROSSHATCH
290 NEXT X:REM     NEXT COLUMN
300 PRINT
310 NEXT Y:REM     NEXT ROW
```

```
880 PLOT 6,6:REM     CYAN ON BLACK
890 PLOT 124:REM     LEFT-TO-RIGHT DIAGONAL
900 PRINT :REM     CHARACTER COUNT
910 NEXT Y:REM     NEXT ROW TILL DONE
920 FOR I= 1TO 1000:NEXT I:REM     WAIT AWHILE
930 REM              ***   START THE SAND STREAM   ***
940 PLOT 3,64,0:REM     HIDE THE VISIBLE CURSOR
950 FOR Y= 17TO 28:REM     12 ROWS OF VERTICAL YELLOW LINES
960 PLOT 3,127:REM     BLIND CURSOR, SMALL CHARACTERS
970 PLOT 31:REM     COLUMN (CENTER OF SCREEN)
980 PLOT Y:REM     ROW
990 PLOT 51:REM     BLIND CURSOR CCI CODE - YELLOW ON CYAN
1000 PLOT 110:REM     VERTICAL LINE (SEE CHARACTER SET)
1010 FOR I= 1TO 15:NEXT I:REM     WAIT A BIT TO SLOW STREAM
1020 NEXT Y:REM     NEXT ROW
1030 REM
1040 REM           ***   ROUTINE TO DROP SAND   ***
1050 REM
1060 REM     SET UP STARTING PARAMETERS
1070 C= 0:REM     COUNTER TO KEEP TRACK OF PASSES
1080 TL= 62:REM     TOP-LEFT COLUMN
1090 TR= 63:REM     TOP-RIGHT COLUMN
1100 BL= 62:REM     BOTTOM-LEFT COLUMN
1110 BR= 63:REM     BOTTOM-RIGHT COLUMN
1120 TY= 95:REM     TOP ROW
1130 YB= 11:REM     BOTTOM ROW
1140 LT= 47:REM     TOP-LEFT EDGE LIMIT
1150 RT= 78:REM     TOP-RIGHT EDGE LIMIT
1160 LB= 43:REM     BOTTOM-LEFT EDGE LIMIT
1170 RB= 82:REM     BOTTOM-RIGHT EDGE LIMIT
1180 YT= 95:REM     TOP ROW LIMIT
1190 BY= 11:REM     BOTTOM ROW LIMIT
1200 CS= 29772:REM     SCREEN CHECK FOR TOP EDGE LIMIT ADJUSTMENT
1210 LS= 32:REM     CHECK VALUE FOR TOP-LEFT DIAGONAL PLOTTING
1220 REM     - THE ROUTINE -
1230 X= INT (TL/ 2):REM     DERIVE TOP-LEFT COLUMN FROM PLOT POSITION
1240 Y= INT ((127- TY)/ 4):REM     DERIVE TOP-LEFT ROW
1250 SC= 128* Y+ X+ X+ 28672:REM     CALCULATE SCREEN MEMORY LOCATION
1260 GOSUB 1610:REM     SEE WHAT'S IN THERE AND REPLACE IT
1270 X= INT (TR/ 2):REM     DERIVE TOP-RIGHT COLUMN
1280 SC= 128* Y+ X+ X+ 28672:REM     CALCULATE TOP-RIGHT SCREEN LOCATION
1290 GOSUB 1610:REM     REPLACE THE CONTENTS
1300 TL= TL+ 1:REM     UPDATE TOP-LEFT PLOT POSITION
1310 TR= TR- 1:REM     UPDATE TOP-RIGHT PLOT POSITION
1320 TY= TY- 1:REM     UPDATE TOP PLOT POSITION ROW
1330 IF TR= 62THEN C= C+ 1:GOSUB 1770:REM     SET UP NEXT PASS
1340 YB= YB+ 1:REM     UPDATE BOTTOM ROW PLOT POSITION
1350 X= INT (BL/ 2):REM     DERIVE BOTTOM-LEFT COLUMN
1360 Y= INT ((127- YB)/ 4):REM     DERIVE BOTTOM-LEFT ROW
1370 SC= 128* Y+ X+ X+ 28672:REM     CALCULATE SCREEN MEMORY LOCATION
1380 GOSUB 1830:REM     SEE WHAT'S THERE, POKE SOMETHING NEW
1390 BL= BL+ 1:REM     UPDATE BOTTOM-LEFT PLOT POSITION
1400 IF BL= 63THEN BL= 62- C:REM     SET UP FOR NEXT PASS
1410 IF BL= < LBTHEN BL= LB+ 1:REM     PAST LEFT EDGE?  FREEZE IT!
1420 X= INT (BR/ 2):REM     DERIVE BOTTOM-RIGHT COLUMN POSITION
320 REM              ***   DRAW BOTTOM CROSS-MEMBER   ***
330 FOR Y= 29TO 30:REM     TWO ROWS
340 PLOT 3,22,Y:REM     SET CURSOR
350 FOR X= 22TO 40:REM     19 COLUMNS
360 PLOT 96:REM     PRINT CROSSHATCH
370 NEXT X:REM     NEXT COLUMN
380 PRINT
390 NEXT Y:REM     NEXT ROW
400 REM        ***   DRAW TOP AND BOTTOM STRAIGHT-SIDED GLASS
410 PLOT 6,48:REM     SET GLASS COLOR (BLACK ON CYAN)
420 FOR Y= 4TO 6:REM     THREE ROWS OF STRAIGHT-SIDED GLASS
430 PLOT 3,22,Y:REM     SET CURSOR START
440 PRINT SPC( 19):REM     PRINT CYAN SPACES
450 NEXT Y:REM     NEXT ROW
460 PRINT :REM     DONE AT THE TOP
470 REM     BOTTOM
480 FOR Y= 26TO 28:REM     THREE ROWS OF STRAIGHT-SIDED GLASS
490 PLOT 3,22,Y:REM     SET CURSOR START
500 PRINT SPC( 19):REM     CYAN SPACES
510 NEXT Y:REM     NEXT ROW
520 PRINT :REM     DONE AT BOTTOM
530 REM        ***   DRAW CLEAR SLOPED GLASS AT TOP   ***
540 PLOT 3,22,7:REM     CURSOR START
550 PLOT 124:REM     LEFT-TO-RIGHT DIAGONAL (SEE CHARACTER SET)
560 PRINT SPC( 17);:REM     CYAN SPACES
570 PLOT 126:REM     RIGHT-TO-LEFT DIAGONAL (SEE CHARACTER SET)
580 PRINT :REM     DONE WITH CYAN
590 REM        ***   FILL TOP WITH SAND   ***
600 PLOT 6,24:REM     BLACK ON YELLOW
610 REM     SET STARTING VALUES
620 X= 22:REM     COLUMN
630 SP= 17:REM     NUMBER OF YELLOW SPACES
640 FOR Y= 8TO 15:REM     ROW NUMBERS
650 X= X+ 1:REM     INCREMENT COLUMN
660 SP= SP- 2:REM     DECREASE NUMBER OF SPACES BY 2
670 PLOT 3,X,Y:REM     CURSOR START
680 PLOT 124:REM     LEFT-TO-RIGHT DIAGONAL
690 PRINT SPC( SP):REM     YELLOW SPACES
700 PLOT 126:REM     RIGHT-TO-LEFT DIAGONAL
710 PRINT :REM     CLEAR CHARACTER COUNT BUFFER
720 NEXT Y:REM     NEXT ROW UNTIL DONE AT TOP
730 REM              ***   DRAW "WAIST" OF GLASS   ***
740 PLOT 6,6:REM     CYAN ON BLACK
750 PLOT 3,31,16:REM     SET CURSOR TO CENTER OF DISPLAY
760 PLOT 120:REM     DRAW X-SHAPED CHARACTER (SEE CHARACTER SET)
770 REM              ***   DRAW SLOPED BOTTOM GLASS   ***
780 X= 31:REM     COLUMN START
790 SP= - 1:REM     NUMBER OF SPACES
800 FOR Y= 17TO 25:REM     NINE ROWS
810 X= X- 1:REM     DECREMENT COLUMN POSITION
820 SP= SP+ 2:REM     INCREASE NUMBER OF SPACES
830 PLOT 6,6:REM     CYAN ON BLACK
840 PLOT 3,X,Y:REM     SET CURSOR START
850 PLOT 126:REM     RIGHT-TO-LEFT DIAGONAL
860 PLOT 6,48:REM     BLACK ON CYAN
870 PRINT SPC( SP):REM     CYAN SPACES
```

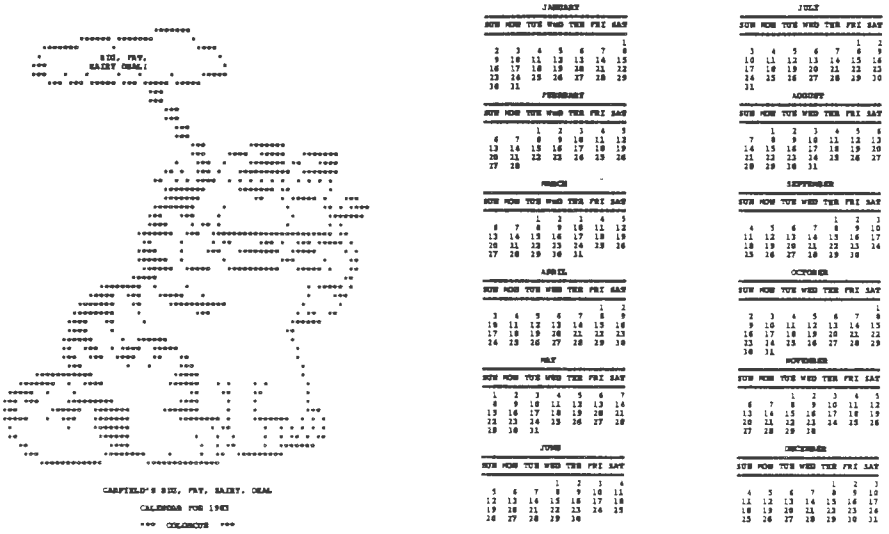
```
1430 SC= 128* V+ X+ X+ 28672:REM    CALCULATE SCREEN MEMORY LOCATION
1440 GOSUB 1830:REM    CHECK IT AND CHANGE IT
1450 BR= BR- 1:REM    UPDATE BOTTOM-RIGHT PLOT POSITION
1460 IF BR= 62THEN BR= 63+ C:VB= BV:REM    SET UP NEXT PASS
1470 IF BR= > RBTHEN BR= RB- 1:BV= BV+ 1:VB= BV:REM    RIGHT EDGE?
1480 IF PEEK (30654)= 255THEN 1520
1490 REM    CYAN SPACE THERE?  DONE, EXCEPT FOR CLEAN-UP.
1500 GOTO 1230:REM    DROP SOME MORE SAND
1510 REM          ***   GET RID OF REMAINING SAND STREAM   ***
1520 FOR V= 17TO 22:REM    FIVE ROWS
1530 PLOT 3,127,31,V,51:REM    BLIND CURSOR - BLACK ON CYAN
1540 PLOT 32:REM    PRINT SPACE OVER "SAND STREAM"
1550 FOR I= 1TO 15:NEXT I:REM    WAIT A BIT FOR REALISM
1560 NEXT V:REM    DO IT AGAIN
1570 PLOT 6,2:REM    BACK TO NORMAL PRINT
1572 PLOT 3,0,13
1574 PRINT "YOUR 3-MINUTE ":PRINT
1576 PRINT "EGG IS "
1578 PLOT 3,5,17:PRINT "!":PLOT 28,28
1580 PLOT 27,11:REM    ENABLE SCROLL MODE
1590 END
1600 REM      ***   SUBROUTINE TO DROP SAND   ***
1610 IF PEEK (SC)= 32AND X> 30THEN POKE SC,1:POKE SC+ 1,158:GOTO 1730
1620 IF PEEK (SC)= 32THEN POKE SC,16:POKE SC+ 1,158:GOTO 1730
1630 IF PEEK (SC)= 10R PEEK (SC)= 16THEN POKE SC,17:GOTO 1730
1640 IF PEEK (SC)= 17AND X> 30THEN POKE SC,19:GOTO 1730
1650 IF PEEK (SC)= 17THEN POKE SC,49:GOTO 1730
1660 IF PEEK (SC)= 49OR PEEK (SC)= 19THEN POKE SC,51:GOTO 1730
1670 IF PEEK (SC)= 51AND X> 30THEN POKE SC,55:GOTO 1730
1680 IF PEEK (SC)= 51THEN POKE SC,115:GOTO 1730
1690 IF PEEK (SC)= 55OR PEEK (SC)= 115THEN POKE SC,119:GOTO 1730
1700 IF PEEK (SC)= 119AND X> 30THEN POKE SC,127:GOTO 1730
1710 IF PEEK (SC)= 119THEN POKE SC,247:GOTO 1730
1720 IF PEEK (SC)= 127OR PEEK (SC)= 247THEN POKE SC,255
1730 IF X< 32AND X+ V= LSAND PEEK (SC)= 255THEN POKE SC- 2,124:POKE
     SC- 1,48:LS= LS+ 2
1740 IF X+ V= 46AND PEEK (SC)= 255THEN POKE SC+ 2,126:POKE SC+ 3,48
1750 RETURN
1760 REM      ***   SUBROUTINE TO RESET TOP PASS VALUES   ***
1770 TL= 62- C:TR= 63+ C
1780 IF PEEK (CS)= 255THEN LT= LT+ 2:RT= RT- 2:VT= VT- 2:CS= CS+ 126
1790 IF TR= > RTTHEN TR= RT- 1:TL= LT+ 1:VT= VT- 1:TV= VT:GOTO 1810
1800 TV= VT
1810 RETURN
1820 REM          ***   SUBROUTINE TO ACCUMULATE SAND   ***
1830 IF PEEK (SC)= 32AND X> 30THEN POKE SC,8:POKE SC+ 1,179:GOTO 1960
1840 IF PEEK (SC)= 32THEN POKE SC,128:POKE SC+ 1,179:GOTO 1960
1850 IF PEEK (SC)= 110THEN POKE SC,8:POKE SC+ 1,179:GOTO 1960
1860 IF PEEK (SC)= 80R PEEK (SC)='128THEN POKE SC,136:GOTO 1960
1870 IF PEEK (SC)= 136AND X> 30THEN POKE SC,140:GOTO 1960
1880 IF PEEK (SC)= 136THEN POKE SC,200:GOTO 1960
1890 IF PEEK (SC)= 140OR PEEK (SC)= 200THEN POKE SC,204:GOTO 1960
1900 IF PEEK (SC)= 204AND X> 30THEN POKE SC,206:GOTO 1960
1910 IF PEEK (SC)= 204THEN POKE SC,236:GOTO 1960
1920 IF PEEK (SC)= 206OR PEEK (SC)= 236THEN POKE SC,238:GOTO 1960
1930 IF PEEK (SC)= 238AND X> 30THEN POKE SC,239:GOTO 1960
1940 IF PEEK (SC)= 238THEN POKE SC,254:GOTO 1960
1950 IF PEEK (SC)= 239OR PEEK (SC)= 254THEN POKE SC,255
1960 RETURN
```

HAVE YOU RENEWED?

It's renewal time for most subscribers. In order for COLORCUE to continue, your subcriptions are urgently needed. Don't delay!

Garfield Hairy Deal Calendar

revised for CCII **by Mike Barrick**
Valley Forge High School
9999 Independence Blvd.
Parma Heights, OH 44130



```
1000 REM  ************************************************
2000 REM  *                                              *
3000 REM  *    GARFLD - GARFIELD HAIRY DEAL CALENDAR     *
4000 REM  *   AUTHOR :  CARL REINKE - VFHS - MAY 1983     *
5000 REM  *    REVISED FOR CCII - M. BARRICK - 5/83      *
6000 REM  *                                              *
7000 REM  ************************************************
8000 REM
9000 CLEAR 1000 : DIM A(12,42),A$(12)
10000 REM   ROUTINE :  MAIN PROGRAM
10300 L=0 :   PLOT 12 :
10400 PRINT "GARFIELD CALENDAR BY KARL REINKE "
10500 PRINT "    COMPUCOLOR II VER 8/79"
10600 PRINT
10700 INPUT "ENTER YEAR NUMBER - ";YR
10800 PRINT  : IF Y/4=INT(YR/4) THEN L=1
10900 PRINT "SUNDAY = 1 ..... SATURDAY = 7"
11000 INPUT "ENTER DAY THE YEAR BEGINS - ";D
11100 IF D>7 OR D<1 THEN 11000
12100 PRINT
12200 PRINT "TYPE PERSONALIZED MESSAGE BELOW:"
12300 INPUT "";H$(3)
12600 PRINT
12700 INPUT "TYPE  4  FOR 1200 BAUD - ";PR
12800 IF PR>0 THEN PLOT 27,18,PR
14200 PLOT 27,13 : POKE 33289,80
14400 H$(1)="GARFIELD'S BIG, FAT, HAIRY, DEAL"
14600 H$(2)="CALENDAR FOR"+STR$(YR)
14700 PRINT  : PRINT  : PRINT  : PRINT
15000 REM   BEGIN GARFIELD PICTURE ROUTINE
15100 READ A,B,C : IF A=999 THEN 20000
15300 FOR X=A TO B : PRINT TAB(X);"*"; : NEXT X
16100 IF C=1 THEN PRINT  : GOTO 15000
16300 IF C=2 THEN PRINT "           BIG, FAT,"; : GOTO 15000
16500 IF C=3 THEN PRINT "        HAIRY DEAL!";
17500 GOTO 15000
```

```
20000 REM   BEGIN CALENDAR PRINTING ROUTINE
20200 FOR X=1 TO 12 : READ A$(X),N : IF X=2 AND L=1 THEN N=N+1
20400 FOR Y=1 TO N : A(X,Y+(D-1))=Y : NEXT Y
20600 D=D-(28-N) : IF D>7 THEN D=D-7
20700 NEXT X
20800 PRINT  : PRINT
21200 FOR I=1 TO 3 : PRINT TAB((80-LEN(H$(I)))/2)H$(I)
21300 PRINT  : NEXT I
21500 OUT 6,12
22100 FOR X=1 TO 6
22300 PRINT TAB((25-LEN(R$(X)))/2);A$(X);
22500 PRINT TAB(49+((27-LEN(A$(X+6)))/2));A$(X+6)
22700 Z=0 : GOSUB 40000
23100 PRINT TAB(01);"SUN MON TUE WED THR FRI SAT";
23300 PRINT TAB(49);"SUN MON TUE WED THR FRI SAT"
23500 Z=0 : GOSUB 40000
25200 FOR Y=1 TO 6 : Q=1 : U=0 : FOR Z=1+(7*(Y-1)) TO 7*Y
25400 GOSUB 45000 : NEXT Z
25600 Q=49 : U=6
26200 FOR Z=1+(7*(Y-1)) TO 7*Y : GOSUB 45000 : NEXT Z
26400 PRINT
26600 NEXT Y
26800 NEXT X
28200 PRINT  : PRINT
28300 PRINT TAB(03)"PROGRAMMED BY KARL REINKE";
28400 PRINT " *** VALLEY FORGE HIGH SCHOOL - CLASS OF 1983"
29700 OUT 6,12 : OUT 6,12 : POKE 33265,0
29900 RESTORE 10000 : GOTO 10000
40000 REM
40500 FOR Y=Z+1 TO Z+27 : PRINT TAB(Y)"="; : NEXT Y
40700 IF Z=0 THEN Z=48 : GOTO 40000
40800 PRINT
40900 RETURN
45000 REM
45300 PRINT TAB(Q-1);
45500 IF A(X+U,Z)=0 THEN PRINT "    "; : RETURN
45600 DN$=STR$(A(X+U,Z)) : IF A(X+U,Z)<10 THEN DN$=" "+DN$
45700 DN$=" "+DN$ : PRINT DN$;
45900 RETURN
50000 REM   GARFIELD PICTURE DATA
51100 DATA 33,39,1,19,24,0,26,32,0,40,40,1
51200 DATA 12,18,0,25,25,0,41,44,1
51300 DATA 11,11,2,45,45,1
51400 DATA 10,12,3,46,46,1
51500 DATA 10,12,0,16,16,0,20,20,0
51600 DATA 26,26,0,30,30,0,36,36,0,42,46,1
52100 DATA 13,15,0,17,19,0,21,25,0,27,29,0,31,35,0,37,41,1
52200 DATA 32,34,1
52300 DATA 32,34,1
52400 DATA 35,37,1
52500 DATA 35,37,1
52600 DATA 37,39,1
52700 DATA 37,39,1
52800 DATA 40,42,0,51,57,1
52900 DATA 40,42,0,47,50,0,53,57,0,61,67,1
53100 DATA 37,37,0,41,43,0,46,46,0,51,57,0,59,60,0,64,67,1
53200 DATA 35,36,0,38,38,0,41,43,0,45,45,0,49,58,0,61,67,1
53300 DATA 33,34,0,37,37,0,39,39,0,41,44,0,47,47,0,49,49,0
53400 DATA 51,51,0,53,53,0,55,55,0
53500 DATA 57,57,0,60,60,0,62,62,0,64,64,0,66,66,1
53600 DATA 35,42,0,49,55,0,57,58,0,66,66,1
53700 DATA 35,41,0,52,53,0,57,57,0,59,63,0,65,65,0,69,71,1
53800 DATA 33,40,0,46,46,0,58,58,0,65,66,0,68,68,0,70,73,1
53900 DATA 32,34,0,38,40,0,45,45,0,59,59,0,66,72,1
54100 DATA 32,35,0,40,40,0,42,43,0,45,45,0
54150 DATA 58,58,0,66,66,0,68,68,1
54200 DATA 30,36,0,39,41,0,44,66,0,68,69,1
54300 DATA 31,32,0,38,38,0,44,44,0,46,47,0,53,55,0,57,57,0
54400 DATA 60,62,0,64,68,0,70,70,1
54500 DATA 30,33,0,38,38,0,43,45,0,49,56,0
54700 DATA 59,63,0,69,69,0,71,71,1
55100 DATA 30,35,0,38,38,0,46,46,0,57,59,0,65,65,0,70,71,1
55200 DATA 31,36,0,39,43,0,47,56,0,60,64,0,66,66,0,70,71,1
55300 DATA 29,33,0,69,70,1
55400 DATA 23,30,0,33,33,0,39,39,0,62,62,0,64,68,0,70,70,1
55500 DATA 21,25,0,28,30,0,34,38,0,42,42,0,62,62,0,67,69,1
55600 DATA 19,25,0,29,30,0,36,46,0,61,61,0,68,68,1
55700 DATA 18,25,0,30,30,0,41,43,0,60,60,0,62,63,0,66,67,1
55800 DATA 17,20,0,24,25,0,31,31,0,59,64,1
55900 DATA 17,20,0,25,25,0,58,59,1
56100 DATA 16,21,0,27,27,0,30,35,0,59,59,1
56200 DATA 16,17,0,20,22,0,25,25,0,27,30,0,33,37,0,60,60,1
56300 DATA 16,18,0,21,23,0,26,29,0,33,34,0,38,39,0,60,60,1
56500 DATA 17,19,0,28,28,0,32,32,0,38,40,0,61,61,1
56700 DATA 11,21,0,37,41,0,61,61,1
56900 DATA 08,14,0,17,17,0,20,23,0,36,41,0,45,45,0
57100 DATA 47,47,0,52,52,0,54,54,0,62,62,1
57300 DATA 06,10,0,13,13,4,18,18,0,40,41,0
57500 DATA 46,47,0,52,53,0,62,62,1
57700 DATA 05,07,0,09,11,0,17,25,0,38,41,0,47,47,0
57900 DATA 52,52,0,62,62,1
58100 DATA 05,07,0,14,19,0,22,28,0,35,41,0,47,47,0
58300 DATA 52,52,0,62,64,1
58500 DATA 05,09,0,13,13,0,16,18,0,22,28,0,38,45,0
58700 DATA 47,47,0,52,58,0,61,61,0,63,63,0,65,65,1
58900 DATA 06,06,0,17,17,0,22,28,0,44,44,0,46,47,0
59100 DATA 54,54,0,57,57,0,59,59,0,61,61,0,63,63,0,65,65,1
59300 DATA 06,07,0,23,28,0,41,41,0,45,45,0,48,48,0
59500 DATA 54,54,0,57,57,0,59,60,0,62,62,0,64,65,1
59700 DATA 08,09,0,25,28,0,42,42,0,45,45,0,47,48,0
59900 DATA 54,54,0,57,63,1
60100 DATA 09,11,0,24,31,0,36,36,0,42,42,0,45,46,0,49,57,1
60300 DATA 12,23,0,32,45,1
60500 DATA 999,999,999
62000 REM    *** CALENDAR DATA ***
62100 DATA "JANUARY",31,"FEBRUARY",28
62200 DATA "MARCH",31,"APRIL",30
62300 DATA "MAY",31,"JUNE",30,"JULY",31
62400 DATA "AUGUST",31,"SEPTEMBER",30
64500 DATA "OCTOBER",31,"NOVEMBER",30
64600 DATA "DECEMBER",31
```

Back Issues Sale

Back issues of **Colorcue** are an excellent source of information about Compucolor computers, ISC computers, and programming in general. Interviews, interesting articles, and programs are all there with a touch of history.

The list below includes every **Colorcue** ever published. If it's not on the list, then there wasn't one.

RETROVIEW: Vol. 3, #1 (Dec 79/Jan 80) includes: an interview with Bill Greene; Compucolor-teletype interface; user group hotline; introduction to the Screen Editor; PEEKing at BASIC programs; talking to other computers; making programs compatible with V6.78 and V7.80 software; software modifications.

MULTI-ISSUES at $3.50 each

__ Oct, Nov, Dec 1978
__ Jan, Feb, Mar 1979
__ Apr, May/Jun 1979
__ Aug, Sept/Oct, Nov 1979

INDIVIDUAL ISSUES at $1.50 each

__ Dec 1979/Jan 1980
__ Feb 1980
__ Mar 1980
__ Apr 1980
__ May 1980
__ Jun/Jul 1980

INDIVIDUAL ISSUES at $2.50 each

__ Dec 1980/Jan 1981
__ Aug/Sept 1981
__ Oct/Nov 1981
__ Dec 1981/Jan 1982
__ Feb/Mar 1982
__ Apr/May 1982
__ Jun/Jul 1982
__ Aug/Sept 1982
__ Oct/Nov 1982
__ Dec 1982/Jan 1983
__ Feb/Mar 1983
__ Apr/May 1983
__ Jun/Jul 1983

POSTAGE

US, Canada and Mexico -- First Class Postage included.
Europe, S. America -- Add $1.00 per item for air, or
$ .40 per item for surface.
Asia, Africa, Middle East -- add $1.40 per item for air, or
$ .60 per item for surface

Discount

For orders of 10 or more items, subtract 25%
from total after postage.

ORDER FROM: **Colorcue**
Editorial Offices
161 Brookside Drive
Rochester, NY 14618

BULK RATE
U.S. POSTAGE
PAID
Rochester, N. Y.
Permit No. 415

Colorcue
161 Brookside Dr.
Rochester, NY 14618