

Colorcue



Aug/Sept 1981

\$2.

Colorcue

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

August/September 1981
Volume 4, Number 1

Editors:
Ben Barlow
David B. Suits

-
- 3 **Editors' Notes**
 - 3 **Compucolor Policy**
 - 5 **The Serial Port, by Ben Barlow**
How to use the RS232 port.
 - 13 **Lower Case y's, by Tom Devlin**
A simple, cheap fix for a nagging problem
 - 14 **Screen Dump to MX-80, by Mark Fairbrother**
Subroutine to move graphic image from screen to printer
 - 17 **User Group Bulletin Board**
 - 19 **Assembly Language Programming, by David B. Suits**
Part 1 of a series for the beginner
-

Advertisers: You will find our advertising policies attractive! Write for details.

Authors: This is a user-oriented and supported publication. Your articles/tips/hints are required to make it go. Send your articles or write for information.

Colorcue is published bi-monthly by Intelligent Systems Corporation, with editorial offices in Rochester, New York. Editorial and subscription correspondence should be addressed to the Editors, Colorcue, 161 Brookside Drive, Rochester, NY 14618. Product related correspondence should be addressed to ISC, 225 Technology Park, Norcross, GA 30092, ATTN: Susan Sheridan. Reproduction in whole or part without permission is discouraged. Opinions expressed in by-line articles are not necessarily those of the editors or of ISC. Hardware/software items are checked to the best of our abilities, but are NOT guaranteed.

Editors' Notes . . .

The editorial duties of Colorcue have been assumed by a new editor. In fact, there are two of us. Since we plan to be around for a while, let us introduce ourselves.

David B. Suits obtained his PhD. in philosophy from the University of Waterloo in Ontario, Canada, and teaches philosophy at the Rochester Institute of Technology. He has published articles in both philosophy and microcomputer journals and has written a book about Intecolor/Compucolor graphics. Dave became interested in computers when he discovered that his programmable calculator was too small to handle a chess-playing program. He's been the librarian of the Rochester Compucolor Users Group for 2 1/2 years and is interested in artificial intelligence, music, and hardware.

Ben Barlow is a computer hacker. A systems programmer for many years, he is now involved with computers only as a hobby. He works for the Xerox Corporation in Rochester, and for the past 2 1/2 years he has edited and published Datachip, the Rochester Compucolor Users Group newsletter. His interests lie in the area of hardware and external world interfaces.

A word about policies is in order, especially in light of the recent shifts of Colorcue direction. Colorcue will continue to be a bi-monthly (every 2 months) publication directed toward both new and experienced users of Intecolor and Compucolor compu-

ters. We are committed to its timely publication and to a goal of providing top-quality, accurate and useful information to our readers. At the same time, however, we're dependent upon your participation. We need your articles, programs and ideas to make this magazine into what it should be and what we're sure you want it to be. Let us and your fellow users know what you've been doing with your computer.

All current subscriptions are good now for one year (six issues), as are the applications of those of you who responded to the query about an Intecolor publication. No renewal worries for anybody until next year! So, sit back, read, enjoy. (And, we trust, learn.)

Dave and Ben

Compucolor Policy

Because of a ruling by the Federal Communications Commission, Intelligent Systems Corporation is no longer permitted to sell the Compucolor II personal computer in the United States. The ruling stems from the FCC's determination that most personal computers generate substantial interference--resulting in impaired television reception. You may have noticed that when you work with the Compucolor II, a television viewer in the next room can have troubles getting a good picture. This problem is especially troublesome to apartment resi-

dents, who have no control over the activities of neighbors in adjoining rooms. Since televisions and personal computers are communication media, the FCC stepped in. Vacuum cleaners, hair dryers, and other appliances are not affected by the ruling, even though they are often much more annoying to the television viewer, because they are not communications equipment.

But personal computer owners need not be worried that they are operating machines illegally. The ruling applies only to machines manufactured after the FCC deadline, which, in the case of Intelligent Systems, was April, 1981. All Compucolor IIs already in service are perfectly acceptable--no modifications are necessary.

Virtually all personal computer manufacturers were affected by the FCC's decision. Companies such as Apple and Radio Shack were faced with modifying their products in order to meet the new standards. Intelligent Systems started to research the changes in design, components, and production that would have to be made in order to meet the new standards. Our financial planners and marketing people assessed the overall costs of such changes and weighed them against current profits and projected profits. Then we investigated all possible alternatives to come up with a final decision about how we would comply with the ruling.

In the course of studying the problem, it became evident that re-designing the Compucolor II in order to comply with the FCC standards was simply too expensive. It would mean further channeling of resources away from

the company's mainstay line of business and scientific computers, and that was not feasible. Consequently, Intelligent Systems Corporation had no choice but to discontinue manufacture of the Compucolor II. However, since the FCC ruling does not affect overseas clients, the company still manufactures Compucolor IIs for sale in foreign markets.

Once the decision was made to discontinue the Compucolor II for domestic markets, company leaders turned their attention to Compucolor II owners in order to determine how they might best be served. A two-point plan was developed that serves past customers well beyond any legal or moral responsibility. Intelligent Systems Corporation provides:

(1) A source of information -- Colorcue. Despite increasing costs of postage, editorial time, and materials, Colorcue will continue to be published. This newsletter supplies users with valuable information such as names of people who welcome contact from other users, and user group locations for those who wish to meet with other owners. Intelligent Systems decided to sponsor publication of the newsletter so that all users will have a place to turn for information.

(2) A source of parts and repair. Intelligent Systems continues to maintain its repair facility at the factory. Replacement parts and trained service personnel are available to give satisfactory turn-around time at a reasonable cost. This ensures that all owners will be able to maintain their units properly for best performance. **C**

The Serial Port

by Ben Barlow

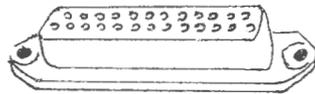
Stop! You apply the brakes as the traffic light changes first to yellow and then to red. You are obeying the rules associated with a standard established back in the 1930's just as your computer's RS232 port obeys the RS232 standard as it sends and receives data. The traffic light's purpose is to control traffic flow; the RS232's purpose is to control data flow. Just as the traffic light does not care what kind of traffic is stopping below it, the RS232 standard specifies rules for sending and receiving data but does not restrict the data being sent. The only requirement is that data characters be transmitted in a serial (bit by bit) manner, rather than in parallel (all bits simultaneously). RS232 is not concerned with the type of code, parity, or the number of stop bits, synchronous or asynchronous transmission - only with the signalling levels to be used to transmit data and the protocol used to interchange it.

Why have a standard, anyway? The answer is obvious, just as it is at the intersection of two streets. Some commonly understood method of control is required to make sure control is orderly, and to ensure compatible products. A good standard serves as the base for manufacturers to develop products, and promotes healthy competition (read lower price). Look at the host of different terminal and computer devices on the market manufactured by a myriad of manufacturers whose products are compatible because of the RS232 standard.

The RS232 standard (current revision level C adopted in 1969) governs the connection of terminals and computing equipment to modems. Although it's not intended to describe direct terminal to computer interfacing, it's been used that way often, especially since the onset of personal computing. The standard defines three aspects of the interface - 1) the electrical standard (which is what most advertising refers to) 2) the mechanical standard, and 3) the interchange, or protocol standard. The latter is frequently determined by the software in the respective computers and terminals. Let's look at these three aspects separately.

The easiest one to understand is the mechanical interface. The ubiquitous DB25 connector has become almost synonymous with RS232C. This familiar 25 pin plug (male) and socket (female) combination appears on the end of every cable and is built into most equipment that meets the

standard. Some of the ISC computers use an edge connector to provide the RS232 interface, which must have an added cable to give a DB25-type connection. By convention, terminals and modems usually have female connectors, which leads to male/male cables, but this is not universally true.



female - S



male - P

DB 25 Connectors

Fig. 1

Electrical characteristics are also easy to understand. RS232 signals are based on specified voltage levels. Your computer is constructed largely of TTL family devices (integrated circuits). This family also uses voltage levels for signal determination. TTL levels range between 0 and 5 volts; logic 0 is recognized as being a voltage between 0 and .4 volts, and logic 1 is recognized as being between 2.4 and 5.0 volts. A voltage between .4V and 2.4V is undefined. This is a relatively small difference in voltage, and can easily be affected by electrical surges caused by motors turning on (refrigerators, copiers, vacuum cleaners), or even by line capacitance if the TTL signal must be passed over a long (measured in feet) distance. Thus, it's unacceptable as a data communication method over any useful distance.

The RS232C standard, on the other hand, specifies voltages ranging from -25V to +25V and provides much greater immunity to noise. A positive signal is a voltage between 3V and 25V, and a negative signal is a voltage between -25V and -3V. The area between -3V and +3V is undefined. ISC computers use plus 9V and minus 9V as the signalling levels.

The interchange standards are the most difficult to describe because of the many options which are possible. The simplest is the send-only interface, sometimes known as "send and pray". A simple send/receive interface can be constructed relatively easily, and a send/receive (or simply send) interface with appropriate handshake can also be constructed without much trouble. The standard includes many features such as

signalling on a secondary channel, ring detection (for dialup connections), and data rate selection. These are fairly uncommon, though, so we'll not discuss their use. Let's look instead at the signals used in common interfaces that you may apply to your computer, and then actually put them to use.

Pin #1 (**FG**) is a common **Frame Ground** electrically connecting the frames of the two machines being connected. It can sometimes be omitted, especially if, as in ISC machines, frame ground and signal ground (see later) are connected. When connecting to another piece of equipment (e.g. modem or printer), check the outlets into which each is plugged to make sure the polarity is identical. If the electrician made a mistake, it could be a costly one for your machine. Sears sells an inexpensive tester for this purpose. (Cat. # 49662)

Pin #2 (**TD**) is for **Transmitted Data**, meaning data going out of the interface (or, in the case of a modem, into the interface). The transmitted data circuit at one end must be connected to the received data circuit at the other. This reversal, sometimes accomplished by the modem and sometimes by the cable, is frequently the cause of initial operational difficulties when connecting gear.

Pin #3 (**RD**) is **Received Data**, data coming into the interface.

Pin #7 (**SG**) is **Signal Ground** and is the reference point for positive and negative signal determination.

Some other signals used for handshaking occur in pairs, such as:

DataSet Ready and **Data Terminal Ready** (pins 6 (**DSR**) and 20 (**DTR**) respectively). It's not difficult to imagine the use of these. They tell each connected machine that the other is up.

Request To Send (pin 4, **RTS**) and **Clear To Send** (pin 5, **CTS**) are a pair of signals used to request and grant permission to transmit data. In the conventional terminal/modem arrangement, CTS is the modem's response which enables data flow from the terminal.

These eight signals are the most commonly used signals of the 25 (3

unassigned) defined by the standard.

When a sender can send data faster than the receiver can process it, there needs to be some interchange protocol established so that the sender does not swamp the receiver with data and cause data loss (overrun). Several different conventions have been established over the last decade and one of these is implemented using the RS232 signals. It is used by most of the popular personal computer printers, such as the IDS 440-450-460 lines, the Epson, Heath, Base-2 and others. The method is to lower the CTS (or sometimes DTR) line when the receiver does not want to receive any more data. (The ability to sense this signal was omitted from early Compucolors, and is the subject of an engineering change known as ECN002137, the "handshake mod").

Now let's take a look at what is required to connect an ISC computer to an Epson MX80 printer and a Cat Novation modem, two of the most popular and useful peripherals to be found.

The first step is the construction of a cable that will connect the RS232 port to the devices. Figure 2 shows the connections needed for each. Notice that the signals for each are different. It would seem at first that two cables will be needed, one for the printer, and one for the modem. With a little cleverness, though, we can make one cable in the form of a Y, with your computer on the bottom and the printer and modem each on its own arm. It'll be necessary to plug and unplug devices, since both should not be powered on and connected to the Y-cable simultaneously. Fancy switching arrangements are an alternative, as are separate cables, but the Y-cable works, and it's cheap. If you follow the construction diagram (Figure 2) you should have no trouble, but some hints are in order:

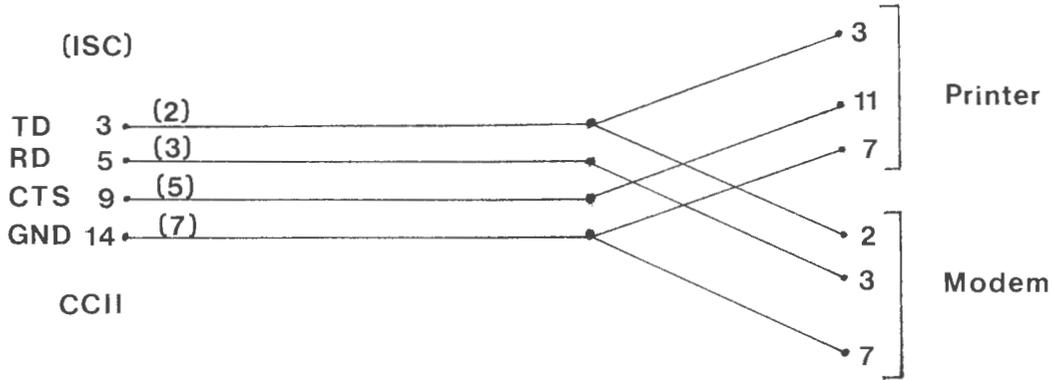
If you use an IDC-type (insulation displacement connector) at the PC board end, press it solidly and evenly together in a vise using even pressure and care. Don't plan to take it apart again!

If you use a solder-type connector at the PC board end, insulate the pin by slipping a length of heatshrink tubing over the wires before soldering. Any 20-24 gauge 4 conductor stranded wire can be used. Belden 8723 is excellent, but can be obtained only in 500 ft. spools.

Solder-type DB25 connectors are the easiest to use.

Use lengths of heatshrink tubing on the Y connections and on the Y itself.

Label the connectors on the legs as "printer" and "modem".



"Y" Cable

Fig. 2

The popular and versatile Epson MX80 printer has recently received rave reviews, and its connection to your computer is not difficult, and will serve as an illustration for the use of other printers. The Epson must be ordered with the RS232/current loop interface (Cat. no. 8141). Follow the instructions for installing the interface, and when the case is open, make the following jumper and switch settings on the interface board:

JNOR - cut and remove.

JREV - solder (carefully) a jumper of 28 or 30 gauge wire.

SW 1-8 - all off.

The printer is now set up to receive data at 9600 BPS, its maximum speed, and respond to potential overflow conditions by raising the clear to send line to your computer.

If you have a CompuColor with a Rev. 3 or earlier computer/logic board, you'll need to add ECN002137. Your dealer can do this for you, or by following the logic in Figure 3, you can make the modification yourself. (If sufficient interest develops among older CompuColorists,



TITLE CCL1 LOGIC PCB ASSY.		DWG NO 100462		REV 4		ECN 002137									
TYPE OF CHANGE		<input type="checkbox"/> DESIGN DEFICIENCY (MANDATORY)		<input checked="" type="checkbox"/> PRODUCT IMPROVEMENT		SHEET 1 OF 1									
		<input type="checkbox"/> DOCUMENTATION CORRECTION ONLY		<input type="checkbox"/> OTHER		PARTS DISPOSITION									
REASON FOR CHANGE						IN PROCESS									
PROVIDE PROPER CLEAR-TO-SEND LOGIC LEVELS FROM MODEM PORT TO LOGIC PCB.						<input type="checkbox"/> REWORK									
DESCRIPTION OF CHANGE						<input checked="" type="checkbox"/> USE AS IS									
1) TIE PIN 9 OF 'J2' EDGE CONNECTOR TO 'UD1' PIN 4.						<input type="checkbox"/> SCRAP									
2) TIE PIN 6 OF 'UD1' TO PIN 3 OF 'UC1'.						EFFECTIVITY									
3) TIE PIN 4 OF 'UC1' TO PIN 10 OF 'UE1'.						NEXT ARTWORK*									
4) ADD 10K 1/4 WATT RESISTOR (ISC#200036) BETWEEN PIN 4 OF 'UD1' AND +12 VOLTS						ECN 2104 IS VOID									
REWORK EXISTING BOARDS AS REQUIRED FOR PRINTER HANDSHAKE OR MODEM 'CTS'.															
ADD RESISTOR BETWEEN PIN 4 OF 'UD1' AND +12 VOLT DC SOURCE.															
NOTE: COMPUCOLOR SOFTWARE 'S1OUT' ROUTINE SAMPLES BIT 7 OF KEYBOARD 'J2' INPUT. 'S1OUT' IS CALLED BY SETTING 14 ₁₀ IN OUTPUT FLAG LOCATIONS.															
EXAMPLE: IN BASIC- POKE 33265,14 THIS SENDS BASIC TO RS232 PORT.															
POKE 33249,14 THIS SENDS FCS TO THE RS232 PORT.															
						<table border="1"> <thead> <tr> <th colspan="2">IC TABLE</th> </tr> </thead> <tbody> <tr> <td>UE1</td> <td>- 74LS157</td> </tr> <tr> <td>UC1</td> <td>- 74LS04</td> </tr> <tr> <td>UD1</td> <td>- MCL489</td> </tr> </tbody> </table>		IC TABLE		UE1	- 74LS157	UC1	- 74LS04	UD1	- MCL489
IC TABLE															
UE1	- 74LS157														
UC1	- 74LS04														
UD1	- MCL489														

ISC 102-1178

Fig. 3

the editorial staff will develop a step-by-step installation sequence and kit. - ed.) Connect the "printer" connector of the Y-cable to the MX80 and turn on the power. There are several ways to direct BASIC output to the RS232 port, to the printer (or modem, when you connect it):

1. Enter BASIC with ESC M rather than ESC E.
2. Execute a PLOT 27,13 statement.
3. Execute a POKE x,14 where x = 33265 (for V6.78 and V8.79) or x = 40886 (for V5.79).

Return BASIC output to the screen by executing a POKE x,0 statement, where x is as above.

FCS output may also be directed to the RS232 port (handy for printing directories) by entering FCS by the ESC G sequence rather than ESC D. Output can be returned to the screen by typing either:

1. CPU RESET, or
2. executing POKE y,0 where y = 33249 (V6.78, V8.79) or y = 40884 (V5.79).

With the MX80 set as described above, no other controls or options need be set (the computer defaults to 9600 BPS). Other printers may require speed and stop bit selection. Since the modem connection requires this, let's take a look now. As with directing output, the speed, stop bits, and half/full duplex modes can be set either with keystroke sequences or by executing BASIC statements.

SPEED

Keystrokes:

ESC R s	where s = 1 for 110 BPS	4 for 1200 BPS
	2 150	5 2400
	3 300	6 4800
		7 9600.

BASIC:

PLOT 27,18,s where s is as described above.

STOP BITS

Keystroke:

A7 ON (one stop bit)
A7 OFF (two stop bits).

BASIC:

PLOT 14 (one stop bit)
PLOT 15 (two stop bits).

HALF/FULL DUPLEX

Keystrokes:

ESC H (half duplex - echo each character locally
as well transmitting it)
ESC F (full duplex - just send it).

BASIC:

PLOT 27,8 (half)

PLOT 27,6 (full).

Connecting the Cat Novation modem is simple. Disconnect the printer and plug the "modem" connector of the Y-cable into the modem. Select speed and stop bits with appropriate commands, direct the output to the RS232 port and dial your favorite timesharing system, information utility, or friend. The command sequence to use is:

1. CPU RESET
2. A7 ON or A7 OFF
3. ESC R speed
4. ESC F or H
5. ESC M.

Two of the more popular information utilities to which you may subscribe are:

MICRONET

Personal Computing Division
Compuserve Inc.
5000 Arlington Centre Blvd.
Columbus, Ohio 43220
614-457-8600

The SOURCE

Telecomputing Corporation of America
1616 Anderson Rd.
McLean, Virginia 22102
703-821-6660

Unfortunately, there may be more to using the port than is indicated in this article. Your printer may cause lower case y's or slashes to print on the screen or printer, and when connected to a remote timesharing system, the distant computer may send control sequences that will cause your disk to spin, your screen to clear, or strange characters to be displayed. These problems can be very annoying, but don't despair--there are cures. The lower case y problem can be easily cured by software or hardware. (See the following article). To solve the remote data/local interpretation problem, a program such as ISC's (or other vendor's) is needed to act as a data filter and translator for the incoming data. (A past issue of Colorcue contained a simple data filter program. For a reprint, send a SASE to our editorial offices requesting it. -ed.)

Lower Case y's

by Tom Devlin
3809 Airport Rd.
Waterford, MI 48095

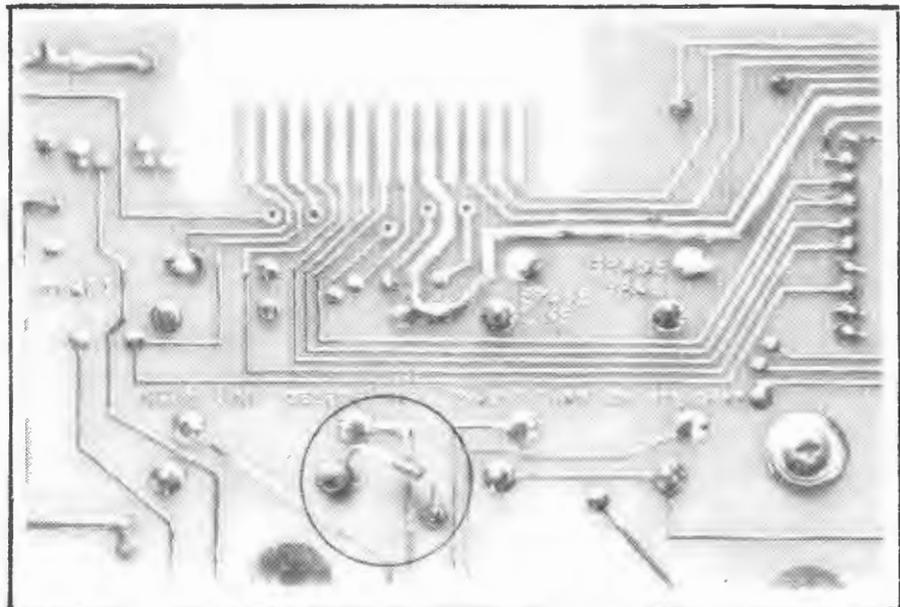
The following fix for the lower case y problem is the simplest, cheapest one I've found to date. It requires the use of the handshake mod (described in the previous article - ed.) and the addition of a single 2 cent diode to the keyboard printed circuit board.

Basically, the fix takes advantage of the fact that the output of UC1 which you connect to pin 10 of UE1 for the handshake mod also goes to R6 in the keyboard via pin 11 of the keyboard cable. This gives us the CTS signal at TTL levels. Adding a diode from this point to any of the column outputs will cause the CCII to think that several keys are being pressed whenever CTS is low, causing the keyboard scan routine to give up and return without a character for that scan.

Following the placement of the diode as shown in the accompanying photograph, you should have no trouble adding the fix to your machine. Use a 1N914 diode, and make sure the band, i.e. the cathode, is toward the right, as shown in the photo.

(Note: Although this fix works with no apparent side effects, it has not been sanctioned by ISC, and may affect your warranty.- ed.)

Photo 1.
Trace side of keyboard PCB. Solder a signal diode as shown, cathode end to right.



Screen Dump to the MX80 Printer

by Mark D. Fairbrother
Carriage House East, A5
Rt. 11
Kirkwood, NY 13795

This program will produce a hardcopy of the 128 by 128 resolution ISC graphics screen on an Epson MX-80 printer. Only those character positions that have the plot bit set in their color byte will be printed; all others will be treated as blanks. The reason I chose not to print other characters is due to the difference in size of an ISC plot block (4 by 2) versus an MX-80 plot block (3 by 2). In order to faithfully reproduce the ISC graphics, the screen is scanned three Y positions and two X positions at a time to form an MX-80 graphic character. Since a character such as an "A" occupies what looks to the printer like one and a third lines, to avoid gaps it is not printed.

The program consists of three sections: initialization, dot scan/character build/print, and point test. The following is a breakdown of the program, by section and line.

INITIALIZATION

0 and 1 Set up the array MS, which is used by the point test subroutine. These lines must be executed before the dump subroutine is called.

DOT SCAN/CHARACTER BUILD/PRINT

9000 Disable keyboard.

9010 Set up for stop bit (PLOT 14 . . 15), 1200 Baud (PLOT 27,18,4), and send all output to the RS-232 port (PLOT 27,13). Set print width to 132 characters per line. The PLOT 14 at line 9030 sets double width characters, which change the line width to 66. The PLOT 14 must be reissued for every line output.

9020-9040 Scan the CRT Y positions in increments of three and X in increments of two. This corresponds with the number of dots per MX-80 block.

9050-9200 CH is the character to be sent to the printer and is built

one dot at a time. The characters are located from CHR\$(160) to CHR\$(223), or all dots off to all dots on. On return from the point test routine, PT will be set to -1 to skip two X points, 0 if that X,Y point is off, or >0 if that X,Y point is on. The printing process will take 20 to 25 seconds per printer line (around 15 minutes for a whole screen) depending on the number of blanks. The code at line 9060 is to handle the scanning at the next to last lines.

9210 Move to top of the next sheet of paper.
9220 Return output to screen.
9230-9240 Enable keyboard interrupt and return.

POINT TEST - TEST POINT AT SX,SY

9510 Initialize PT.
9520 Compute the address within the screen buffer of the character the dot specified by SX, SY is within.
9530 Get the ASCII code and color byte of the character.
9540 If it is not a blank and is a graphic character, continue testing.
9550-9560 Return PT = -1 if two X tests can be skipped as being unprintable (each character is two points wide).
9570 Convert SX and SY to an index into MS.
9580 See if point is on. MS and DO are used to isolate a particular bit in the byte stored for the graphic character. (See your Programming Manual for the relationship between a graphic point and the byte stored in memory.) PT will be zero if the point is off and non-zero if the point is on.

As mentioned above, it will take around 15 minutes for a complete picture to print. Translation of the algorithm into machine language will undoubtedly speed it up. (Any takers? - ed.)

To use the subroutine, simply bury it in a program that draws a picture, and GOSUB 9000 to start it. If you do not have the y-eliminator, put some sort of delay in before calling the subroutine to allow time to set up the printer and turn it on. **■**

(Listing follows.)

Listing 1. The screen dump subroutine

```

0 DIM MS(7)          9150 NEXT J
1 FOR I= 0 TO 7      9160 NEXT I
  :MS(I)= INT (2^ I) 9170 PRINT CHR$( CH);
  :NEXT I           9180 NEXT X
9000 OUT 8,4         9190 PRINT CHR$( 13)
9010 PLOT 14,27,18,4,15,27,13,15 9200 NEXT Y
9020 FOR Y= 0 TO 126 STEP 3 9210 PRINT CHR$( 12)
9030 PLOT 14         :REM OUTPUT A FORMFEED
9040 FOR X= 0 TO 126 STEP 2 9220 POKE 33265,0
9050 CH= 160         9230 OUT 8,255
  :M= 1             9240 RETURN
  :IM= 2           9500 REM SUBROUTINE TO SEE IF POINT SX,SY IS ON. PT=0 IF NOT.
9060 IF Y= 126 THEN IM= 1 9510 PT= 0
9070 FOR I= 0 TO IM 9520 AD= 28672+ 2* INT (SX/ 2)+ 128* INT (SY/ 4)
9080 FOR J= 0 TO 1   9530 DA= PEEK (AD)
9090 SX= X+ J       :CL= PEEK (AD+ 1)
9100 SY= Y+ I       9540 IF DA< > 32AND CL> 127GOTO 9570
9110 GOSUB 9500     9550 PT= - 1
9120 IF PT= - 1 THEN M= M* 4 9560 RETURN
  :GOTO 9160       9570 DO= 4* (SX AND 1)+ (SY AND 3)
9130 IF PT> 0 THEN CH= CH+ M 9580 PT= MS(DO) AND DA
9140 M= M* 2       9590 RETURN

```

LATE NEWS: Compuworld Inc. has announced business software packages for Intecolor computers. Programs include ColorCalc, ColorGraph, General Ledger, and others. (See the August, 1981 issue of Mini-Micro Systems.) We'll have more on these next issue. In the meantime, for more information contact Fred Calev, Compuworld, 125 White Spruce Blvd., Rochester, NY 14623. (716) 424-6260.

ALIEN INVASION

V1.0 (no sound) (US funds)

Copyright (c) 1981 by David B. Suits

Fast, machine language invaders arcade game with color graphics and four levels of difficulty from "not-too-difficult" to "what-the-?!#*??" . Runs on V6.78 and V8.79 software with standard keyboard.

Also included: fast, machine language version of LIFE.

Special bonus: V2.0 of Alien Invasion (if I ever get around to writing it; I'm basically lazy, so I'm making no promises) will have sound effects if you have Cap Electronics Soundware or an equivalent device, and it will be free to all purchasers of V1.0.

David B. Suits
49 Karenlee Dr.
Rochester, NY 14618

User Group Bulletin Board

The following list represents our most current information on Intecolor/Compucolor User Groups. In the interest of keeping our readers informed and in touch, we plan to run this list periodically. If you have additions (because your group is not listed) or changes (because names or numbers are different), drop a line to our editors so we can tell the world. We'd like to include news of your group here, too. Have your secretary write a piece about your group, its activities, interests, and members. We also welcome pictures. (5 x 7 black and white glossies, with good contrast.)

User groups are a very important medium for information transfer, and we at Colorcue want to provide a platform for group information interchange. Get your secretaries on the ball!

ALABAMA

Compucolor Users Group
Eike Mueller
12117 Comanche Trail, S.E.
Huntsville, Alabama 35803
(205) 883-7614

San Jose User Group
Vicki Oliver
1358 Branham Lane #4
San Jose, California 95118
(408) 267-5250
The SOURCE: CL0691

CALIFORNIA

Compucolor/Intecolor User Group
Stan Pro
S.P. Electronics Systems
5250 Van Nuys Boulevard
Van Nuys, California 91401
(213) 788-8850

El Cerrito User Group
Frasier Hewitt
c/o P.C. Computers
10166 San Pablo Avenue
El Cerrito, California 94530
(415) 428-0468 (F. Hewitt)

GOTO Group
Tommy W. Schenck
c/o Tom & Bobbie of Newberrys
1136 Fulton Mall
Fresno, California 93721

San Diego User Group
Hal Brehe
4671 Mt. Arnet Drive
San Diego, California 92117

FLORIDA

JACKS - Jacksonville Area
Compucolor Knowledge
Seekers
Gary Haney
1723 Debbie Lane
Orange Park, Florida 32073
(904) 264-6785

Robinson High Computer Club
Mrs. Byman
6311 S. Lois Avenue
Tampa, Florida 33616
(813) 835-1211

GEORGIA

Compucolor Users Group
Irv Mullins
2194 Briarcliff Road, N.E.
Atlanta, Georgia 30329
Day: (404) 586-5156
Night: (404) 634-3919

MASSACHUSETTS

Compucolor Users Group
 Richard Manazir
 13 Grandview Street
 Southwick, Mass. 01077
 Day: (203) 688-1911
 Night: (413) 569-6621

NEW JERSEY

Compucolor Users Group
 Peter J. Miller
 125 Buena Vista Drive
 Ringwood, New Jersey 07456
 Night: (201) 839-7251

NEW YORK

Compucolor Users' Group of Rochester
 Gene Bailey
 28 Dogwood Glen
 Rochester, New York 14625
 (716) 381-4027

OREGON

Compucolor II Users' Group
 Bruce Vanderzanden
 2006 "C" Street
 Forest Grove, Oregon 97116
 (503) 357-2772

PENNSYLVANIA

Philadelphia CCII Users Group
 Howard Rosen
 P.O. Box 434
 Huntington Valley, Pa. 19006
 (215) 464-7145

VIRGINIA

Compucolor Users Group
 Rick Vick
 702 W. Holly Avenue
 Sterling, Virginia 22170
 Day: (703) 827-3894
 Night: (703) 430-3843

INTERNATIONAL

Ham Radio Users Group
 Bill Shanks, W2GTX
 7 Lake Circle Drive
 Vicksburg, Mississippi 39180

AUSTRALIA

Compucolor User Group/Sydney Area
 Andrew McIntosh
 91 Regent Street
 Chippendale, N.S.W., 2008
 Australia

Compucolor User Group/Melbourne Area
 Neil Brandie
 212 High Street
 Windsor, Victoria, 3181
 Australia

CANADA

Canadian Users Group
 Glen Davis
 Bsmt-59 Kendal Avenue
 Toronto, Ontario
 Canada M5R 1L8

Compucolor Users Group
 Mark Herzog
 House of Computers
 368 Eglinton Avenue, West
 Toronto, Ontario
 Canada M5N 1A2
 (416) 482-4336 **C**

Cueties . . .

```

10 REM COLORED SQUARES
20 REM BY STEVE REUBART
30 PLOT 27,24,15,29,6,0,12
40 IN=1:C=1
50 FOR Y=0 TO 31 STEP 4
60 FOR X=0 TO 63 STEP 8
70 C=C+IN:PLOT 6,C
80 FOR I=Y TO Y+3:PLOT 3,X,I
90 PRINT "``````````":REM 8 NULLS
100 NEXT I,X,Y
110 FOR W=1 TO 600:NEXT
120 IF IN=1 THEN IN=-1:C=64:
    GOTO 50
130 PLOT 27,11,6,2,8
  
```

Assembly Language Programming

by David B. Suits

The Intecolor and Compucolor computers have a great deal to offer in the way of color graphics. Using BASIC to access this power is relatively easy, but its main disadvantage is speed. Assembly language programming provides the ultimate in speed, but its main disadvantage is coding time. Another disadvantage is instruction: what can you do with assembly language if you don't know anything about it? Where can you go to learn about it?

Answer: here! I sympathize with all those of you who want to learn assembly language but are afraid to begin, or don't know where to begin. This series of articles will be devoted to you, the rank beginner. I will assume you know a little bit about programming your Intecolor or Compucolor in BASIC. Other than that, all you need is a little curiosity, and some perseverance. (And a subscription to Colorcue.)

Inside your computer's plastic shell there is an 8080A microprocessor (call it "8080" for short), connected to some Read Only Memory (ROM) and some read/write memory, also called Random Access Memory (RAM). In addition, the 8080 has a tiny bit of read/write memory inside itself, referred to as the 8080's **registers**. Each of these registers is eight binary digits (one byte)--except for the Stack Pointer (SP) and the Program Counter (PC), which are 16 bits (two bytes) each. Certain of the eight bit registers may be grouped together into pairs: registers D and E can be paired and referred to as register pair D (some people prefer to be explicit and say DE). H and L can be referred to as register pair H (or HL), and so on. For the most part you will be concerned to read from or write to registers A, B, C, D, E, H and L, and, often, the register pairs BC, DE and HL. PC, SP and FLAGS are used automatically

REGISTER		REGISTER PAIR NAME
PC		PC (Program Counter)
SP		SP (Stack Pointer)
FLAGS	A	PSW (Program Status Word)
B	C	B
D	E	D
H	L	H

ONE BYTE ONE BYTE

by the 8080 for its own purposes, and we won't bother with them for quite a while, except for the FLAGS, which we'll make use of frequently.

The 8080 microprocessor is capable of understanding and carrying out a varied set of instructions, each of which is simply a number (eight bit binary) held somewhere in the computer's memory. The 8080 will look at memory, copy the number it finds there into a secret internal register of its own, and execute the instruction. This means that the 8080 must know where in memory to find that instruction number in the first place. That's the function of the Program Counter (PC), which points to a specified memory location. When the information (always an eight bit binary number) is read into the 8080, the PC is automatically incremented to point to the next instruction in memory: that is, it is automatically set up to get the next instruction as soon as the present instruction has been carried out. More or less as in BASIC, the 8080 will execute an instruction and then move on to the next one, execute it, move on to the next one, and so on. And, more or less as in BASIC, you can use GOTOs and GOSUBs to veer off to other parts of the program, except that the 8080 will jump hither and yon according to JMP and CALL instructions. More accurately, since the 8080 can understand only binary numbers, and not letters, it will GOTO (JMP) somewhere or other in response to the binary number 11000011 (=195 decimal), and GOSUB (CALL) some subroutine or other in response to binary 11001101 (=decimal 205). Since we humans have difficulty in talking in long strings of 1s and 0s, we write programs using mnemonic names like JMP and CALL, which are then translated into binary numbers by a program called an **assembler**. So there are two steps to assembly language programming: (1) You write the program in assembly language, perhaps using ISC's Screen Editor (or, Heaven forbid, the old Text Editor), and save it on disk. (2) You run the assembler (or the macro-assembler), which translates your assembly language program into a series of binary numbers and stores the result (we can now call it **machine language**, since it is in a form which the 8080 can understand) on disk. You will need, then, to learn how to write an assembly language program, and then you will have to learn how to use the assembler program to translate what you've written. We'll cover both of those topics (and more) in this series.

A crucial difference between BASIC (and other high level languages) and assembly language is that in assembly language you do everything one tiny, painstaking step at a time. Most single instructions in BASIC would require a large number of assembly language instructions. Whereas in BASIC you could write **A=B+C**, in assembly language you would have to

break it down into a series of minimal steps:

- (1) Find the location in memory called "B".
- (2) Copy the contents of location "B" into an internal register of the 8080.
- (3) Find the location in memory called "C".
- (4) Add the contents of "C" to that internal 8080 register which has been holding the contents of "B".
- (5) Find the location in memory called "A".
- (6) Copy the contents of that 8080 internal register (which now contains the sum of "B" and "C") into that memory location called "A".

Steps (1), (3) and (5) are themselves a series of operations which must be broken down into a number of discrete 8080 instructions. (To make matters more complex, BASIC uses not one, but four locations in memory to store a number.)

Before we get into a discussion of the assembly language instructions and what they mean, let's get accustomed to binary and hexadecimal arithmetic. If you don't already know something of binary and hex, then the following will serve as a brief introduction. The real key to learning is experience. That means practice. And practice. The more you fiddle around, the more you will feel at home. (Isn't this how many of us learned BASIC?)

BINARY NUMBERS

When you add one to a (decimal) number, you get the next higher number. Take the largest number representable by a single decimal character, 9, and add one to it. You get 10. The digits shift just like the wheels in an odometer: once a character reaches 9, it shifts back to 0 and the character to the left moves up one. Each column in a decimal number is occupied by some decimal character. The value of the whole decimal number can be determined by multiplying the value of the character by its column's **weight**. A decimal number has a one's column, a ten's column, a hundred's column, and so on. These are merely the weights each column has.

	weight	1000	100	10	1
example number		0	2	3	8

The decimal number 238 is two one hundreds plus three tens plus eight ones: $(2 \times 100) + (3 \times 10) + (8 \times 1) = 238$. These weights are simply powers of ten

(powers of ten, because this is the decimal number system). The powers start with 0: $10^0=1$. (In fact, any number to the zero power =1.)

	powers	3	2	1	0
decimal weights		$10^3=1000$	$10^2=100$	$10^1=10$	$10^0=1$

We now have all the principles necessary to develop any other number system we please. For the binary number system, just repeat the previous discussion, replacing "decimal" with "binary", the ten allowable decimal characters with the two allowable binary characters (1 and 0), and so on.

A binary odometer would turn considerably ^{fast} slower than the decimal odometer (but there would be no savings in gas):

000	001	010	011	
<u>+ 1</u>	<u>+ 1</u>	<u>+ 1</u>	<u>+ 1</u>	...
001	010	011	100	

As soon as you reach 1, you must start over again, whereas in the decimal system you don't have to start over until you reach the character "9". Don't misinterpret these numbers. The binary number 10 is not equal to the decimal number 10. If you start at zero, you will have counted to two when you get to binary 10, but you will have counted to ten when you get to decimal 10.

Each column of a binary number has a weight, determined by the powers of two:

	powers	3	2	1	0
binary weights		$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$

Thus, a binary number is represented using the same underlying principles as a decimal number. If we wish to translate binary into decimal, we need only multiply the column weights by the column numbers and add them up. For example:

weight					
(expressed					
in decimal)	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$	
example number	0	1	1	0	(binary)

In decimal this would be $(0 \times 8) + (1 \times 4) + (1 \times 2) + (0 \times 1) = 6$. Your computer stores numbers as eight binary digits (one byte), so we ought to become accustomed to this table:

powers
 (expressed
 in decimal) $2^7=128$ $2^6=64$ $2^5=32$ $2^4=16$ $2^3=8$ $2^2=4$ $2^1=2$ $2^0=1$

This means, incidentally, that the largest number which can be stored in a given memory byte is 11111111 (binary) = 255 (decimal). Some of the 8080's internal registers are 16 bits wide; in addition, each memory location in the computer is referenced by its **address** (more on that later), which is 16 bits. So we ought to become familiar with this table:

$2^{15}=32768$ $2^{14}=16384$ $2^{13}=8192$ $2^{12}=4096$ $2^{11}=2048$ $2^{10}=1024$ $2^9=512$ $2^8=256$
 $2^7=128$ $2^6=64$ $2^5=32$ $2^4=16$ $2^3=8$ $2^2=4$ $2^1=2$ $2^0=1$

The largest number which two bytes can hold (and the highest address allowed) is therefore 11111111 11111111 (binary) = 65535 (decimal).

With some "hunting and pecking", these tables will allow you to translate from decimal to binary and also back again. To translate from decimal to binary, find the bit with the highest weight equal to or less than the decimal number to be converted. Make the corresponding bit in the binary number = 1 and subtract its weight from the decimal number. Resume the process with the result of that subtraction until the result is zero. For example, we can translate 238 (decimal) into binary by noticing that the highest binary weight equal to or less than 238 is 128. Make that bit = 1.

... 512 256 128 64 32 16 8 4 2 1
 0 0 1

Subtract 128 from 238 to get 110. The highest binary weight equal to or less than 110 is 64. Set that bit = 1.

... 512 256 128 64 32 16 8 4 2 1
 0 0 1 1

110-64=46. The bit under 32 becomes 1.

... 512 256 128 64 32 16 8 4 2 1
 0 0 1 1 1

46-21=14, so the bit under 8 becomes 1.

... 512 256 128 64 32 16 8 4 2 1
 0 0 1 1 1 0 1

14-8=6. Etc. The final result is 11101110 (binary).

bit	7	6	5	4	3	2	1	0	
	1	1	1	0	1	1	1	0	(binary)

An alternative method is to divide the decimal number by 2 and place the remainder into the right-most bit of the binary number. Now divide the quotient (i.e. the integer result) of the previous division by 2, put the remainder in the next bit position to the left, and so on until the quotient = 0.

119	remainder = 0	(bit 0, i.e., right-most bit)
2)238		
59	remainder = 1	(bit 1)
2)119		
29	remainder = 1	(bit 2)
2)59		
14	remainder = 1	(bit 3)
2)29		
7	remainder = 0	(bit 4)
2)14		
3	remainder = 1	(bit 5)
2)7		
1	remainder = 1	(bit 6)
2)3		
0	remainder = 1	(bit 7)
2)1		

HEXADECIMAL NUMBERS

The problem with binary numbers is that they are so confusing to work with: all those zeros and ones create a fertile environment for errors. Decimal numbers are easier, because there are fewer characters per number and because we're already familiar with them. Unfortunately, the computer is too stupid to deal with ten different characters. We might use decimal numbers for our own figuring and then translate between decimal and binary when required. But that is usually a tedious operation. What we need, then, is a way of expressing binary numbers without being restricted to two digits, and yet without the burden of tedious translation. Enter the hexadecimal system.

The binary system is built upon powers of two (that's 10 in binary). The decimal system is built upon powers of ten (that's 10 in decimal). So the hexadecimal system is built upon powers of 6 (hex) + 10 (dec) = 16 (that's---you guessed it!---10 in hexadecimal). Hexadecimal is convenient because 16 is a power of two (the fourth power). This means

that we can represent four binary digits with one hexadecimal digit. Or, that is, two hex digits will represent one byte, and four hex digits will represent two bytes. That convenient symmetry does not exist between binary and decimal. (Ten is not an integer power of two.) This does two things for us: (1) it decreases the number of characters necessary to represent a number, and (2) it allows a human to convert quickly between binary and hex as the occasion demands.

The conversion is simple: start with the four least significant bits (i.e., the right most four bits, or half a byte--sometimes called a nibble, or nybble) and translate them into its hex equivalent. Move left and convert the next nibble, and so on. To convert from hex to binary, just replace each hex digit with its binary equivalent.

Well, in order to make these conversions, we'll have to define the hexadecimal characters. We need sixteen of them, but we are accustomed to using only ten (0-9). So we invent six more. It really doesn't matter what they are, except that tradition calls for using the letters A-F. (Perhaps because these symbols were easily available on typewriter keyboards.) The relationships between the binary, decimal and hexadecimal schemes are shown below. After working with hex and binary numbers for a while, you will have this table memorized.

<u>HEX</u>	<u>BINARY</u>	<u>DECIMAL</u>
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15
10	00010000	16
11	00010001	17
.	.	.
.	.	.
.	.	.

If you recall the principles which structure all number systems, you should have no trouble dealing with hex. A hex odometer would turn slower than a decimal one:

00	09	0A	0B	0C	0D	0E	0F	10	
<u>+1</u>	...	<u>+1</u>	...						
01	0A	0B	0C	0D	0E	0F	10	11	

I said before that one reason for not using decimal numbers was because the computer was too primitive to handle ten different characters; two is its limit. (After all, the computer is merely a clever arrangement of switches, each of which can be either on, =1, or off, =0.) And now I offer the hexadecimal system with its **sixteen** characters. So? I lied.

No, no. It is true that the computer can't deal directly with anything other than binary numbers, but hexadecimal numbers at least provide a fast and efficient manner in which to represent binary numbers to humans, which is what we are. We can always write a simple routine to translate between hexadecimal and decimal if we need it. Besides, hex is firmly rooted in tradition (short as the computer tradition is), and any attempt to rip it up will probably cause the destruction of the universe.

Your computer's FCS understands only hex. (This makes it difficult for BASIC, which understands only decimal, to talk to its own FCS! Computer manufacturers work in mysterious ways.) Furthermore, ISC's Machine Language Debug Package (a spectacular piece of software, by the way) is easier to deal with in hex (although you can get by with decimal), and we will be using it in this series. (Do you have yours yet?)

Next time: the 8080 instruction set. In the meantime, read anything and everything you can get your hands on dealing with 8080 assembly language. There are articles in magazines, books, and even assembly language program listings--cryptic as they might seem at first--can help you become acquainted with this new language. Every human is partly a self-teacher. You'd be surprised at how much information you can assimilate simply by looking over some 8080 assembly language listings. At least you will come to have an unconscious "feel" for the language. And that's important. **■**

HOWARD ROSEN, INC.

Put the finishing touches to your CompuColor II or ISC computer.

- > Come up to the world of word processing.
- > Extend the utilization of your computer to the other members of your family.
 - * Letters
 - * School reports
 - * Business reports
 - * If you now type-write it, Comp-U-Write it for a better product.
- > Basic requirements for CCII or 3651/9651:
 - 16K RAM.
 - 117key keyboard.
 - Printer.
 - Comp-U-Writer software and instruction manual.
- > For maximum capability:
 - Full 32k RAM.
 - Lower case characters.
- > Talk to other computers: Add a MODEM to your system.

We carry the entire CCII & ISC line of hardware/software, including spares. Send for our 4-page order form for hardware/software. Request separately by item your spare parts needs.

Send your order now. We pay the shipping.
Allow 5 weeks for delivery.

CCII	3650/9650	Description	Quantity	Cost	Amount
010057	010053	Upgrade 72/101 keys	_____	150.00	_____
010058	010054	Upgrade 72/117 keys	_____	250.00	_____
010059	010055	Upgrade 101/117 keys	_____	100.00	_____
010044		24in. RS 232C Cable	_____	45.00	_____
100986	0C2100	16k RAM Add-on	_____	310.00	_____
010051		Switchable Lower Case	_____	150.00	_____
	0C03LC	32 Lower Case Characters	_____	100.00	_____
990001	990030	5in. Formatted Twin Pack	_____	9.95	_____
	900041	8in. - 10 One Side Format	_____	75.00	_____
	900044	8in. - 10 Two Side Format	_____	100.00	_____
991544	991545	5in. Comp-U-Writer	_____	262.50	_____
	991546	8in. Comp-U-Writer	_____	262.50	_____
991509	991532	FORTRAN	_____	75.00	_____
	Centronics 737-3	Correspondence printer	_____	850.00	_____
	Base-2, Inc. 850	Impact printer	_____	750.00	_____
	C. Itoh	Daisy wheel printer	_____	2100.00	_____
	CAT Novation	MODEM Transmit/Orginate	_____	175.00	_____

	Sub Total	_____
Pa. residents add 6% Pa. Sales Tax	Pa. Tax	_____
	Total	_____

Terms - Cash with order

Name _____ Telephone # (_____) _____

Address _____ City _____ St _____ ZIP _____

HOWARD ROSEN, INC.
PO Box 434
Huntingdon Valley, Pa. 19006

Signature (please sign order)

BULK RATE
U.S. POSTAGE
PAID

Rochester, N. Y.
Permit No. 415

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

An  **Publication**

Colorcue

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

October/November 1981
Volume 4, Number 2

Editors:
Ben Barlow
David B. Suits

- 3 **Editors' Notes**
 - 4 **ISC Product Review**
 - 6 **Assembly Language Programming, by David B. Suits**
Part II: Using the Machine Language Debug Package
 - 15 **Sphere Program, by Mark D. Fairbrother**
Generate a 3-D sphere on your screen
 - 17 **Incremental Plot Table, by Bob V. Smith**
Find those incremental plot numbers the easy way
 - 18 **Lissajous Figures, by Trevor Taylor**
Mimicking oscilloscope graphics
 - 20 **Ask Mr. C. O. Lorcue**
 - 21 **CALLable Sort Routine, by Alan D. Matzger**
Ultra fast sorting for your BASIC programs
-

Advertisers: You will find our advertising policies attractive. Write for details.

Authors: This is a user-oriented and supported publication. Your articles/tips/hints are required to make it go. Send your articles or write for information.

Colorcue is published bi-monthly by Intelligent Systems Corporation, with editorial offices in Rochester, New York. Editorial and subscription correspondence should be addressed to the Editors, Colorcue, 161 Brookside Dr., Rochester, NY 14618. Product related correspondence should be addressed to ISC, 225 Technology Park, Norcross, GA 30092, ATTN: Susan Sheridan. Opinions expressed in by-line articles are not necessarily those of the editors or of ISC. Hardware/software items are checked to the best of our abilities but are NOT guaranteed.

Editors' Notes . . .

The last issue of **Colorcuc** carried an article by Tom Devlin about an elegant, inexpensive solution to the lower case y problem that has bugged CompuColor owners for so long. As it frequently happens, inventors seem to come up with nearly the same ideas at nearly the same time. Ken Orford, a member of **Forum**, the Canadian users group, hit upon the same fix for the problem several months before Tom and published it in the July/August issue of the **Forum** newsletter. Since by that time we had already received Tom's article, we ran it in our August/September issue. But we neglected to mention Ken's article, so we want to take this opportunity to acknowledge Ken's efforts and **Forum** magazine. It is a fine newsletter published six times per year. Subscription is \$15. By the way, our User Group Bulletin Board in the last issue of **Colorcuc** was out of date for Canada. **Forum** is the only user group. For information, write to **Forum**, 21 Dersingham Crescent, Thornhill, Ontario, Canada L3T 4P5.

DEBUG BUG DEBUGGED - Dave Suits

There is a fatal bug in at least some versions of the Machine Language Debug Package program. In order to see if your version has a bug, and to correct it if it does, read on.

The MLDP disk comes with two versions of the Debug program. On the disk they are named MLDP.PRG;1 and MLDP.PRG;2. Only version one has the bug. If you've a 32K machine, you probably won't notice it. (Besides, you'll usually be

using version two.) But if you have a 16K machine, you must use version one (since version two loads into very high memory--which would be non-existent on a 16K machine). Enter FCS with **ESC D** and then run MLDP;1. You should be greeted with a heading and the **DBG>** prompt. Type in this: **@A03A** and hit RETURN. There should be a yellow **A03A** on the left. (If it's not there, try again.) Off to the right, and in white, you will see **H,0E000H**. If that's **not** there (for example, if you have **H,0A000H**), then your MLDP program is OK. But if it is there, then so is the bug. To correct it, type in: **LXI H,A000** and hit RETURN. You should now be shown, on the left, a yellow **A03D**. (If that's not there, go back to the first step by typing **@A03A** and try again.)

You have now altered the MLDP program, and you should save this corrected version on disk. To do that, first hit CPU RESET and ESC D. If you have single density disks (i.e. CompuColors) then you'll have to mount a fresh disk: there's no room on the original disk to save the corrected program. Now type: **SAVE MLDP A000-BF00**. This new version of MLDP will now run correctly on both 16K and 32K machines.

The bug, in case you're interested, was the program's initializing the pseudo stack pointer to E000H. That will work with 32K machines, but on 16K machines, E000H is a non-existent RAM address. The alternative to the permanent correction is to correct the error each time the MLDP program is restarted (either run from FCS or else entered with (ESC) (USER)). In this case, you will have to re-initialize the pseudo stack pointer by typing **%SP=A000**.

EPSON NOTES - Ben Barlow

Those of you with Epson printers have no doubt seen the advertisements for GRAFTRAX 80, the bit graphics package for the MX-80. I got mine in the mail the other day, and what I read sent me scurrying for the Aug./Sept. issue of **Colorcue** in which I had written: "The Epson must be ordered with the RS/232 interface (Cat. no. 8141)". Then back to the front page of the GRAFTRAX 80 documentation: "Note: GRAFTRAX 80 will not function with the Epson 8141 serial interface". Curses! Not only can I not try GRAFTRAX 80, since I've got an 8141 interface, but I've possibly misled some readers into buying into the same dilemma.

If so, I offer apologies. If not, I can think of two solutions. One, purchase the 8150 version serial interface port (it costs about twice the 8141), or, two, build a serial to parallel interface and connect it to the MX-80's parallel connector. Always looking for the cheap solution, I'm opting for the second method. I hope to

have an article about the parallel interface and GRAFTRAX 80 in the next **Colorcue**.

Having heard rumors of printer malfunctions when using GRAFTRAX-80, and seeing a prominent warranty disclaimer in the documentation, I called Epson America to inquire. I was told that there were no problems using GRAFTRAX-80. It can cause no problems with the hardware because it is simply a replacement program for the printer. The warranty disclaimer is there to protect against claims due to customer mis-installation or claims arising from applications of the printer (for example, a bug in the program using a printer connected to an EKG machine, causing the death of three patients). I was advised of a problem with a kit for a friction feed roller to add to the MX-80. The kit apparently causes the stepper motor, which advances the paper, to burn out. Its installation will void the Epson warranty. (Epson has no modification or plans for one to add friction feed. They advise the use of the MX-80FT printer.) **■**

INTELLIGENT SYSTEMS CORPORATION PRODUCT LINE REVIEW

Intelligent Systems designs, manufactures, markets and supports several different types of color graphic microcomputers. These products are used in varied applications such as process control, energy management, and network monitoring. There are tens of thousands of units in use in over 40 countries.

Currently, the product line consists of three different computer types, each of which can be ordered with various options such as different screen sizes, different memory amounts, printers, software, etc.

The "workhorse" of the Intecolor product line is the 8000 series. This series includes terminals, desktops, and complete computer systems. These

computers are in industrial cabinets and have card cages for adding features as required. All prices quoted below are for single quantity.

8001G terminal \$2560

Includes refresh RAM, terminal control software, 72-key keyboard, 19" CRT, RS-232C, 8 foreground 8 background colors, graphics software and editing features. 160 x 192 resolution.

8001I terminal \$4460

Same as 8001G, but with graphics resolution of 480 x 384. Each dot on the screen is addressable to any of eight colors. Includes dot, arc, vector, and incremental vector generators. 4 by 4 dot super-pixel yields 4913 individual lines.

8051 desktop \$4265

Complete desktop computer with color graphics, 19" CRT, Disk BASIC, lower case ASCII, operating system, 48 lines by 80 characters, 80K mini-disk and 8K of user RAM.

When the setting requires a contemporary cabinet, Intelligent Systems has the 8300 and 8900 product series, with 13" and 19" CRTs respectively. These fit well in business environments, where they are used for MIS, communications, and word processing. Each of the following pairs has features comparable to the corresponding industrial units described above.

8301G \$3560

8901G terminal \$2760

8301I terminal \$5460

8901I terminal \$4660

8351 desktop \$5265

8951 desktop \$4465

Many users do not require the high density of information that can be displayed in the 80 character by 48 line format of the 8000 series. For these users, Intelligent Systems manufactures a line of single-board computers whose less dense (and less expensive) displays show 64 characters by 32 lines. These computers have fewer available options than the 8000 series products, but their low price makes them very attractive. They are

(Continued on page 13.)

Assembly Language Programming

PART II: USING THE MACHINE LANGUAGE DEBUG PACKAGE by David B. Suits

I mentioned last time that instructions to the 8080 are always eight binary digits (one byte). This is misleading. With one byte there are 256 possible numbers (0-255), but the 8080 does not understand 256 different instructions. The number 237, for example, is undefined: what the 8080 will do, if anything, with that number is unpredictable.

There is another sense in which it is misleading to say that the 8080 instructions are one byte numbers. Sometimes (I should say "often") the 8080 will know **what** to do, in response to an instruction code, but it will have to be given certain **data** to do it with, or else it will have to be told where to get that data. Let me explain.

In response to the number 71 (=01000111 binary) the 8080 will copy the contents of the A register (the Accumulator) into the B register. That's simple and straightforward. The 8080 doesn't care what is in either register. It just blindly copies A into B. (Consequently, whatever used to be in register B will no longer exist.) Suppose A contains 0001100 and B contains 11010110.

Before executing 01000111	After executing 01000111
A 00011100	A 00011100
B 11010110	B 00011100

For each possible register copy instruction (not counting PC, SP and FLAGS registers) there is some unique number which tells the 8080 to do that copying. In assembly language you specify the copy instruction by using some mnemonic. The standard set of mnemonics was defined by Intel Corporation (the inventors of the 8080) some years ago. Unfortunately, instead of something obvious, like **COPY A TO B**, they chose the more cryptic **MOV B,A**. 'MOV' stands for 'move', but what it means is simply copy. After the **MOV**, the destination register is specified, and then the source register. The general format for a **MOV** instruction is therefore **MOV <dst. reg.>,<src. reg.>**. Don't forget the comma between the two registers' names. The assembler translates such MOV instructions into their appropriate binary codes.

Some other 8080 instructions, however, need more than one byte to

operate. For example, the number 00001110 (=0E hex) instructs the 8080 to put the following number into the C register. But what is "the following number"? It is the byte which immediately follows the 00001110 itself. Hence, this is called an **immediate** type of instruction.

```
00001110 00000001  put 1 into the C register.  
00001110 01000001  put 65 into the C register.
```

And so on. In assembly language, this is a **move immediate** instruction, whose mnemonic is **MVI**. The general format is **MVI <reg.>,<number>**. Some examples:

```
MVI A,0    put 0 into the accumulator  
MVI C,25   put 25 into the C register  
MVI L,15   put 15 into the L register
```

Note that the number to be placed into a register must be between 0 and 255 -- i.e., one byte. The instruction **MVI A,1024** is an impossible instruction (and your assembler will give you an error message), since the number 1024 is 00000100 00000000 binary and occupies two bytes, not one. (By the way, negative numbers are allowed, but that's a topic for a future installment.)

Register pairs may also be loaded with immediate data (two bytes, this time), but we'll save discussion of that for another time.

Using the MLDP

ISC's Machine Language Debug Package is a marvelous tool for debugging machine language programs. It is not an assembler, although it can do some of the same things that an assembler will do. And it can be used by us to learn about 8080 instructions and what they can be made to do.

Before we jump in for some hands-on experience, please read the section in this issue's **Editors' Notes** dealing with a bug in the MLDP program. When you're ready to go, enter FCS with ESC D and run **MLDP;1** (if you have a 16K machine) or **MLDP;2** (if you have a 32k machine). The program will print a heading and then give you a **DBG>** prompt. There are a number of commands you can now give it. For example, it will translate a number from hexadecimal into decimal or vice-versa if we give it the "=" command. (The computer's output is in bold face in the following examples; my input is in regular type and is followed by a RETURN.)

```
DBG>=10
=0010 #16
```

The number typed in after the "=" is assumed to be a hexadecimal number, but if it is preceded by "#", then it is taken to be decimal. So DBG understood my command to be: "Translate the hexadecimal number 10 into decimal". This is did; it printed out hexadecimal 10 and then the decimal equivalent, 16 (preceded by "#" to indicate decimal). Similarly,

```
DBG>=42-#13
=0035 #53
```

tells me that subtracting decimal 13 from hex 42 yields hex 35 (=decimal 53). You now have a convenient hex/dec calculator-converter. You don't really need binary output as well, since you can easily read off the binary equivalent from the hex number. (See the previous article in this series.)

The MLDP can also set up a series of machine language instructions and then execute them. (Furthermore, we can choose two modes of execution, as I'll explain below.) The command "@" followed by a number tells the program to point to the address in the computer's memory specified by the number. For example,

```
DBG>@8200
8200 E5      'E'    PUSH    H
MEM>
```

Just above the **MEM>** there is a line of information consisting of four fields. The first (at the left of the screen) tells you the memory location (in hex, always). In our case it is 8200. To the right of that is a two digit hex number, or else two two digit hex numbers, or else three two digit hex numbers. Whether there is one, two, or three numbers depends upon what information is presently found at memory location 8200H. We'll come back to this later. To the right of the numbers is one (or two, or three) characters within single quotes. Let's ignore that for the moment. And finally, there are two more columns; together they comprise the assembly language mnemonics corresponding to the information stored at 8200H. More on that later.

Notice that the **DBG>** prompt has been changed to **MEM>**. This means we are in the so-called "memory mode" rather than the debug mode. In the MEM mode we can put a byte of data--for example, 3EH--into the present memory location by entering

```
MEM>=3E
```

After pressing RETURN, the 3EH (=62 decimal) was stored in 8200H and now another line of display tells us what is at the next location, 8201H. Let's enter 41H there:

```
MEM>=41
```

(Remember that whatever is typed in is assumed to be in hex unless preceded by "#" to indicate decimal.) Now enter these three bytes: CD, 33 and 00.

```
MEM>=CD
MEM>=33
MEM>=00
```

Now let's go back where we started and see what has been accomplished. Go back to location 8200H by entering:

```
MEM>@8200
```

You should get this result:

```
8200 3E 41      '>A'      MVI      A,41H
```

This says that we are at memory location 8200H and that the bytes 3EH and 41H are found there. Well.... There can be only **one** byte per memory location. That's the 3EH. The 41H is evidently in the next memory location, namely, 8201H. Why are we shown two bytes at once? The answer is more easily seen from the information at the right: **MVI A,41H**. This means, in assembly language mnemonics, "Put the number 41H into the Accumulator." The "**MVI A...**" is the assembly language equivalent of 3EH (=00111110), which is what is stored at 8200H. That is, the MLDP program has translated the numbers at 8200H and 8201H into assembly language mnemonics for us. It has done just the reverse of what an assembler does, and so we say that it has **disassembled** some machine language instructions. Instead of telling us that "**MVI A**" was stored at 8200H, it tells us something more useful, namely, "**MVI A,41H**".

Now press the "+" key and RETURN. You will be shown the contents of the next location in memory, namely,

```
8201 41          'A'      MOV      B,C
```

Oops! What is that "**MOV B,C**" doing there? That is an instruction to copy the contents of register C into register B. But the **MOV B,C** instruction apparently overlaps the **MVI A,41H** instruction. What's going on?

The 8080 understands numbers held in memory locations in two ways: (1) as a number, such as 41H, or (2) as an instruction, such as ... er ... 41H! How does the 8080 know that 41H is on one occasion merely a number to do something with and on another occasion an instruction? Well, that

depends. If the 8080 **starts** executing at 8200H, it will find 3EH (or, in assembly language, MVI A). But the instruction is incomplete: move **what** into the accumulator? Why, the next number, of course: 41H. So in this case 41H is understood as a mere byte of data to be used somehow. On the other hand, if execution were to begin at 8201H, then 41H would be interpreted as **MOV B,C**. So if we ask the MLDP to tell us what is at location 8201H, it will oblige by displaying 41H and then telling us that 41H is equivalent to **MOV B,C**. After all, the MLDP doesn't know where we plan to begin execution.

The "+" key moved us forward one byte in memory. The "-" key will move us backward one byte. Try it. You should be back at 8200H.

Now press RETURN without entering anything. This will move you ahead not just one byte, but rather to the next instruction, even if, as in the case of 3EH 41H, the instruction takes up more than one byte. Now you should see:

```
8202 CD 33 00 'M3@' CALL 0033H
```

The CDH (that's a hex number, by the way, and not some kind of mnemonic) is a **CALL** (like **GOSUB** in BASIC) instruction. But CALL what (or where)? In BASIC, a GOSUB must have a line number to GOSUB to. When you're talking directly to the 8080, you will have to give it an **address**. Remember that memory addresses are two bytes long, and so an address can be anywhere from 0 decimal to 65535 decimal. So 33H 00H following the CDH represent (in hex) the address of the subroutine. But notice that out to the right it says **CALL 0033H**, whereas to the left it says **CD 33 00**. Why have the "33" and the "00" been reversed? The 8080 always assumes that the **low byte** (i.e. the right-most, or least significant eight bits) of the address comes first, then the **high byte**. Some people feel that this is backwards, but I have found it quite natural. Just remember low byte first in memory, eventhough we write it with the high byte first.

When the 8080 performs a CDH (=CALL) instruction, it goes off to execute some subroutine until it encounters C9H (=201 decimal, or **RET** in assembly language), in which event it will return to where it was before and continue. How does it know where it was before? When CDH (=CALL) is encountered, the 8080 will store the address of the next instruction **after** the CALL instruction. When it returns from the subroutine it can retrieve that address and continue. Since our CALL instruction is located at addresses 8202H, 8203H and 8204H, the address of the instruction after the CALL is address 8205H. Where is this address saved during a CALL? It is saved in a reserved area of memory called the **STACK**. We'll talk more

about that in a later installment.

The display is still in **MEM** mode. Let's go back to the **DBG** mode. We do that by entering **"/** and **RETURN**. (Be careful not to enter **/** when in the **DBG** mode, because that is a command to exit the **MLDP** altogether and return control to the machine's **CRT** mode.)

Now we're going to set a **BREAKPOINT**. A breakpoint is a location in memory that will halt execution of a machine language program and return control to **DBG**, sort of like putting an **END** statement into a **BASIC** program to halt execution and return control to **BASIC**. A breakpoint is set by entering **AT <address>**. Thus,

```
DBG>AT 8205
```

will set a breakpoint at 8205. You can put in a number of breakpoints throughout a program for debugging purposes. To list all the breakpoints, type in **L**. In our case, there is only one breakpoint, so we'll see:

```
DBG>L
8205: C3
```

This tells us that a breakpoint is set at 8205H, and that the contents of 8205H is C3H. (Your memory might have something else there.) We don't care what's there; we only want to make sure that the program will stop there and not continue on trying to execute whatever random assortment of numbers might be in memory. If we wanted to clear a breakpoint, we would type in

```
DBG>C 8205
```

But don't do that, because we want the breakpoint there.

Believe it or not, we now have a program to execute. There are two ways of executing a program: **R <address>** and **I <address>**. The **R** command will transfer control to the 8080, which will begin executing at the address specified. It will run at top speed until it bumps up against a breakpoint, in which case control will return to **DBG**. The **I** command will execute much slower and in this case execution may also be interrupted by pressing the **ATTN/BREAK** key. Sometimes the **I** command is helpful because you will want to see things happen in slow motion.

Let's execute the machine language instructions beginning at 8200H.

```
DBG>R 8200      or else   DBG>I 8200
```

In this case it won't matter whether you use **R** or **I**, because the program is so short.

What will be the result?

```

DBG>R 8200
A
BREAKPOINT AT 8205
A B C D E H L M SW:(SZXPC) PC SP (SP+0,SP+2,SP+4,SP+6)
41 0000 0000 0000 C3 02 00000 8205 E000 3C3A FE09 CAF1 E00E
8205 C3 33 00 'C30' JMP 0033H
DBG>

```

(Some of the values on your display will differ from mine.) The first thing to notice is the **A** on the second line. Our program was a simple one: it put 41H (=65 dec., which is the ASCII code for "A") into the accumulator and then CALLED a subroutine in ROM (located at 33H). That subroutine takes whatever is in the accumulator and puts it on the screen. The subroutine was RETURNed back to our program, where it immediately encountered a breakpoint. The rest of the display shows the resulting contents of the 8080's registers (A, B, C, D, E, H, L), the contents of the **Status Word** (i.e. the **FLAGS**), the **Program Counter**, the **Stack Pointer**, and the contents of part of the stack. On the last line of the display, we are shown the contents of memory location 8205H, where the breakpoint was encountered.

For now, nevermind all that information. The interesting thing is that we have written a program to put something onto the screen. It works just like the PLOT statement in BASIC: PLOT 65 would put an "A" on the screen at the present cursor location. The rule is, each time you want to put a character onto the screen (or change colors, or draw vectors, or do anything else you do with BASIC'S PLOT statement), just put the appropriate number into the accumulator and then CALL 33H. Try this:

```

DBG>C 8205      (clear the breakpoint)
DBG>@8200
8200 3E 41      '>A'   MVI    A,41H
MEM>

```

Now type in each of the following 40 two digit hex numbers, preceding each with "=" and following each with the RETURN key.

BASIC equivalent

3E 0C CD 33 00	PLOT 12
3E 02 CD 33 00	PLOT 2 -- general plot mode
3E 00 CD 33 00	PLOT 0
3E 0A CD 33 00	PLOT 10
3E F2 CD 33 00	PLOT 242 -- vector plot
3E 64 CD 33 00	PLOT 100
3E 7F CD 33 00	PLOT 127
3E FF CD 33 00	PLOT 255 -- plot mode exit

You should now be pointing to location 8228H. (If not, go back to 8200H and check your entries.) Return to the DBG mode with "/". Set a breakpoint at 8228H and then execute the program beginning at 8200H. (Use either **R 8200** or **I 8200**.) Notice how fast the 8080 executes when allowed full speed. The **I** command is interesting, since the machine language program will then be **interpreted**, and the results will be much slower--slower, sometimes, than even BASIC. For example, the PLOT 12 instruction evidently first homes the cursor and then erases each line of the screen from the bottom up. Did you know that?

NEXT TIME we'll learn more about the 8080 instruction set, and we'll explore some much easier methods of putting stuff onto the screen. Until then, fiddle around with the present method. Have fun! **☛**

(Continued from page 5.)

especially well-suited to small businesses, personal use, and MIS environments.

3651 desktop \$2945

Complete desktop computer with color graphics, 13" CRT, 72-key keyboard, RS-232C, selectable baud rate, disk BASIC, internal mini-disk drive with 92K bytes storage, 16K user RAM. Graphics resolution: 128 x 128.

9651 desktop \$3345

As above, but with 19" CRT.

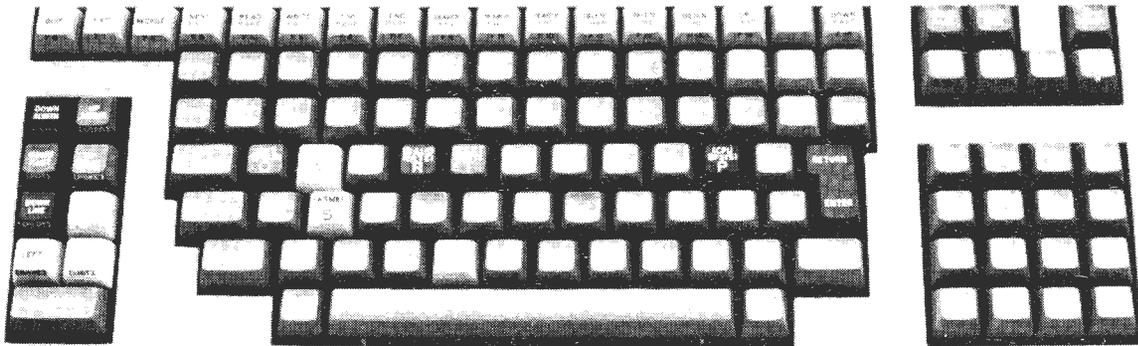
3654 desktop \$5445

Same as 3651 but with 1182K dual 8" double-headed floppy disk drive.

9654 desktop \$5854

Same as above but with 19" CRT. **☛**

LOOKING FOR CUSTOM KEYCAPS for your **COMPUCOLOR II or INTECOLOR COMPUTER?**



We stock blank keycaps that match the original keycaps supplied on ISC's keyboards

1 × 1 size in 16 colors

1 × 2 size in 14 colors

Single quantity prices for one line engraved keycaps start at:

\$1.17 for 1 × 1 size with 5 characters

\$1.95 for 1 × 2 size with 10 characters

Other sizes and specific colors are available on special order.
Custom molded keytops available for high volume users.

For order form call or write:

Arkay Engravers

2073 Newbridge Road
Bellmore, NY 11710

516-781-9859

Sphere Program

By Mark D. Fairbrother

Carriage House East, A5

Rt. 11

Kirkwood, NY 13795

This program will generate a 3-D line drawing of a sphere on a 128x128 screen using routines from "Mathematical Elements of Computer Graphics." Note that lines 2000 and up are for use with the MX-80 screen dump routine if you wish to add it. (See Aug./Sept. **Colorcue**.) And in that case you ought to add line 10 DIM MS(7) and line 20 FOR I=0 to 7:MS(I)=INT(2^I):NEXT.

```
100 REM GENERATE A SPHERE USING THE SPHERICAL COORDINATE SYSTEM
110 DIM X(100),Y(100),Z(100)
120 RO=30:I=1
130 FOR TH=0 TO 2.51327 STEP .628317
140 FOR PH=0 TO 6.28319 STEP .330694
150 X(I)=RO*COS(TH)*SIN(PH)
160 Y(I)=RO*SIN(TH)*SIN(PH)
170 Z(I)=RO*COS(PH)
180 I=I+1
190 NEXT PH
200 NEXT TH
210 P=100
1080 REM DIMETRIC PROJECTION ROUTINE
1090 REM FROM 'MATHEMATICAL ELEMENTS FOR COMPUTER GRAPHICS'
1100 REM P = NUMBER OF X,Y,Z TRIPLETS
1110 REM X( ) = ARRAY CONTAINING X-COORDINATES
1120 REM Y( ) = ARRAY CONTAINING Y-COORDINATES
1130 REM Z( ) = ARRAY CONTAINING Z-COORDINATES
1140 DIM U(100,4),V(100,4),T(4,4)
1150 FOR I=1 TO P:FOR J=1 TO 4:U(I,J)=0:V(I,J)=0:NEXT J:NEXT I
1160 FOR I=1 TO P
1170 U(I,1)=X(I):U(I,2)=Y(I):U(I,3)=Z(I):U(I,4)=1
1180 NEXT I
1190 FOR I=1 TO 4:FOR J=1 TO 4:T(I,J)=0:NEXT J:NEXT I
1200 T(1,1)=.92582:T(1,2)=.133631:T(1,3)=-.353553
1210 T(2,2)=.935414:T(2,3)=.353553
1220 T(3,1)=.377964:T(3,2)=-.327327:T(3,3)=.866025
1230 T(4,4)=1.0
1240 FOR I=1 TO P:FOR J=1 TO 4:FOR K=1 TO 4
1250 V(I,J)=U(I,K)*T(K,J)+V(I,J)
1260 NEXT K:NEXT J:NEXT I
1270 FOR I=1 TO P
1280 X(I)=V(I,1):Y(I)=V(I,2):Z(I)=V(I,3)
1290 NEXT I
1300 N=3
1310 REM AXONOMETRIC PROJECTION ROUTINE
1320 REM FROM 'MATHEMATICAL ELEMENTS FOR COMPUTER GRAPHICS'
1330 REM P = NUMBER OF X,Y,Z TRIPLETS
1340 REM X( ) = ARRAY CONTAINING X-COORDINATES
1350 REM Y( ) = ARRAY CONTAINING Y-COORDINATES
1360 REM Z( ) = ARRAY CONTAINING Z-COORDINATES
1370 REM N = NUMBER INDICATING THE PERPENDICULAR AXIS
1380 REM N=1 X-AXIS, N=2 Y-AXIS, N=3 Z-AXIS
1390 FOR I=1 TO P:FOR J=1 TO 4:U(I,J)=0:V(I,J)=0:NEXT J:NEXT I
1400 FOR I=1 TO P
1410 U(I,1)=X(I):U(I,2)=Y(I):U(I,3)=Z(I):U(I,4)=1
1420 NEXT I
```

```

1430 FOR I=1 TO 4:FOR J=1 TO 4:T(I,J)=0:NEXT J:NEXT I
1440 T(1,1)=1:T(2,2)=1:T(3,3)=1:T(4,4)=1
1450 IF N=3 THEN 1490
1460 IF N=2 THEN 1480
1470 T(1,1)=0:GOTO 1500
1480 T(2,2)=0:GOTO 1500
1490 T(3,3)=0
1500 FOR I=1 TO P:FOR J=1 TO 4:FOR K=1 TO 4
1510 V(I,J)=U(I,K)*T(K,J)+V(I,J)
1520 NEXT K:NEXT J:NEXT I
1530 FOR I=1 TO P
1540 HX=X(1):LX=X(1):HY=Y(1):LY=Y(1)
1550 FOR I = 2 TO P
1560 IF X(I)>HX THEN HX=X(I):GOTO 1580
1570 IF X(I)<LX THEN LX=X(I)
1580 IF Y(I)>HY THEN HY=Y(I):GOTO 1600
1590 IF Y(I)<LY THEN LY=Y(I)
1600 NEXT I
1610 DX=HX-LX:DY=HY-LY:MX=127/DX:MY=127/DY
1620 PLOT 12,6,6,2,INT(0.5+MX*(X(1)-LX)),INT(0.5+MY*(Y(1)-LY))
1630 PLOT 242
1640 FOR I=1 TO P
1650 X(I)=INT(0.5+MX*(X(I)-LX))
1660 Y(I)=INT(0.5+MY*(Y(I)-LY))
1670 PLOT X(I),Y(I)
1680 NEXT I
1690 PLOT 255
2000 REM 2XXX LINES ADDED TO RUN SCREEN-PRINT IF WANTED.
2010 PLOT 15:INPUT "HARDCOPY(Y/N)";A$
2020 IF A$="N" THEN END
2030 PLOT 3,0,0:INPUT "HIT RETURN TO START";A$
2040 GOSUB 9000
2050 END

```

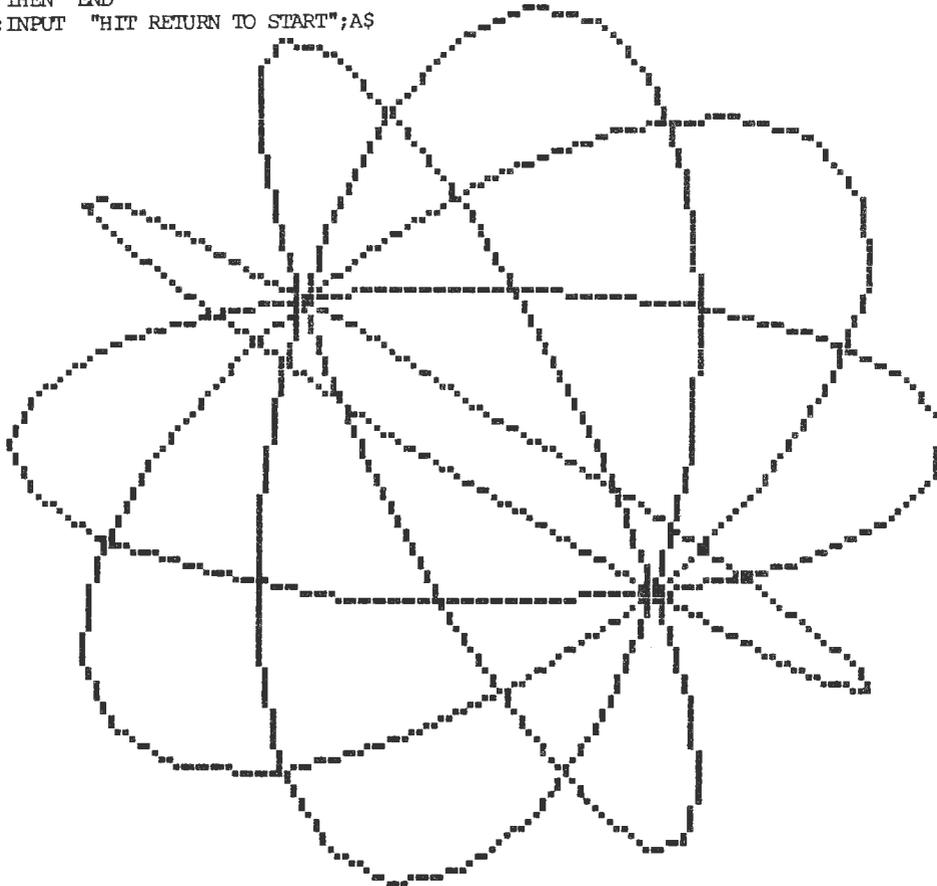


Image is 65%
of original size.

Incremental Plot Table

by Bob V. Smith
498 Brown Street
Napa, CA 94558

This table allows you to discover the proper plot number for incremental point and bar graph plotting. In the general plot mode, PLOT 251 enters the incremental point plot submode. The next number specifies the placement of two more plot blocks out from the first, and this number can be found in the table. For example, if you want to move northeast and then southeast for the two incremented blocks, then PLOT 169 is the correct number. If the original block is at, say, X=63,Y=63, then the instruction:

```
PLOT 2,63,63,251,169,255
```

will plot the original block and the two incremented blocks.

The table also applies to incremental bar graph plotting, where the increment number will move the end point of the bar graph in the specified direction(s). **■**

First Block

	N	NE	E	SE	S	SW	W	NW	
	N	34	162	130	146	18	82	66	98
	NE	42	170	138	154	26	90	74	106
	E	40	168	136	152	24	88	72	104
Second	SE	41	169	137	153	25	89	73	105
Block	S	33	161	129	145	17	81	65	97
	SW	37	165	133	149	21	85	69	101
	W	36	164	132	148	20	84	68	100
	NW	38	166	134	150	22	86	70	102

Lissajous Figures

by Trevor Taylor

36 Tarm St.

Wavell Heights

Brisbane 4012

Australia

Lissajous figures are drawn by applying sine waves to the horizontal (x) and vertical (y) deflection plates of an oscilloscope. The two variables that determine the shape of the figure are the ratio of the frequencies of the two sine waves and the phase angle between them.

When the frequency ratio is an integer, it will be the number of lobes (loops in the figure) if the phase angle is 90 degrees. If you consider the figure to be three dimensional, the phase angle (which is how much one sine wave is shifted with respect to the other) determines how much it is rotated. A frequency ratio of one and a phase angle of zero will draw a straight line, i.e. a circle turned on edge!

In the program, line 30 determines the foreground and background colors. The step size (degrees) used in the calculations is set in line 70. This is a compromise between speed and accuracy. Note that line 1050 and 2010 convert from degrees to radians as required for the SIN function. In lines 2040 and 2050 the sine waves are scaled for the screen and the aspect ratio is taken care of so that a circle will appear round.

Lines 2085 and 2087 take care of the case when you want only one figure drawn, which you specify by giving an increment of zero. Removing the 12 in line 1060 stops the program from erasing the screen between figures, and can give some pretty patterns. You could also modify the program to run through a range of frequency ratios.

A good example is: Frequency ratio = 3, Starting Phase angle = 90, Ending Phase angle = 90, and Increment = 30.

Try to visualize the figure rotating as the program draws successive pictures. Taking the reciprocal of the frequency ratio (for example, 0.33333) will turn the figure sideways, i.e., interchange x and y on the screen. Experiment and have fun! Try to understand the math if you can.

Note: The Australian Broadcasting Commission used this particular Lissajous figure as its symbol for a number of years. Of course, on TV it was done using an oscilloscope and rotated much faster than is possible on your computer. 

```

10 REM LISSAJOUS FIGURES
11 REM
12 REM BY TREVOR TAYLOR
13 REM
20 PI=3.14159
30 PLOT 6,2
40 PLOT 12
50 Z=360
60 GO=1
70 SS=10
79 REM MIDPOINT X,Y OF SCREEN
80 XI=64:REM FOR INTECOLOR 8001 USE XI=80
90 YI=64:REM FOR INTECOLOR 8001 USE YI=96
99 REM SCALING FACTORS
100 XF=.75*(XI-1)
110 YF=YI-1
500 INPUT "FREQUENCY RATIO (0.1-10): ";E
510 INPUT "STARTING PHASE ANGLE (0-360): ";PS
520 INPUT "ENDING PHASE ANGLE (0-360): ";PE
530 INPUT "INCREMENT: ";IC
540 PLOT 12
550 IC=ABS(IC)
560 PH=PS
1000 IF E>10 THEN E=10
1010 IF E<0.1 THEN E=0.1
1020 IF E<1 THEN Z=360/E
1030 IF PH<0 THEN PH=-PH
1040 IF PH>360 THEN PH=PH-360
1050 A=PH*PI/180
1060 PLOT 12:REM THIS MAY BE OMITTED
1070 ST=YI+YF*SIN(A)
1080 PLOT 2,253,XI,ST,242
2000 FOR I=SS TO Z STEP SS
2010 T=I*PI/180
2020 X=SIN(T)
2030 Y=SIN(E*T+A)
2040 X=XI+XF*X
2050 Y=YI+YF*Y
2060 PLOT X,Y
2070 NEXT I
2080 PLOT 255
2085 IF IC=0 THEN 2500
2087 IF GO<>0 THEN 2100
2090 IF PH=PE THEN 2500
2100 PH=PH+IC
2110 FOR I=1 TO 50:NEXT
2120 GO=0
2130 GOTO 1000
2500 END

```

Ask Mr. C. O. Lorcue . . .

Dear Mr. Lorcue,

Strange things happen when I press the INSERT LINE key on my extended keyboard when I'm in BASIC. Can you explain what is going on? Or is my machine defective? Signed: Keyed Up

Dear Keyed Up,

The explanation is not hard to find. Each key on the keyboard is translated into a number from 0-255. The 'A' key, for example, translates as 65 (the ASCII code for 'A'); the ERASE PAGE key translates as 12; and so on. These numbers are PLOTted by BASIC. So PLOT 65 produces an 'A' on the screen, and PLOT 12 erases the screen. The INSERT LINE key, just like CONTROL C, translates as 3. Now, what is PLOT 3? Cursor control! The next two keys you press would ordinarily determine the X,Y coordinates of the cursor on the basis of those keys' ASCII values. Unfortunately, BASIC doesn't know enough to stay in control, and it will go away after such key presses. You can get it back with ESC E. Unlike many keyboard controls, the INSERT LINE cannot be embedded within quotes in BASIC statements. So it looks like the only obvious use of this is for one method of cursor control in the CRT mode. For example, go into CRT mode and try these:

<u>Keystrokes</u>	<u>Comment</u>
INSERT LINE (or CONTROL C)	=3
SPACE	=32 = PLOT 3,32,0
CONTROL @	=0
INSERT LINE (or CONTROL C)	=3
A7 ON	=14 = PLOT 3,14,15
BL/A7 OFF	=15

(If you have questions for Mr. Lorcue, send them in and we'll pass them on. -Eds.) **◀**

Cueties . . .

```
PLOT 12,3,64,0,2,63,63,251:FOR X=0 TO 1000:PRINT -ATN(X);:  
PLOT 255,6,X AND 7,2,251:NEXT:PLOT 255,6,2 ◀
```

CALLable Sort Routine

by Alan D. Matzger

960 Guerrero St.
San Francisco, CA 94110

A few issues of *Colorcue* ago, Myron Steffy had an article introducing the CALL statement. I came upon its utility first when I was writing a text editor for Assembler source code. Like Mr. Steffy, I wanted to move large blocks of memory from one place to another and BASIC's PEEKing and POKEing was unconscionably slow. Having solved the memory moving with an assembly language routine, I went on to use the CALLED routine to list text, edit individual lines, search for particular strings, and insert, replace, delete, move, and copy one or more lines.

Which function was performed depended only on the particular X in A=CALL(X). I have since used the CALL statement in the screen editor, in a "stolen" HP-85 Cribbage game to draw cards and evaluate hands and, as illustrated below, to sort any one of BASIC's string arrays.

The January 1981 *Colorcue* listed the labels for scratchpad addresses in alphabetical order. I wanted the listing in ascending order of the hex addresses. First, I copied each entry, interchanging the label with the hex location, and saved them in a .RND file with 137 records, each 50 bytes long. Using a simple program which read the records into a string array and then CALLED the sort routine, I was able to sort the 137 records in about 3 seconds.

The algorithm for the routine is the Shell-Metzner sort, modified to leave the array in place and exchange the entries' index numbers in a "pointer array" if the lower entry is "higher" than the higher one in the ASCII collating sequence. An explanation of the Shell-Metzner algorithm may be found in any of several computer science texts, and in articles in computer magazines, such as BYTE. My intent here is to demonstrate another way of using the CALL statement. A short explanation of the assembly language routine is in order, however.

When RUN through FCS, the routine is entered at the label START, where it sets up the CALL vector and adjusts BASIC's end of memory pointer to just in front of START itself, to avoid having the routine overwritten. Doing this within the routine itself cleans up the BASIC program, leaving only a mandatory CLEAR statement to be done there.

When CALLED, the routine is entered at the label MAIN with the value of X in A=CALL(X) converted into two bytes and received in the DE register

(with the high byte in B). The HL register contains BASIC's return address and it must be restored to HL upon RETURNing. The two bytes in DE upon RETURN are converted to a decimal number which is the value of A. It may or may not be useful to the CALLing program; in this case it is used to pass back the address of the pointer array. Strictly speaking then, only one parameter may be passed and returned by the CALL statement. But as demonstrated by the BASIC demonstration program below, the CALLing program can have access to any amount of data calculated by the assembly language coded routine.

(At the start of the routine, the pointer array INDX0 is set up in 1,2,3,... order. At the end, the index number of the "least-valued" string will be in the first slot of this pointer array. The demo program reads this array sequentially, converting the two bytes to a decimal number which is the index of the next higher string and prints it. Similarly, more than one value can be passed to the routine by POKEing into appropriate locations within the assembled code. One must be careful to note the addresses of wanted data areas from the listing by the assembler, convert them into decimal numbers, and then insert these numbers into the BASIC program.)

FNDAR finds the array name passed to it in BASIC's array table, and saves the number of entries and the address of the first string for the sort routine.

Finally, the CALLED routine need not have its own stack; I have consistently used whichever one BASIC uses and have never run out of PUSHing room. The CALL statement, thus, is not nearly as restrictive as suggested in the Programming Manual. BASIC is fast enough for most input-output operations, and the use of assembly language routines inserted via CALLs greatly enhances its usefulness by adding speed when needed. CALLing machine language routines has made my machine even more enjoyable, fast and powerful. **C**

```

PAGE 60
; ASSEMBLY LANGUAGE SORT ROUTINE CALLABLE FROM BASIC
;
; Alan D. Metzger
; 960 Guerrero St.
; San Francisco, CA 94110
;
; Establish run-time linkages
; START is entered when the PRG file is RUN by FCS
;
ENTRY START
START:
PUSH PSW ;save regs at entry
PUSH H
LXI H,MAIN ;set up CALL vector with address,
SHLD 33283
MVI A,(JMP) ; and JMP op-code
STA 33282
LXI H,START-1
SHLD 32940 ;set top limit of BASIC to below us
POP H ;restore regs
MVI B,0 ;(except B, a 0 in which indicates OK to FCS)
POP PSW
RET ;and leave.
; Shell sort of BASIC's string arrays
; MAIN is entered when BASIC program executes a CALL
;
MAIN: PUSH H ;Save BASIC address
CALL FNDAR ;Subrtn to find list to sort
CALL MKNDX ;Initialize pointer array
CALL SHSRT ;The actual sort
LXI D,INDX0 ;Return the address of ptr. array
EXIT: POP H ;Retrieve return address
RET ;Back to BASIC
;-----
V678 EQU 1 ;Select version by setting aprop. V to 1
V879 EQU 0 ; and others to 0
V990 EQU 0
;-----
BGARR EQU 32984 ;POINTER to start of arrays
NDARR EQU 32985 ;Pointer to end of arrays

```

```

IF V879 OR V980
OSTR EQU 182AH ;Puts out string to screen
NEGH EQU 195AH ;Negates HL
ENDIF
IF V678
OSTR EQU 33F4H
NEGH EQU 3524H
ENDIF
PAGE
; FNDAR - Find array subroutine
; DE contain array name as 256*first ch + second ch + 128
; e.g. if array name is ALS, then D = x'41', C = x'CC'
;
; on exit, NAME0 --> ARRAY
; NELEM has # elements
;
FNDAR:
LHLD BGARR ;HL points to first array
MOV B,D ;Keep name in BC
MOV C,E
PUSH H
FNDR1: POP H
MOV A,M ;look at first byte
INX H ;point to second
CMP C ;lobytes the same?
JNZ FNDR2 ; IF NOT, look at next entry
MOV A,M ;look at second
CMP B ;hibytes the same?
JZ FOUND ;yes.
FNDR2: INX H ;point to next byte
MOV E,M ;these contain offset to next entry
INX H
MOV D,M
INX H
DAD D ;add offset
PUSH H
XCHG ;but maybe we're
DCX D
LHLD NDARR ;beyond last entry
MOV A,H
CMP D
JM FNDER ;we are - show error
JNZ FNDR1 ;we're not - look at next
MOV A,L ;compare lobytes
CMP E
JNC FNDR1 ;we're not = look again

```

```

FNDER: LXI    H, FNEMG
        CALL   OSTR    ;display error msg
        POP    H
        POP    H
FOUND:  LXI    B,4     ;nothing more to do
        DAD    B      ;in a one dimension list,
        ; # elements is 4 bytes away
        MOV    E,M
        INX    H      ;get that number
        MOV    D,M
        XCHG
        SHLD   NELEM  ;and store it away
        XCHG
        INX    H      ;next byte is first for #0
        SHLD   NAME0  ;save that away
        RET
PAGE
; MKNDX - Make index array subroutine
;
; During the sort, the strings themselves are not
; changed; their indices in a pointer array (INDX0)
; are switched. This subrtm initializes that array.
MKNDX:  LXI    H, INDX0 ;addr of 0th element
        LXI    D,0     ;index and value start the same
        LXI    B,1280 ;there are 640 entries MAX
MKNX1:  MOV    M,E     ; 2 bytes each
        INX    H
        MOV    M,D    ;value stashed
        INX    H      ;prepare for next
        INX    D      ; value is one more
        DCX    B      ;are we all done?
        MOV    A,C    ; we'll see
        ORA    B      ;not if result is <>0
        JNZ   MKNX1  ; nope
        RET          ;yup
PAGE
; SHSRT - The actual sort routine
; INNO1 & INNO2 are the pointer array indices
; NSTR1 & NSTR2 are the values in the index
; and are themselves the indices to the two
; strings in their own array.
SHSRT:  LHLD   NELEM  ;the number of strings
SSLP1:  ANA    A      ;clear array
        MOV    A,H

```

```

        RAR                    ;we are dividing by two
        MOV    H,A            ;to get the partition factor
        MOV    A,L
        RAR
        MOV    L,A
        ORA    H              ;if it is zero
        RZ                    ; we're all done
        SHLD   PARTN         ;but we aren't
        CALL   NEGH          ;this # is used
        SHLD   NGPTN        ; in many subtractions
        XCHG
        LHLD   NELEM        ;as here
        DAD    D
        SHLD   LPLIM        ;this is loop limit
        LXI    D,0          ;start with first string
SSLP2:  INX    D            ;get next index
        XCHG
        SHLD   INN01        ;store it
        XCHG
        LHLD   LPLIM        ;is it > limit?
        CALL   NEGH
        DAD    D            ;compare them
        LHLD   PARTN        ;but load this before the test
        JC    SSLP1        ;it is greater, goto loop1
SSLP3:  DAD    D            ;it's not, add partition factor
        SHLD   INN02        ; to get second index
        CALL   DTSTR        ;sr to get string indexes
        CALL   GT2ST        ;this gets their len and address
        CALL   COMPR        ;sr to compare two strings
        LHLD   INN01        ;load first index in case
        XCHG
        CPI    DOSWT        ; the two must be switched
        JNZ   SSLP2        ;they don't, go back for more
        CALL   SWTCH        ;they do
        LHLD   INN01
        XCHG                ;INN01 now in DE
        LHLD   NGPTN        ;1st str of next comparison is
        DAD    D            ;INN0 - PARTN, if that's
        MOV    A,H          ; not zero or less
        ANA    A
        JM    SSLP2
        JZ    MRTST
        JP    RTT03
MRTST:  ORA    L            ;is L zero too?
        JNZ   RTT03        ;if not, goto LP3
        LHLD   PARTN        ; if so, goto LP2
        JMP   SSLP2

```

```

RTT03: SHLD   INN01   ;stash new str1 index
        XCHG           ;put in DE
        LHL   PARTN
        JMP    SSLP3

; DTSTR - Obtain NSTR's from INN0's
DTSTR:  LXI    D,INDX0 ;start of index
        LHL   INN01
        LXI    B,NSTR1 ;address, not the value
        CALL  NFRIN   ;this gets and inserts the value
        LHL   INN02 ;repeat for second
        LXI    B,NSTR2
        CALL  NFRIN
        RET

NFRIN:  DAD    H      ;HL*2, each entry is 2 bytes
        DAD    D      ;points to value in index
        MOV    A,M
        STAX  B      ;store lobyte
        INX   H
        INX   B
        MOV    A,M
        STAX  B      ;store hibyte
        RET
        PAGE
; GT2ST - Get length and starting addresses of the 2 strings
GT2ST:  LHL   NAME0   ;addr of ptr to string # 0
        XCHG           ;put in DE
        LHL   NSTR1
        CALL  GLNAD   ;this gets 'em
        MOV    B,C    ;len1 now in B, len2 will be in C
        SHLD  ASTR1   ;addr returned in HL
        LHL   NSTR2
        CALL  GLNAD
        SHLD  ASTR2   ;do same for 2nd str
        RET

GLNAD:  DAD    H      ;each entry is 4 bytes
        DAD    H      ; so mult nstr by 4
        DAD    D      ;add to NAME0
        PUSH  D      ;this byte is len of str
        MOV    C,M    ;this one is filler
        INX   H      ;lobyte of addr in string space
        INX   H
        MOV    E,M

```

```

        INX   H      ;stash this addr in DE
        MOV    D,M
        XCHG           ;put it in HL
        POP    D      ;retrieve NAME0
        RET

; COMPR - The comparison routine
; If 1st <= second, A returns FFH, else 00H
COMPR:  LHL   ASTR2   ;Point to its first byte
        XCHG           ;In DE
        LHL   ASTR1   ;addr of 1st string
        LDAX  D      ;get the byte
        CMP    M      ;is str2 > str1?
        JM    OGT2    ;no, it's less
        JZ    OEQ2    ;no, it's equal
        OLT2: XRA    A      ;yes -- put 00 in A
        RET

OGT2:   MVI    A,0FFH ;tell caller to switch
        RET

OEQ2:   DCR    B      ;end of str1?
        JZ    OLT2    ;yes, so 2nd > 1st
        DCR    C      ;end of str2?
        JZ    OGT2    ;yes, 1st > 2nd
        INX   D      ;point to next byte
        INX   H      ; ditto
        JMP   COMPR

PAGE
; SWTCH - Switch values in index array
;
SWTCH:  LXI    D,INDX0
        LHL   NSTR2   ;this value will go
        PUSH  H      ; where NSTR1 was
        LHL   INN01   ; but we have to find
        DAD    H      ; original address
        DAD    D      ;here it is
        POP    B      ; now the value is in BC
        MOV    M,C
        INX   H
        MOV    M,B    ;all moved, now for other
        LHL   NSTR1
        PUSH  H
        LHL   INN02
        DAD    H
        DAD    D      ;here's the address
        POP    B

```

```

MOV M,C
INX H
MOV M,B ;switch completed
RET

; VARIABLES
;
NELEM: DS 2
INN01: DS 2
INN02: DS 2
NSTR1: DS 2
NSTR2: DS 2
ASTR1: DS 2
ASTR2: DS 2
PARTN: DS 2
NGPTN: DS 2
LPLIM: DS 2
NAME0: DS 2
DOSMT EQU OFFH
FNEMG: DB 5,1,3,20,5,237,50
DB 'LIST NOT FOUND',11,239,239
INDX0: DS 1280 ;THERE ARE 520 2-BYTE
END START ENTRIES
    
```

```

CALL IN ASSEMBLY LANGUAGE SORT ROUTINE

10 PLOT 27,4:PRINT "RUN SMSORT":PLOT 27,27
20 CLEAR 20000
25 REM

SET UP TEXT ARRAY TO BE SORTED

30 DIM AS(100)
40 FOR X= 1TO 100
45 SS= ""
50 FOR L= 1TO RND (3)* 20:REM LENGTH OF AN ITEM
60 SS= SS+ CHR$ (55+ 26* (RND (2)))
70 NEXT
75 AS(X)= SS
80 NEXT
85 REM

CALL SORT ROUTINE TO SORT AS

90 X= 128+ 256* ASC ("A")
95 PRINT "GOING TO SORT"
100 B= CALL (X)
105 REM

PRINT SORTED ARRAY, USING POINTERS AT LOC. B

110 DEF FN I(Z)= PEEK (Z)+ 256* PEEK (Z+ 1)
125 REM STEP THRU POINTER ARRAY, EACH 2 BYTES LONG
130 FOR I= 2TO 2* 100STEP 2
140 IX= FN I(I+ B)
150 PRINT AS(IX):NEXT
    
```

ALIEN INVASION \$15

V1.0 (no sound) (US funds)

Copyright (c) 1981 by David B. Suits

Fast, machine language invaders arcade game with color graphics and four levels of difficulty from "not-too-difficult" to "what-the-?!#*??. Runs on V6.78 and V8.79 software with standard keyboard.

Also included: fast, machine language version of LIFE.

Special bonus: V2.0 of Alien Invasion (if I ever get around to writing it; I'm basically lazy, so I'm making no promises) will have sound effects if you have Cap Electronics Soundware or an equivalent device, and it will be free to all purchasers of V1.0.

David B. Suits
49 Karenlee Dr.
Rochester, NY 14618

HOWARD ROSEN, INC.

Put the finishing touches to your CompuColor II or ISC computer.

- > Come up to the world of word processing.
- > Extend the utilization of your computer to the other members of your family.
 - * Letters
 - * School reports
 - * Business reports
 - * If you now type-write it, Comp-U-Write it for a better product.
- > Basic requirements for CCII or 3651/9651:
 - 16K RAM.
 - 117key keyboard.
 - Printer.
 - Comp-U-Writer software and instruction manual.
- > For maximum capability:
 - Full 32k RAM.
 - Lower case characters.
- > Talk to other computers: Add a MODEM to your system.

We carry the entire CCII & ISC line of hardware/software, including spares. Send for our 4-page order form for hardware/software. Request separately by item your spare parts needs.

Send your order now. We pay the shipping.
Allow 5 weeks for delivery.

CCII	3650/9650	Description	Quantity	Cost	Amount
010057	010053	Upgrade 72/101 keys	_____	150.00	_____
010058	010054	Upgrade 72/117 keys	_____	250.00	_____
010059	010055	Upgrade 101/117 keys	_____	100.00	_____
010044		24in. RS 232C Cable	_____	45.00	_____
HR1001	HR1002	16k RAM Add-on	_____	185.00	_____
010051		Switchable Lower Case	_____	150.00	_____
	0C03LC	32 Lower Case Characters	_____	100.00	_____
990001	990030	5in. Formatted Twin Pack	_____	9.95	_____
	900041	8in. - 10 One Side Format	_____	75.00	_____
	900044	8in. - 10 Two Side Format	_____	100.00	_____
HR0006	HR0006	5in. Exec. Comp-U-Writer	_____	299.00	_____
HR0007	HR0007	5in. Mail-Merge C-U-Writer	_____	349.00	_____
991509	991532	FORTTRAN	_____	75.00	_____
		Epson MX80 Serial printer	_____	665.00	_____
		Base-2, Inc. 850 Impact printer	_____	750.00	_____
		CAT Novation MODEM Transmit/Originate	_____	175.00	_____

	Sub Total	_____
Pa. residents add 6% Pa. Sales Tax	Pa. Tax	_____
	Total	_____
Terms - Cash with order		
Name _____	Telephone #(_____)	-
Address _____	City _____	St _____ ZIP _____

HOWARD ROSEN, INC.
PO Box 434
Huntingdon Valley, Pa. 19006
(215)-464-7145

Signature (please sign order)

BULK RATE
U.S. POSTAGE
PAID

Rochester, N. Y.
Permit No. 415

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

An  Publication

Colorcue



Dec/Jan 1982

\$2.

Colorcue

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

December 1981/January 1982
Volume 4, Number 3

Editors:

Ben Barlow
David B. Suits

-
- 3 Editors' Notes**
- 5 Protected Fields, by Bernie Raffee**
Control keyboard input with a machine language patch
- 13 Serial to Parallel Interface, by Ben Barlow**
A simple but useful project for your peripherals
- 19 Assembly Language Programming, by David B. Suits**
Part III: More on the 8080 instruction set
- 25 'The' BASIC Editor, reviewed by David B. Suits**
A powerful new utility for BASIC programmers
-

Advertisers: You will find our advertising policies attractive. Write for details.

Authors: This is a user-oriented and supported publication. Your articles/tips/hints are required to make it go. Send your articles or write for information.

Colorcue is published bi-monthly by Intelligent Systems Corporation, with editorial offices in Rochester, New York. Editorial and subscription correspondence should be addressed to the Editors, Colorcue, 161 Brookside Dr., Rochester, NY 14618. Product related correspondence should be addressed to ISC, 225 Technology Park, Norcross, GA 30092, ATTN: Susan Sheridan. Opinions expressed in by-line articles are not necessarily those of the editors or of ISC. Hardware/software items are checked to the best of our abilities but are NOT guaranteed.

Editors' Notes . . .

Notes on Assemblers

The Macro Assembler is a very powerful tool when working with assembly language programs. It is, however, different from and a little more difficult to use than the old original Assembler program. The differences caused some trouble to a few of our readers who keyed in the Assembly Language Sort subroutine in the last issue, and then tried to "Assemble" it (rather than "Macro Assemble" it). We apologize to those readers we misled. Although the Macro Assembler rates its own set of tutorial articles, the following brief notes will tell you what is needed to "translate" between the two versions of assembly language. **Colorcue** will often publish programs in the Macro version, since one of its editors threw away his old Assembler. [Tsk, tsk. --Other Ed.]

The most powerful feature of the Macro Assembler (hereafter known affectionately as Mac) is its support of relocatable, modular code. Instead of one mammoth source program, the Mac user can develop small modules, assemble them individually (faster editing, faster assembly, faster debug) and then "link" them together using the L80 linker. Mac doesn't know (generally) where the final PRG program will reside; the linker is told that. Mac tells the linker which labels must be known outside the current module by making available **ENTRY** and **EXTRN** commands. An **ENTRY** point is a name in a module that may be referenced by another module; an **EXTRN** is a name in another module that the current module wishes to reference. In our subroutines, replace an **ENTRY START** line with an appropriate **ORG** statement.

The Sort routine needed an **ORG 0F000H**.

Another one of Mac's features is that of "conditional assembly". It's possible to define conditions within a source file, and then select one of several options with assembly-time "switches". This feature can be seen in the Sort routine where different **EQU**s are generated based on the setting of the version variables, V879, V980 or V678. To use with the regular Assembler, simply delete the statements for the version in which you're not interested, as well as any **IF**, **ELSE**, and **ENDIF** commands you may find.

Mac's third major feature is the ability to define and use macros. Since these are just notes, and macros haven't been used in any **Colorcue** programs to date, we won't discuss them now, but save them for the later article they deserve.

Busses

We know that several hardware-oriented readers have developed peripheral devices and busses for the 50 pin bus. It would be advantageous to all of us--developers, users, would-be consumers--if we had a standard for an external bus interface. **Colorcue** (and some of its corresponding electrical engineers) will be happy to coordinate a standard-setting activity. Just send a definition of your bus to our editors, and we'll chair a remote committee of respondents. We'll also be happy to publish a dialog on the matter. Send us your comments, notes, articles.

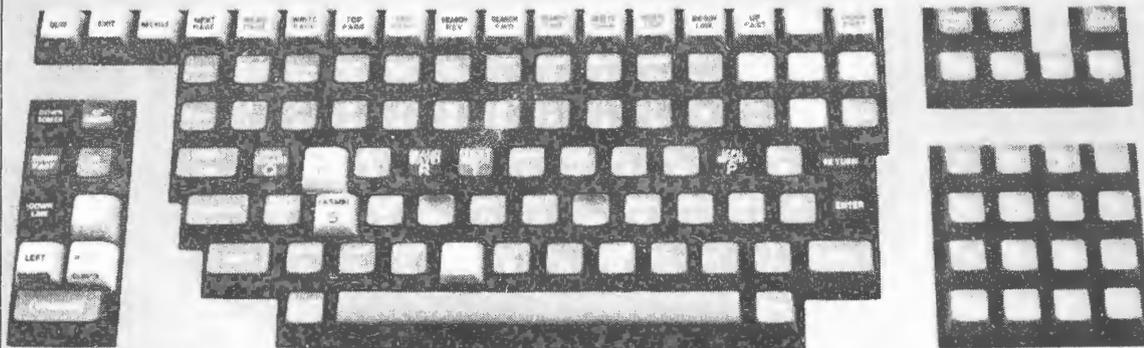
Back Issues

Please note that **Colorcue** back issues are for sale. See inside of this issue's back cover. **C**

LOOKING FOR CUSTOM KEYCAPS

for your

COMPUCOLOR II or INTECOLOR COMPUTER?



We stock blank keycaps that match the original keycaps supplied on ISC's keyboards

1 × 1 size in 16 colors

1 × 2 size in 14 colors

Single quantity prices for one line engraved keycaps start at:

\$1.17 for 1 × 1 size with 5 characters

\$1.95 for 1 × 2 size with 10 characters

Other sizes and specific colors are available on special order.
Custom molded keytops available for high volume users.

For order form call or write:

Arkay Engravers

2073 Newbridge Road
Bellmore, NY 11710

516-781-9859

Protected Fields

by **BERNIE RAFFEE**
31 Malvern Avenue
South Harrow, Middlesex
England HAU 9EU

Although the common method of input to a BASIC program (prompted INPUT) is adequate for most applications, there are times when it is not so suitable. Consider, for example, a data entry application where unskilled users must enter names, addresses, and numbers from forms through a screen for a program to write to a disk file. It would be nice if INPUT would allow the programmer to exclude all the undesirable keys (the menu, escape, blinks, erases, home, or down arrow) that can cause such harm if pressed. Programmers themselves can press those keys by mistake, and trying to get new users to realize the importance of NOT striking the wrong key is sometimes a tough job. (And understandable, from their point of view. If the computer doesn't want to see a menu key in the middle of typing in an address, why doesn't it just ignore it?) Another drawback to INPUT is that the user can type in 26 characters for a 25 character field. Nothing stops them. The program can generate an error message after the fact, but it would be much neater to inhibit input after the 25th character.

This article describes a CALLable assembly language subroutine that does just that. In addition, it can disallow entry of alpha data in fields defined as numeric, and has editing features such as the use of the insert/delete character and delete field (line) keys. There are two programs presented in this article. One of them, the assembly language subroutine, will not be described. If you're interested, it makes interesting reading; if not, just type it in and assemble it. The second program is a BASIC program that illustrates the use of the CALLable "FIELDS" assembly language routine. The assembly language program can be assembled by either assembler (see the **Editors' Notes** in this issue about the differences between the macro and regular assemblers), and that when RUN from FCS, it establishes all necessary linkages, and adjusts for different software versions.

The major divisions in the BASIC program should be evident. Lines 550-870 display the initial screen. Each input field is shown as a blue rectangle of appropriate size. For example, the first field - NAME - is displayed on line 10. The title of the field, NAME, identifying the data item, is displayed followed by a blue rectangle 12 bytes long. (Lines 630, 680.) Lines 880-1080 are the main body of the code, where each field is "read" from the screen. Lines 880-920 read the first field, NAME. Variables X and Y are set to the first position of the blue rectangle for NAME, ML is set to 12, the maximum field length, NA\$ is set to "A" indicating an

alphanumeric field, and the CCI set to white on blue. The control subroutine at line 100 (through 195) is called to interface to the machine language subroutine. The 100 subroutine POKEs parameter values for the assembly language routine into a known common location (right after the end of BASIC RAM), and CALLs the routine. On return, FI\$ is constructed from the field, byte by byte, and control returned to line 910. The variable KB is used to return the ending character of the field, and if a TAB key was pressed, the program "backs up" a field. Lines 1210-1220 are where the main field processing program would be. In this demo program, the entered field is simply written back on the same line along with its length.

When the assembly language program gets control after the CALL, it takes over all keyboard input and disallows any control characters and any "undesired" characters. It processes editing keys, such as cursor controls and character insert keys. The delete line key causes the field to be cleared from the cursor to the end of the rectangle. When the return key signifies the end of field entry, what is shown in the field will be what the program gets, unlike INPUT, which inputs only to the cursor. (In the assembly language program, changing the value of ALPHMAX to 126 will permit lower case letters.)

Although the routines here may be a little long, I'm sure you'll find them to be quite useful. They can greatly extend the use of your computer by making the "feeding" of it more friendly. ■

(Program listings begin on next page.)

CALL FOR ARTICLES

Colorcue gets its material from those who write it. It don't grow on no trees. Nevermind your bad spelling or badly grammar: that's the job of the Editors. You come up with the ideas, splash them onto paper (or disk), and we'll wrestle them into an intelligible form for publication. We can't pay you for your time--you won't become rich. But, since **Colorcue** is read all over the world (well, not **all** over, exactly), then maybe you'll become world famous.... We're looking for ideas, programs, reviews of books and/or software and/or hardware applicable to ISC machines, hardware/software modifications, user group news, and.... Well, you get the idea.

FOR SALE. 32K CompuColor II with 117 key keyboard and lower case. Version 8.79. Centronics 779-2 dot matrix printer with lower case and interface/cable. Novation CAT modem with cable and Com-Tronics TERMII communications software. Disks: Sampler, Basic Editing, Assembler, Text Editor, Star Trek, Personal Data Base, Blackjack, Cubic Tic-tac-toe, plus 25 additional disks. CompuColor Programming Manual. \$2200.00. Contact: Don Miller, 112 Marble Drive, Rochester, NY 14615 (716) 663-1175.

Listing 1. Assembly language listing for the protected fields routine.

```

;FIELD HANDLING ROUTINE WHICH
;ENABLES BASIC TO CREATE
;'PROTECTED' & 'UNPROTECTED' FIELDS
;ON THE SCREEN.
;
;AUTHOR. BERNIE RAFFE
;      HARROW
;      ENGLAND
;FEBRUARY 1981
;
;
;
; DATA AREA USED TO PASS PARAMETERS
;
COUNT: DB 0 ;CHARACTER COUNT
NA: DS 1 ;NUMERIC OR ALPHA;SET BY BASIC
CHARIN: DS 1 ;TEMPORARY CHARACTER STORE
FLDADR: DS 1 ;START OF FIELD;SET BY BASIC

MAXBAS EQU 32940 ;POINTER TO END OF BASIC RAM
VCALL EQU 33282 ;CALL VECTOR

; SUBROUTINE TO SET UP LINKAGES WHEN RUN FROM FCS
;
START: PUSH H ;SAVE SOME REGS
      PUSH D
      PUSH PSW
      LXI H,COUNT-1 ;SET END OF BASIC RAM POINTER
      SHLD MAXBAS ;TO 1 BYTE IN FRONT OF PGM
      MVI A,(JMP) ;SET UP JUMP TO OUR RTN IN CALL VECT.
      LXI H,CIIINIT ;SO CALL WILL GO TO CIIINIT
      STA VCALL
      SHLD VCALL+1 ;STORE ADDRESS
      LXI H,1C78H ;ASSUME LO ADDRESS IS IN V879

```

```

LDA 0001H ;CHECK
CPI 6CH ;IS IT?
JNZ STUFF ;YES
LXI H,3392H ;NO, CHANGE TO V678 ADDRESS
STUFF: SHLD LO+1 ;PUT PROPER ADDR. INTO INSTRUCTION
      POP PSW ;PREPARE TO EXIT
      POP D
      POP H
      MVI B,0 ;CLEAR B TO INDIC. NO ERR. TO FCS
      RET ;RETURN TO FCS

; LO LINKAGE
; VERSIONIZED BY START ROUTINE

LO: JMP OFFFFH ;JUMP TO LO ROUTINE IN MONITOR
      ;ACTUAL ADDRESS FILLED IN BY START

CR EQU 13 ;CARRIAGE RETURN
DELIN EQU 04 ;DELETE LINE KEY
INSERT EQU 05 ;INSERT KEY
DELETE EQU 127 ;DELETE KEY
HT EQU 9 ;HORIZONTAL TAB
RIGHT EQU 25 ;MOVE CURSOR RIGHT
LEFT EQU 26 ;MOVE CURSOR LEFT
SPACE EQU 32 ;SPACE CHARACTER
SLASH EQU 47
INPCRT EQU 81C5H ;JUMP VECTOR #31
KBDFL EQU 81DFH ;HOLD NO. OF JUMP
KBRDY EQU 81FFH ;KEYBOARD READY FLAG
      ;VECTOR FOR KEYBOARD

NUMMIN EQU 45 ;NUMERIC MINIMUM
NUMMAX EQU 58 ;NUMERIC MAXIMUM
ALPHMIN EQU 32 ;ALPHANUMERIC MINIMUM
ALPHMAX EQU 91 ;ALPHANUMERIC MAXIMUM

```

```

;THIS CHARACTER INPUT INITIALISATION
;ROUTINE SETS UP THE PARAMETERS
;NECESSARY FOR THE 'CHRINT' AND
;'CI' ROUTINES.

```

```

CIINIT: PUSH   PSW           ;SAVE REGISTERS REQ'STD BY BASIC
        PUSH   H
        LDA   KBDL          ;SAVE BASIC'S JUMP VECTOR #
        PUSH   PSW
        MVI   A,31         ;SETUP NEW JUMP VECTOR #
        STA   KBDL
        MVI   A,0C3H       ;PLACE 'JMP' AT VECTOR LOCATION
        STA   INPCRT
        LXI   H,CHRINT     ;GET ADDRESS OF 'CHRINT' ROUTN
        SHLD  INPCRT+1     ;PLACE ADDRESS AFTER 'JMP'
        XRA   A
        MOV   B,A
        STA   KBRDY
        STA   CHARIN
        STA   COUNT       ;RESET CHARACTER COUNT
        JMP   GETNXT

```

```

;CHRINT - THIS CHARACTER INPUT ROUTINE IS
;VECTORED TO FROM THE KEYBOARD INPUT
;ROUTINE THROUGH THE JUMP VECTOR (#31).
;THE CHARACTER FROM THE KEYBOARD
;ROUTINE IS IN REGISTER 'E'.

```

```

CHRINT: LXI   H,CHARIN     ;GET ADDRESS OF TEMP
;CHARACTER STORAGE.
        XRA   A           ;CLEAR ACCUMULATOR
        CMP   M           ;TEST FOR 'CHARIN' FOR ZERO
        JNZ  CFIN        ;IF NOT ZERO THEN IGNORE INPUT
        MOV   A,E         ;GET CHAR FROM 'E'
        ANI  127         ;STRIP UPPER BIT FOR ASCII
        MOV   M,A        ;PUT CHAR IN 'CHARIN'

```

```

CFIN:   EI             ;ENABLE INTERRUPTS
        RET            ;RETURN FROM INTERRUPT

```

```

;CI - THIS CHARACTER INPUT ROUTINE GETS A CHARACTER
;FROM THE TEMPORARY STORAGE LOCATION 'CHARIN'
;CLEARS THE KEYBOARD READY FLAG
;AND RETURNS WITH THE CHARACTER IN 'A'.
;IF THERE IS NO CHARACTER IN 'CHARIN',
;THEN 'CI' WILL HANG AND WAIT FOR ONE.

```

```

CI:     EI             ;ENABLE INTERRUPTS
CI10:   LDA   CHARIN     ;GET CHARACTER
        CPI   0          ;HAVE A CHAR?
        JZ   CI10       ;IF NOT,HANG FOR ONE
        PUSH  PSW        ;GOT ONE,NOW SAVE IT
        XRA   A
        STA   KBRDY     ;CLEAR KEYBOARD READY FLAG
        STA   CHARIN    ;CLEAR TEMP STORAGE FOR NEXT CHAR
        POP  PSW        ;RESTORE CHAR
        RET

```

```

GETNXT: CALL  CI        ;GET NEXT KEYBOARD CHARACTER
        MOV   D,A       ;STORE IN 'D'
        CPI   LEFT      ;BACKSPACE?
        JZ   LEFTKEY   ;JUMP TO BACKSPACE ROUTINE
        CPI   HT        ;TAB?
        JZ   MAINFIN   ;YES-JUMP TO END ROUTINE
        CPI   CR        ;RETURN?
        JZ   MAINFIN   ;YES - JUMP TO END ROUTINE
        CPI   RIGHT     ;CURSOR RIGHT
        JZ   RHTKEY    ;PUT IT ON SCREEN
        CPI   INSERT    ;INSERT A NEW CHARACTER
        JZ   INS10     ;
        CPI   DELETE    ;DELETE A CHARACTER

```

```

JZ    DEL10
CPI   DELINE ;DELETE LINE CHAR
JZ    DLIN10
MOV   A,B    ;FIELD FULL?
CMP   E
JZ    GETNXT ;YES SO IGNORE
LDA   NA     ;NUMERIC OR ALPHANUMERIC FIELD
CPI   65    ;A=ALPHA
JZ    ALPHA

```

```

NUMERIC:MOV  A,D    ;NUMERIC VALIDATION
CPI   NUMMAX ;RANGE TEST-MAXIMUM VALUE
JNC   GETNXT ;IGNORE
CPI   NUMMIN ;          MINIMUM VALUE
JC    GETNXT ;IGNORE
CPI   SLASH  ;= '/'
JZ    GETNXT ;IGNORE
JMP   KEYOK  ;ACCEPT KEY

```

```

ALPHA: MOV  A,D    ;ALPHANUMERIC VALIDATION
CPI   ALPHMAX ;RANGE TEST - MAXIMUM VALUE
JNC   GETNXT ;IGNORE
CPI   ALPHMIN ;          MINIMUM VALUE
JC    GETNXT ;IGNORE

```

```

KEYOK: CALL  LO    ;PUT IT ON SCREEN
      INR   B     ;INCREMENT FINAL COUNT
      JMP   GETNXT ;GET NEXT CHAR

```

```

MAINFIN: MVI   A,3    ;PUT CURSOR OFF SCREEN
      CALL  LO
      MVI   A,64
      CALL  LO
      XRA   A
      CALL  LO
      CALL  CNT10 ;SETUP FIELD COUNT FOR BASIC

```

```

STA   COUNT
MOV   E,D    ;PUT LAST CHAR IN 'E'
XRA   A
MOV   D,A
POP   PSW    ;RSTORE BASIC JUMP VECTOR #
STA   KBDLFL
POP   H     ;RESTORE USED REGISTERS
POP   PSW
RET

```

```

LEFTKEY: MOV  A,B    ;BACKSPACE SUBROUTINE
      CPI   0     ;IF ON FIRST CHAR
      JZ    GETNXT ;THEN IGNORE
      DCR   B     ;DECREMENT CHAR COUNT
      MVI   A,LEFT ;BACKSPACE CURSOR
      CALL  LO
      JMP   GETNXT

```

```

RGHTKEY: MOV  A,B    ;RIGHT CURSOR FUNCTION
      CMP   E     ;CHECK IF FIELD FULL
      MOV  A,D    ;RESTORE CHARACTER
      JNZ   KEYOK ;NO SO O.K.
      JMP   GETNXT ;IGNORE

```

```

;INSERT A CHARACTER ROUTINE....
;REGISTER USAGE:-
; A=NO OF CHARS ALREADY IN FIELD
; B=POSITION WITHIN FIELD
; C=NO OF CHARS TO SHUFFLE
; E=MAX NO OF CHARS IN FIELD

```

```

INS10: CALL  CNT10 ;DETERMINE SIZE OF FIELD RTN
      CPI   0     ;IGNORE IF NO CHARS IN FIELD
      JZ    GETNXT
      PUSH  D

```

```

LHLD  FLDADR ;SET UP HL                ;DELETE A CHARACTER ROUTINE.....
DCX   H      ;REGISTER USAGE:-
DCX   H      ; A=NO OF CHARACTERS ALREADY IN FIELD (AFTER CNT10 CALL)
SUB   B      ;CALCULATE NO OF CHARS   ; B=POSITION WITHIN FIELD
JM    INS55  ;RETURN IF PAST LAST CHAR (NAUGHTY!) ; C=NO OF CHARACTERS TO SHUFFLE
JZ    INS55
MOV   C,A    ;PUT IN C
ADD   B      ;RESTORE A
CMP   E      ;IS FIELD FULL
JNZ   INS20  ;NO
DCX   H      ;YES - SO ARRANGE FOR LAST CHAR
DCX   H      ;TO DISAPPEAR FROM FACE OF EARTH
DCR   C
JZ    INS55  ;IGNORE IF SITTING ON LAST CHAR
INS20: MVI   D,0 ;POSITION HL TO LAST CHAR
      MOV   E,A
      DAD   D
      DAD   D
      LXI  D,OFFFCH ;TO SUBTRACT 4 LATER ON

INS30: MOV   A,M ;GET CHAR FROM FIELD
      INX  H
      INX  H
      MOV   M,A ;AND SHIFT UP

INS40: DCR   C ;ANY MORE?
      JZ    INS50 ;NO
      DAD   D ;YES POINT TO PREVIOUS CHAR
      JMP  INS30

INS50: DCX   H ;PUT A SPACE AT CURSOR POSITION
      DCX   H
      MVI  M,SPACE

INS55: POP   D
      JMP  GETNXT ;RETURN

```

```

DEL10: CALL  CNT10 ;DETERMINE SIZE OF FIELD
      CPI   0 ;IGNORE IF NOTHING IN IT
      JZ    GETNXT
      PUSH D
      LHLD FLDADR ;SETUP HL
      INX  H
      INX  H
      SUB  B ;CALCULATE NO OF CHARS
      MOV  C,A ;TO SHUFFLE & PUT IN C

DEL20: MOV   E,B ;POSITION HL TO FIRST CHAR TO SHUFFLE
      MVI  D,0
      DAD   D
      DAD   D
      LXI  D,4 ;TO ADD 4 LATER ON

DEL25: DCR   C
      JM    DEL55 ;RETURN IF PAST LAST CHAR
      JNZ  DEL30 ;SPECIAL TEST FOR LAST CHAR
      DCX  H
      DCX  H
      JMP  DEL53

DEL30: MOV   A,M ;GET CHAR FROM FIELD
      DCX  H
      DCX  H
      MOV  M,A ;AND SHIFT UP

DEL40: DCR   C ;ANY MORE?
      JZ    DEL50 ;NO

```

```

DAD D ;YES - POINT TO NEXT ONE
JMP DEL30

DEL50: INX H ;PUT A SPACE AT LAST
INX H ;CHARACTER POSITION
DEL53: MVI M,SPACE

DEL55: POP D
JMP GETNXT ;RETURN

;ROUTINE TO DELETE ALL THE REMAINING
;CHARACTERS IN THE FIELD

DLIN10: CALL CNT10 ;GET NO OF CHARS IN FIELD
PUSH D
LHLD FLDADR ;START OF FIELD
SUB B ;CALCULATE NO.OF CHARS TO DELETE
JM DLIN40 ;IGNORE IF AT END
JZ DLIN40
MOV C,A

DLIN20: MVI D,0 ;SET H&L TO 1ST CHAR TO DELETE
MOV E,B
DAD D
DAD D
MVI A,SPACE

DLIN30: MOV M,A ;DELETE CHARS
DCR C ;ANY MORE?
JZ DLIN40 ;NO
INX H
INX H
JMP DLIN30

```

```

DLIN40: POP D
JMP GETNXT

;ROUTINE TO DETERMINE THE EXACT LENGTH
;OF A FIELD ON THE SCREEN.
;FINAL COUNT IS PUT IN 'A'.

CNT10: PUSH B ;SAVE IT
LHLD FLDADR ;START OF FIELD
DCX H
DCX H
MVI B,0

CNT20: MVI C,0 ;C WILL CONTAIN NO OF SPACES
;PAST THE END OF THE FIELD

CNT30: MOV A,B ;B CONTAINS THE FIELD COUNT
CMP E ;ARE WE AT MAXIMUM
JZ CNT40 ;YES - FINISHED
INX H ;GET TO NEXT CHAR
INX H
INR B ;ADD 1 TO COUNT
MOV A,M
CPI SPACE ;IS IT A SPACE
JNZ CNT20 ;NOPE
INR C ;INCREMENT SPACE COUNT
JMP CNT30 ;BACK & SHUFFLE NEXT ONE

CNT40: MOV A,B ;CALCULATE EXACT COUNT
SUB C
POP B ;RESTORE USED REGISTER
RET

END START

```

Listing 2. A BASIC program which demonstrates the fields machine language patch. Note: numbers in brackets are control codes entered from the keyboard. Thus, [16] is black; [17] is red; [29] is foreground on; etc.

```

1 GOTO 500
100 REM [10][10]FIELD HANDLING SUBROUTINE[10][10]
105 REM [22]X = X CO-ORDINATE[18]
110 REM [22]Y = Y CO-ORDINATE[18]
115 REM [22]ML = MAXIMUM INPUT LENGTH[18]
120 REM [22]NA$= 'N' FOR NUMERIC OR 'A' FOR ALPHANUMERIC[18]
125 REM [22]KB = KEYBOARD CHARACTER[18]
130 REM [22]FI$=FINAL STRING[18]
135 REM [22]SA = SCREEN ADDRESS OF FIELD[18]
140 PO=PEEK(32940)+PEEK(32941)*256+1:REM POINT TO PARAMETER PASS AREA
145 SA=28672+128*Y+X*X:REM SCREEN ADDRESS
150 ZZ=INT(SA/256):POKE PO+3,SA-256*ZZ:POKE PO+4,ZZ
155 SA=SA-2
160 POKE PO+1,ASC(NA$):REM ALPHANUMERIC OR NUMERIC
165 FI$=""
170 PLOT 3,X,Y
175 KB=CALL(ML)
180 IF KB=9 OR PEEK(PO)=0 THEN RETURN
185 FOR I=2 TO PEEK(PO)*2 STEP 2
190 FI$=FI$+CHR$(PEEK(SA+I)):NEXT I
195 RETURN
500 REM [10][10]PROGRAM INITIALIZATION[10][10]
520 CLEAR 100
530 PLOT 27,4
540 PRINT "RUN FIELDS"
545 REM [10]SET UP SCREEN[10]
550 PLOT 27,27,27,24
560 PLOT 12,3,5,1,14
570 PRINT "[29][22]SAMPLE PROGRAM TO DEMONSTRATE THE USE OF '[17]FIELDS[22]'[18]"
580 PLOT 3,0,5,11,3,5,5,15
590 INPUT "[19]ENTER '[22]I[19]' TO INSERT OR '[22]U[19]' TO UPDATE [22]";AS
600 IF AS<>"I" AND AS<>"U"GOTO 580
610 PRINT "[17]
      _[18]"
620 FOR I=1 TO 20:PLOT 10,11:NEXT I
630 PLOT 3,1,10:PRINT "[23]NAME[18]"
640 PLOT 3,1,13:PRINT "[23]ADDRESS[18]"
650 PLOT 3,1,16:PRINT "[23]MONEY OWING[18]"

660 PLOT 3,1,19:PRINT "[23]DESCRIPTION[18]
670 IF AS="U"GOTO 730
680 PLOT 3,14,10:PRINT "[30][20] [16]"
690 PLOT 3,14,13:PRINT "[20] [16]"
700 PLOT 3,14,16:PRINT "[20] [16]"
710 PLOT 3,14,19:PRINT "[20] [16][29]"
720 GOTO 770
730 PLOT 3,14,10:PRINT "[29][22][30][20]BERNIE RAFFE[16]"
740 PLOT 3,14,13:PRINT "[20]HARROW , ENGLAND [16]"
750 PLOT 3,14,16:PRINT "[20]534.43-[16]"
760 PLOT 3,14,19:PRINT "[20]AVERAGE[16][29]"
770 PLOT 3,0,22
780 PRINT "[22]SUMMARY OF FACILITIES--[18]"
790 PLOT 3,0,24
800 PRINT "[19] 1) CONTROL CHARACTERS & CURSOR POSITIONING KEYS (EXCEPT[18]"
810 PRINT "[19] LEFT & RIGHT CURSOR) ARE DISABLED.[18]"
820 PRINT "[19] 2) CURSOR WILL NOT TRAVEL OUT OF A FIELD[18]"
830 PRINT "[19] 3) FINAL FIELD VALUE DEPENDS ON CONTENT OF FIELD ON THE[18]"
840 PRINT "[19] SCREEN & NOT ON THE ACTUAL KEY DEPRESSIONS[18]"
850 PRINT "[19]4) 'DELETE/INSERT CHAR' & 'DELETE LINE' WORK WITHIN AFIELD[18]"
860 PRINT "[19] 5) ALPHABETIC CHARACTERS ARE DISABLED IN NUMERIC FIELDS[18]"

870 PRINT "[19] 6) THE 'TAB' KEY RETURNS CURSOR TO THE PREVIOUS FIELD[18]"
875 REM [10][10]GET EACH FIELD[10][10]
880 X=14:Y=10:ML=12:NA$="A"
890 PLOT 6,38
900 GOSUB 100
910 IF KB=9GOTO 900
920 GOSUB 1200
930 X=14:Y=13:ML=17:NA$="A"
940 PLOT 6,38
950 GOSUB 100
960 IF KB=9GOTO 880
970 GOSUB 1200
980 X=14:Y=16:ML=7:NA$="N"
990 PLOT 6,38
1000 GOSUB 100
1010 IF KB=9GOTO 930
1020 GOSUB 1200
1030 X=14:Y=19:ML=7:NA$="A"
1040 PLOT 6,38
1050 GOSUB 100
1060 IF KB=9GOTO 980
1070 GOSUB 1200
1080 PLOT 3,15,21:INPUT "[17]HIT RETURN FOR ANOTHER GO [18]";AS
1085 REM [10]CLEAR SCREEN AND DO IT AGAIN[10]
1090 PLOT 3,0,8:FOR I=1 TO 23:PLOT 10,11:NEXT I
1100 GOTO 580
1200 REM [10][10]FIELD PROCESSING ROUTINE. THIS JUST PRINTS VALUE;[10]REAL PRO
      GRAM WOULD PROCESS FIELD HERE
1210 PLOT 6,2,3,33,Y:PRINT "[29][17]";FI$;" ([29][22]";LEN(FI$);"[29][17])[29][18]";"
1220 RETURN

```

Serial to Parallel Interface

by Ben Barlow

The Epson line of printers is certainly one of the most popular ones in the personal computer industry. Their combination of features (including high resolution bit-level graphics), high reliability, and low price are hard to beat. Have you noticed the low-end printers being offered by IBM and HP on their lines of personal machines? OEMed Epsoms. Many ISC and CompuColor owners have added Epson printers to their computers, and several of them, as I did, were probably surprised to find that the Graphics package (GRAFTRAX-80) does not work with the low cost serial interface board option in the printer; the \$150 buffered serial interface is required for GRAFTRAX. Having chosen the Epson at least partly because of price, and badly wanting the graphics option, I was distressed to think of \$150 fleeing after the \$75 (inexpensive interface) after the \$90 (graphics). At that rate, the options would soon exceed the basic printer's cost. This article is about a cheaper solution--one that you might want to consider as an alternative to either of the serial boards: a serial to parallel converter that allows the computer (serial) to talk to the Epson in its own tongue (parallel). (After writing this article, I came across an article in the Sept. 1981 issue of **BYTE** magazine by Steve Ciarcia on the construction of an unlimited-vocabulary speech synthesizer that connects to any computer through a parallel interface. So there may be value in this article even if you don't have a printer.)

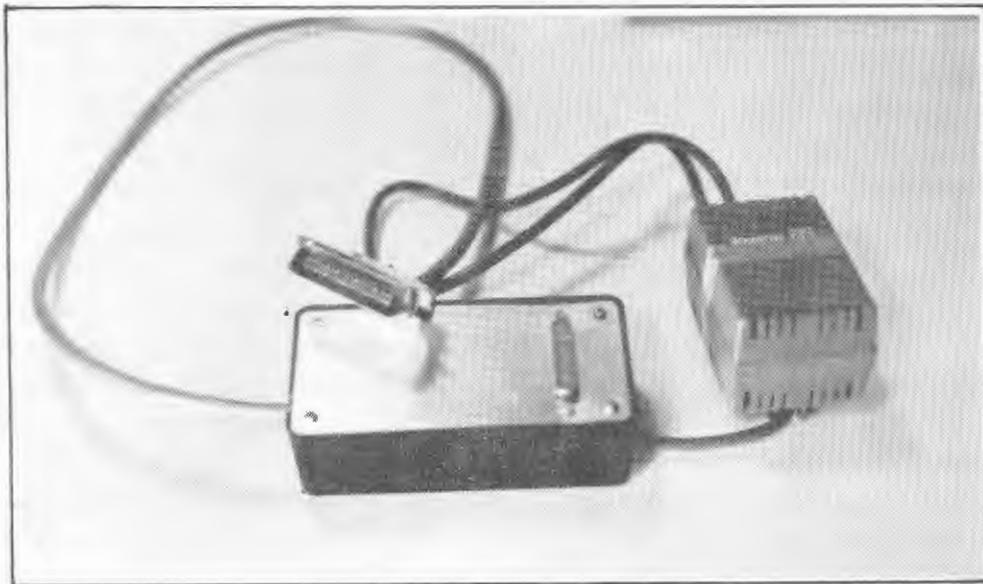


Photo 1.

The converter is a low cost device based on a 5V UART which accepts data at 9600 BPS (Bits Per Second) from the computer, turns them into a parallel byte, and strobes them into the Epson's parallel interface. It accepts the BUSY signal from the printer and passes that back to the computer as not clear-to-send to control data flow. It's constructed with only four IC chips and a handful of other parts, and with judicious shopping and no junk box to pull from, it can be built for about \$45. With any kind of parts box to paw through, the cost can go down significantly. Construction is relatively simple, and the converter can be a rewarding project for the winter months.

Looking in more detail at the operation of the converter, it may help to look at the schematic. The AY-3-1014A UART is one of several chips which are adequate; choosing one that operates on a single 5 volt supply simplifies the process, though those chips are more expensive than the +5 and +12 ones. The 1014 is set up with its selectable options determined by connecting option selecting pins 34, 35, 36, 37 to +5V. Pin 21 must be grounded, and pin 4 must be low to place received data on the output lines. Bits of a byte are clocked into the 1014 under the control of the Receiver Clock (RCP = pin 17). When a serial byte has been completely received, the 1014 puts it onto the output lines RD1 through RD8, and raises the DAV signal on pin 19 to indicate that a byte is ready. DAV is used to generate a short duration pulse which strobes the data on the output lines into the printer's data buffer. The 74123 chip (a 74121 would have been more appropriate, but I didn't have one) provides this pulse, which is about 500 nanoseconds long.

The printer returns the ACKNLG* ("*" indicates active low) signal which is routed to the 1014 on pin 18, RDAV*. The 555 timer chip provides the clock for the 1014, which must be at 16 times the baud rate, or 153.6K (9600 x 16). Although this is outside the 555's specification (100K is its published limit), it seems to produce an adequate (though not quite square) pulse train. (I experimented with CMOS oscillators with no luck. A baud rate generator chip, which would be the professional way to get a clock, would have added about another \$15-\$20 to the low-cost project.) An adjustable potentiometer adjacent to the 555 provides the necessary degree of adjustment of clock frequency to compensate for temperature or burn-in. (My clock seemed to drift for a couple of days until the capacitor apparently stabilized. Since then it's been fine, and unadjusted.) The 1489 chip accepts RS232C signal levels, converting them to TTL levels and inverting them. Serial data comes from the computer at an RS232C level, and is routed through the 1489 before feeding to the 1014. The BUSY signal from the printer also is routed through the 1489, but only for its inverter function. (The BUSY* signal thus supplied works with ISC computers and Compucolors containing the ECN 002137 published in the Aug/Sept **Colorcue**.)

The converter consists of three separately constructable pieces: the interface itself, a 5V power supply, and a printer cable. The interface pictured (photo 1) was built on a small piece of vector board using wire wrap methods. A 16-pin socket was wired in as the parallel cable connector, allowing the cable to be removed from the interface itself, a simplifying factor during construction. I've gotten into the habit on wire-wrap projects of providing labelled test pins for +5V and GND, so that I can easily attach logic probes or meters without crowding the components needlessly. The potentiometer for the 555 should be a multi-turn one, possibly with a series resistor as shown in the schematic, depending on the values you can locate. Having the adjustment screw available from the outside (notice the hole in the case in photo 2) simplifies adjustment somewhat. The power required is low, but I chose not to pull it from the computer. It's not readily accessible there, nor did it seem esthetically correct. My junk box had some non-working calculator power supplies that came from a surplus store as fantastic bargains a year or two back, and modification of one of those proved to be easy. For someone without that type of junk, a 6 or 9V transformer with regulation to 5V on the converter board would be an inexpensive solution. There's room on the board for the required regulator and capacitor.

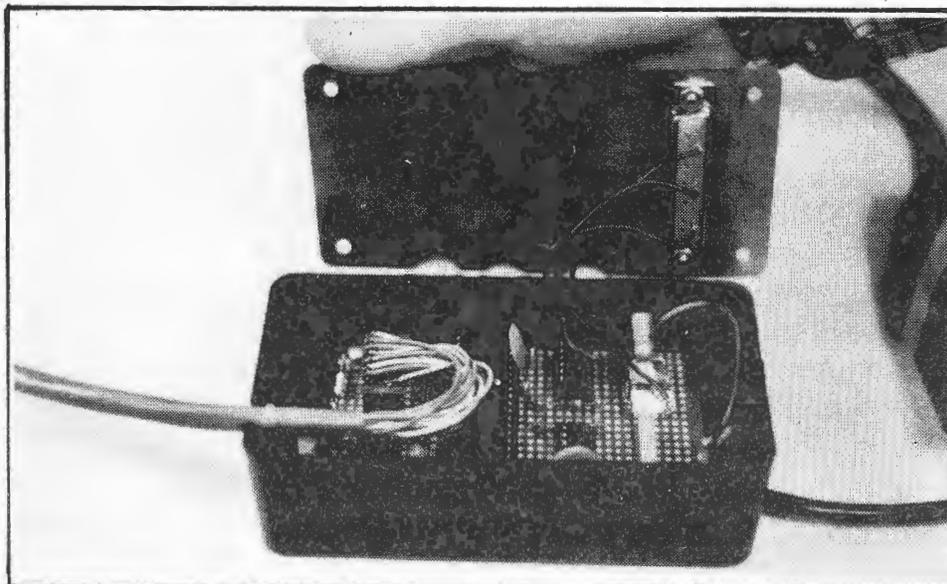


Photo 2.

Cable construction is straightforward. The special connector for the parallel interface is soldered on one end, a 16-pin DIP header on the other. The only trick is to carefully map pin to pin. Although I used a hunk of 12 conductor wire from under the workbench, ribbon cable or separate wires should work as well. Remember to connect pins 35 and 31 on the parallel connector.

Once built, you'll want to check the wiring (and, if you're like me, recheck it). An easy way to do this is to check (with no chips in the sockets) each pin, each wire, with a voltmeter set to measure resistance--make a series of continuity checks. If all the wiring checks out, check the power supply wiring, and if it looks all right, plug it in and hope for 5V. Check power and ground pins on each socket on the board, check the cable, and with the power off, plug in the chips. If it seems to be working (as opposed to smoking), check the 555's output with an oscilloscope. (If you don't have one, don't worry. The following procedure works as well.) Set the pot so that the 555's pulse width is 6.5 microseconds. Or, write a small BASIC program, such as:

```

10 PLOT 27,13
20 PRINT "ABCDEFGHJKLMNOPQRSTUVWXYZ////////TEST....TEST.
...TEST----TEST----TEST"
30 GOTO 20

```

Connect the computer's RS232 cable to the interface, connect the printer, power everything on (all the connections were made with power off, right?), and RUN the program. Adjust the pot until the printer prints text, not garbage. Find the middle of the "print" range, in between garbage settings, and leave it there. You may have to adjust it again, but if it acts like mine, it will settle in like a rock after a day or two.

In summary, if you are willing to trade a little time for money, and enjoy (or want to experiment with) the hardware side of the business, you'll find this project appealing. It's a simple, low cost, safe interface.

Parts List

- | | | |
|------------------------|----|-----------------------------------|
| Small box | 1 | 50K multi-turn pot. |
| 2.5" x 5" Vector board | 1 | DB-25 female soldertail connector |
| 1 8 pin WW socket | | misc. wire (stranded 22 ga. & WW) |
| 1 14 pin WW socket | | .025" square post terminals |
| 2 16 pin WW socket | | ----- |
| 1 40 pin WW socket | 1 | Amphenol 36 pin connector for |
| 1 LM555 IC | | Centronics-type printers |
| 1 MC1489 IC | 1 | length 12 conductor cable (or |
| 1 74123 | | substitute) |
| 1 AY-3-1014-A | 1 | 16 pin DIP header |
| 2 50 pF capacitor | | ----- |
| 2 .1 mF capacitor | 5V | power supply or: |
| 1 10 mF capacitor | | 6 or 9V DC transformer |
| 1 1K Ohm resistor | | 78M05 regulator |
| 1 20K Ohm resistor | | 1000mF 35V capacitor |
| 1 56K Ohm resistor | | |

RENAISSANCE MARKETING ANNOUNCES

Low Cost Business Programs
for your CompuColor model 4, model 5
and Intecolor 3600 series Computers

GENERAL LEDGER* (16K, 32K) \$ 59.95

-
- (1) ACCOUNT PROGRAM: Display account data, list all accounts, add account, delete account, change account data.
 - (2) JOURNAL PROGRAM: Enter journal data, display journal data by entry number, change account data.
 - (3) PROOF PROGRAM: Display or print journal entry proof sheets with titles.
 - (4) POST PROGRAM: Applies journal entries to accounts.
 - (5) REPORT PROGRAM: Print balance sheet, print income statement with titles.

INVENTORY CONTROL* (16K, 32K)\$ 34.95

Gives the following reports:

- (1) Display item data by quantity.
- (2) Print or display all items on file.
- (3) Display item data by class code.
- (4) Print or display item data by vendor.
- (5) Updating section:
 - a. add new item
 - b. update item quantity
 - c. change item data
 - d. delete item
- (6) Display item data by item number.

Provides quantities and data for 750 items or models.

MAXELL MINI DISK for CompuColor. Box of 10\$ 34.95

30% OFF all CompuColor Corp. software in stock. **CALL**

RENAISSANCE MKT.

7 So. Pierson Rd.
Maplewood, NJ 07040
201-762-0585

In N.J. add 5% sales tax

Ship to: _____

- [] General Ledger \$59.95
- [] Inventory Control \$34.95
- [] Maxell Disk Box Qty. _____

Terms: payment with order

Freight: Prepaid in USA

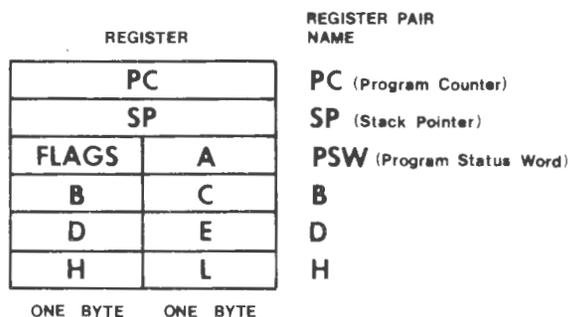
* Available for Intecolor Feb., Mar. 1982 Taking orders now.

Assembly Language Programming

PART III: MORE ON THE 8080 INSTRUCTION SET
by DAVID B. SUITS

Last time we used ICS's Machine Language Debug Package in order to learn about a few of the 8080 machine language instructions (or "operation codes", or just "op codes", as they are often called). We saw how to put an eight bit number into the accumulator and then CALL a certain subroutine contained in your computer's ROM in order to put something onto the screen. This was just like BASIC's PLOT statement. In BASIC, you can specify a whole string of PLOT numbers just by separating them with commas. For example, PLOT 2,0,0,242,127,0,127,127,0,127,0,0,255 will draw a line around the screen. Being able to separate the numbers with commas makes things a lot easier than having to say PLOT for each number: PLOT 2: PLOT 0: PLOT 0: PLOT 242 ... etc. Can we do something like that in assembly language so that we don't have to load the accumulator each time with the next byte and CALL the subroutine? Yes. We will investigate a method which works something like BASIC's DATA and READ statements.

As we learned last time, the MOV instruction tells the 8080 to move (or, really, copy) the contents of one register into another. The destination register is specified first, and then the source register. For example, MOV A,E will copy the contents of the E register into the A register (the Accumulator). The diagram shows the registers. But there is



a register (well, sort of a register) which is not shown: register M. 'M' stands for Memory. Although memory locations are surely not registers in the 8080 chip itself, the 8080 is cleverly designed so that we can sometimes treat them as if they were. But certain conditions must be fulfilled. Specifically, we can call that memory location whose address is contained in the HL register pair a register. Thus, MOV A,M is an instruction to copy the contents of the memory location into the accumulator. Which memory location? Why, the one whose address is contained in the HL register pair.

For example, if the HL pair contained the 16 bit address 00111111 00001110 (which is 3FOEH in hex), then MOV A,M would copy the contents of memory location 3FOEH into the accumulator. The instruction MOV M,E would copy the contents of register E into memory location 3FOEH. And so on. There are other ways of getting data to and from memory, but this is a handy one which will suit our present purposes.

How do we set up the HL pair so that it contains the address of the memory location we want? We can use the MVI instruction which we dealt with last time. Remember, MVI <reg>, <num> will put <num> into <reg>. If we want 3FOEH in the HL pair, then we could write:

```
MVI H,3FH    or    MVI L,0EH
MVI L,0EH    MVI H,3FH
```

(Note that I have explicitly specified that each of the numbers is hex by using the capital 'H'. Although the MLDP assumes that all numbers are in hex, your assembler doesn't: it assumes that everything is in decimal unless followed by 'H' for hex. Since we will be dealing with the assembler eventually, we might as well get used to this way of doing things. Besides, specifying 'H' when hex is meant will help avoid confusion. Of course, for numbers less than 0AH, it won't matter: 1 decimal = 1 hex, 2 decimal = 2 hex, and so on up through 9.) But there is a more convenient method of loading a register pair with two bytes, and that is: **LXI <reg pair>, <num>**. The 'LXI' stands for Load eXtended Immediate. 'Load' in this case is the same as 'move'. So this is just like the MVI (MoVe Immdiate) instruction, except that 'extended' refers to a register pair instead of just one register. The number which is to be loaded into the specified register pair will be interpreted as two bytes. Thus, LXI H,3FOEH is an instruction to load the register pair HL with the two bytes of immediate data, 3FOEH. And LXI H,2 will load 0002H into the HL pair.(The 'H' in the LXI H,... instruction will not be confused with the single register H, because the LXI instruction requires reference to a register pair. 'H' in this case stands for 'HL'. 'B' would stand for 'BC', and 'D' would stand for 'DE'. 'SP' would stand for 'Stack Pointer'. The Accumulator and FLAGS together constitute a register pair only in certain circumstances, and this is not one of them.)

That's fine, but we need some more tools in order to do what I want us to be able to do. Enter the increment command. Any register, or register pair, may be incremented by one by the use of a single command. To increment a single register, **INR <reg>** will do. For example, if register B contains 4, then after INR B it will contain 5. If the Accumulator contains 11111111B (=OFFH, =255 decimal), then what will be in the Accumulator after INR A? OFFH+1=100H, which is 100000000 binary. But that's 9 bits long, and the

Accumulator, like all the other single registers, can hold only 8 bits. As a result, the left-most bit is lost; the Accumulator will now have zero in it.

If we wish to increment a register pair, the INR instruction just won't do. Suppose the HL pair contains 00FFH, that is, H holds 00 and L contains 0FFH. Then INR L will result in H=00 and L=00. But if we want to increment the HL register pair, then we want 00FFH+1 to yield 0100H; that is, H should contain 01 and L should contain 00. In this case we use the INX instruction: INcrement eXtended. Thus, INX H will increment the register pair HL; INX B will increment the register pair BC; and so on. (Corresponding to the increment and increment extended instructions, there are the decrement and decrement extended instructions. DCR A will subtract one from the Accumulator. DCX D will subtract one from the register pair DE. And so on.)

Where does all this get us? Well, now we can store all our PLOT numbers in a section of memory. Then we can load the HL register pair with the address of the start of that section of memory. That is, the HL pair will act as a **pointer**. Each time we wish to read a number from memory into the Accumulator, we use the MOV A,M instruction. Remember, that instruction gets the contents of the memory location which is pointed to by (i.e., whose address is in) the HL register pair. And then to output what is in the accumulator to the screen, we do what we did last time: CALL 33H. To get the next number, we simply make HL point to the next memory location by using the INX H instruction. Then we loop back to the MOV A,M. And so on. Something like this:

```
MOV A,M      ;Get the number from memory.
CALL 33H     ;Put it onto the screen.
INX H        ;Point to next location in memory.
;(Now loop back to the MOV A,M instruction.)
```

(Notice, by the way, that your assembler will treat a semicolon just as BASIC treats "REM".)

Of course, we need something more. We need a LOOP. We want to execute those three instructions a certain number of times. How do we make a loop? We need a counter to keep track of the number of times we have gone through the loop. And we need a conditional branch instruction to go back to the start of the loop in case the counter hasn't yet finished counting. In BASIC, the loop is handled in the familiar way using a FOR-NEXT structure. Unfortunately, the 8080 doesn't understand either 'FOR' or 'NEXT'. How could we make a loop in BASIC if we didn't have the FOR-NEXT structure? Perhaps like this:

```

10 COUNT=1           :REM INITIALIZE COUNTER.
20 blah             :REM HERE'RE THE STATEMENTS
30 blah             :REM   WHICH OCCUR INSIDE
40 blah             :REM   THE LOOP.
50 COUNT=COUNT+1   :REM INCREMENT COUNTER.
60 IF COUNT<8 THEN 20

```

In that example, we repeated a loop 7 times. Alternatively, we could count down instead of up:

```

10 COUNT=7
20 blah
30 blah
40 blah
50 COUNT=COUNT-1
60 IF COUNT>0 THEN 20

```

As it happens, this is a perfectly natural and easy way for the 8080 to handle a loop. For the counter, we could use, say, register C. We initialize register C to 7 with MVI C,7. And we decrement the counter each time through the loop with DCR C. But how do we test for zero and go back to the beginning of the loop? In BASIC, you jump around with GOTO statements. If you want to GOTO some place unconditionally, you simply write GOTO xxxx. But if you want to GOTO some place only when a certain condition is satisfied, you have to put in a test: IF such-and-such THEN GOTO xxxx. The 8080 has its unconditional GOTO instruction, too, only it's called a JUMP instruction. JMP xxxx will cause the program to go to (jump) to address xxxx. (It jumps to an address in memory, rather than to a line number.) But in addition, the 8080 has quite a few conditional jump instructions, all of which take place automatically by testing the status of one or another of the bits held in the FLAGS register. One of those bits is called the zero bit, and that is set (=1) whenever the result of certain operations end up as zero. Some operations affect that flag bit, and some do not. (The DCR instruction does.) The conditional jump instruction we want is: JNZ <adr>, or Jump if Not Zero to address <adr>.

Now we are ready to put together a program. Just as an example, let's erase the page and then plot a line around the screen. That will require 13 plot numbers, just as in BASIC: 12, 2, 0, 0, 127, 0, 127, 127, 0, 127, 0, 0, 255. Let's put these numbers (the data) at some convenient but out of the way location in memory; say, at 9000H. Get out your MLDP program and type

```
DBG>@9000
```

and then enter those 13 numbers in this fashion:

```
MEM>=#12  
MEM>=2  
MEM>=0
```

```
.  
.  
etc.
```

NOTE: If you have a 16K machine, your MLDP program may have a bug in it. Please read last issue's **Editors' Notes** and make the necessary correction, otherwise your efforts here will be in vain.

SECOND NOTE: Remember that you place numbers into memory using the "=" command. Whatever number you type in will be assumed to be hexadecimal unless preceded by "#". Since the numbers 0-9 decimal equal the numbers 0-9 hex, the "#" won't be necessary in such cases. That's why I didn't bother with it in the example above.

And now for the program to transfer those 13 bytes onto the screen. We can put the program anywhere (but not, of course, where it would interfere with the data). Let's put it at 8200H.

<u>Address</u>	<u>Contents</u>	<u>Assembly Language</u>	<u>Comments</u>
8200H	21H	LXI H,9000H	;Point to start of data.
8201H	00H		
8202H	90H		
8203H	0EH	MVI C,0DH	;Initialize counter.
8204H	0DH		
8205H	7EH	MOV A,M	;Get next byte of data.
8206H	0CDH	CALL 33H	;'Plot' it.
8207H	33H		
8208H	00H		
8209H	23H	INX H	;Point to next byte of data.
820AH	0DH	DCR C	;Done with all bytes?
820BH	0C2H	JNZ 8205H	;Not yet.
820CH	05H		
820DH	82H		
820EH			;Yes. All done!

Notice that two byte numbers are always stored low byte first, then high byte. That's why the contents of locations 8201H-8202H, 8207H-8208H and 820CH-820DH are the way they are.

There are two ways of entering this program into memory using the MLDP. You can load each memory location with the necessary hex values. For example:

```
MEM>@8200
MEM>=21
MEM>=00
MEM>=90
MEM>=0E
```

```
.
.
etc.
```

Or you can just type in the assembly language mneumonics (but not the comments):

```
MEM>@8200
MEM>LXI H,9000
MEM>MVI C,#13
MEM>MOV A,M
```

```
.
.
etc.
```

That is, the MLDP will act as a mini-assembler and translate the assembly language code into the required hex values for you.

Don't forget to put a BREAKPOINT right after the program:

```
MEM>/
DGB>AT 820E
```

And now you can execute the program, either full speed with

```
DBG>R 8200
```

or else in the interpreted mode with

```
DBG>I 8200
```

NEXT TIME: Some more 8080 instructions, a look at the Status Flags, and still more ways of putting data onto the screen. In the meantime, have you been reading anything and everything you can get your hands on regarding 8080 programming? That's the way to learn. Our sessions here cannot hope to cover everything; I expect you to be doing some homework. **■**

'THE' BASIC Editor

Reviewed by DAVID B. SUITS

The BASIC in ROM in your machine has some rudimentary editing capabilities: if you wish to make some changes to your BASIC program, you needn't write the entire program over again. But a more sophisticated editor would be a great boon to BASIC programmers. That's why I was so excited when ISC released 'FREDI', their BASIC editor, some time ago.

But now a newer, even more powerful editor is available, written by M. A. E. Linden and distributed by Quality Software Associates. It is called 'The' BASIC Editor. It does all that FREDI does and much, much more besides. Like FREDI, 'The' Editor lives in the top of your computer's memory; it takes up only slightly more room than FREDI. Unlike FREDI, 'The' Editor is a screen editor: instead of editing one line (shown at the top of the screen) as FREDI does, 'The' Editor allows you to move a cursor up and down the screen and then move into a line in order to edit it. You can copy one line or a block of lines into a different part of the program. You can delete a line or a block of lines; scroll the screen up or down; insert one or more lines (with auto line numbering); and search for a specified string.

The good news does not stop there. 'The' Editor will also renumber your program. And it will restore a lost program. (Ever hit ESC W and RETURN and then regret it? Not to worry!) It keeps you informed about the amount of memory you have left. It will LOAD a BASIC file for you. 'The' Editor will also politely disappear in case you don't want it around any longer.

But perhaps the most spectacular feature of this spectacular editor is its ability to append a program on disk to the one you have in memory! What's more, it will put the appended program starting at any line number you wish, resolving all the GOTOs and GOSUBs. This means that you can have a library of routines on disk and use 'The' Editor to append any one or more of them as and where you need them.

The version we received was an early one. Although the upgraded version was not available at press time, it should be available as you read this. The upgrade will have typomatic, it will list the program's variables (including arrays), and it will have a few other esoteric goodies which will boggle your mind.

'The' Editor is priced at \$49.95 (US), and, in my opinion, it is well worth every cent of that. It comes with an extensive manual, and if future changes are made, purchasers receive a free upgrade. If you are more than a very casual BASIC programmer, this is one software tool you will be overjoyed with. 'The' Editor works with V6.78, V8.79 and V9.80 systems, either 16K or 32K. (It will even be available very soon in a ROM version.) Contact:

Quality Software Associates
21 Dersingham Crescent
Thornhill, Ontario,
Canada, L3T 4P5. 

HOWARD ROSEN, INC.

Put the finishing touches to your Compucolor II or ISC computer.

- > Come up to the world of word processing.
- > Extend the utilization of your computer to the other members of your family.
 - * Letters
 - * School reports
 - * Business reports
 - * If you now type-write it, COMP-U-write it for a better product.
- > Basic requirements for CCII or 3651/9651:
 - 16K RAM.
 - 117key keyboard.
 - Printer.
 - COMP-U-writer software and instruction manual.
- > For maximum capability:
 - Full 32k RAM.
 - Lower case characters.
- > Talk to other computers: Add a MODEM to your system.

We carry the entire CCII & ISC line of hardware/software, including spares. Send for our 4-page order form for hardware/software. Request separately by item your spare parts needs. Call us for computer servicing and upgrading

Send your order now. We pay the shipping.
Allow 5 weeks for delivery.

We are an authorized ISC dealer

CCII	3650/9650	Description	Quantity	Cost	Amount
010057	010053	Upgrade 72/101 keys	_____	150.00	_____
010058	010054	Upgrade 72/117 keys	_____	250.00	_____
010059	010055	Upgrade 101/117 keys	_____	100.00	_____
010044		24in. RS 232C Cable	_____	45.00	_____
HR1001	HR1002	16k RAM Add-on	_____	185.00	_____
HR1003		Switchable Lower Case	_____	122.50	_____
	0C03LC	32 Lower Case Characters	_____	100.00	_____
990001	990030	5in. Formatted Twin Pack	_____	9.95	_____
	900041	8in. - 10 One Side Format	_____	75.00	_____
	900044	8in. - 10 Two Side Format	_____	100.00	_____
HR0006	HR0006	5in. Exec. COMP-U-writer	_____	299.00	_____
HR0007	HR0007	5in. Mail-Merge C-U-Writer	_____	349.00	_____
991509	991532	FORTRAN	_____	75.00	_____
CAT	Novation MODEM	Transmit/Originate	_____	175.00	_____
			Sub Total	_____	_____
Pa. residents add 6% Pa. Sales Tax			Pa. Tax	_____	_____
			Total	_____	_____
Terms - Cash with order					
Name _____		Telephone #(_____) _____			
Address _____		City _____		St _____ ZIP _____	

HOWARD ROSEN, INC.
PO Box 434
Huntingdon Valley, Pa. 19006
(215)-464-7145

Signature (please sign order) _____

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 V7.89 software; software modifications.

MULTI-ISSUES at \$3.50 each

___ Oct, Nov, Dec 1978 ___ Apr, May/June 1979
___ Jan, Feb, Mar 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

POSTAGE

US and Canada -- First Class postage included.

Europe, S. America -- add \$1.00 per item for air, or
\$.40 per item for surface.

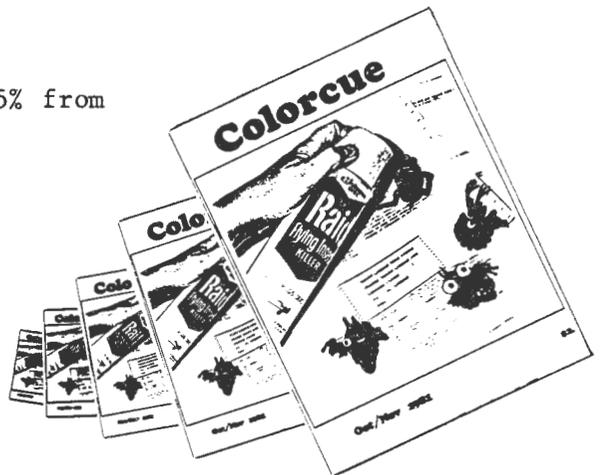
Asia, Africa, Middle East -- add \$1.40 per item for air, or
\$.60 per item for surface.

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

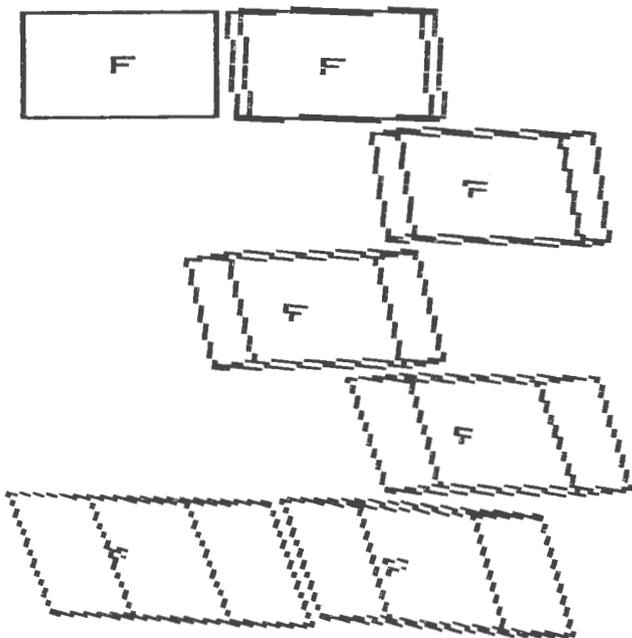
Rochester, N. Y.
Permit No. 415

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

Address Correction Requested

An  Publication

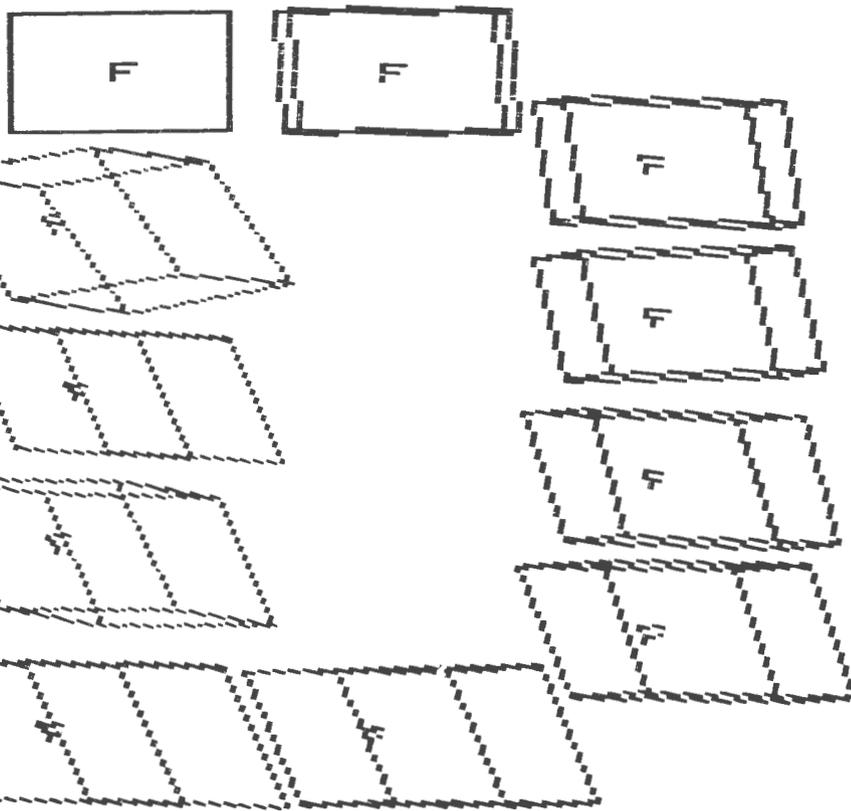
Colorcue



3D Graphics

CRT Mode Plotting

Business Software Reviews



Assembly Language
Programming

More

Colorcue

**A Bi-monthly Publication by and for
Intecolor and Compucolor Users**

Editors:

Ben Barlow
David B. Suits

February/March, 1982
Volume 4, Number 4

- 3 **Editors' Notes**
 - 3 **Compuworld Business Software**
 - 5 **Frepost Computers, Inc. ROM Board**
 - 5 **Cueties**
 - 7 **3-D Graphics, by Doug Van Putte**
Reflection, shear, rotation and scaling
 - 15 **Compuwriter Word Processor, by Howard Rosen**
Software review
 - 16 **Classified Advertising**
 - 17 **CRT Mode Plotting, by Bob V. Smith**
Keyboard tables to make graphics easier
 - 19 **Assembly Language Programming, by David B. Suits**
Part IV: The Status Flags and the Stack
-

Advertisers: You will find our advertising policies attractive. Write for details.

Authors: This is a user-oriented and supported publication. Your articles/tips/hints are required to make it go. Send your articles or write for information.

Colorcue is published bi-monthly by Intelligent Systems Corporation, with editorial offices in Rochester, New York. Editorial and subscription correspondence should be addressed to the Editors, Colorcue, 161 Brookside Dr., Rochester, NY 14618. Product related correspondence should be addressed to ISC, 225 Technology Park, Norcross, GA 30092, ATTN: Susan Sheridan. Opinions expressed in by-line articles are not necessarily those of the editors or of ISC. Hardware/software items are checked to the best of our abilities but are NOT guaranteed.

Editors' Notes

Dial Up Colorcue

Some of our subscribers have accounts with the Source, Compuserve (Micronet), or other system. We'd like to publish a directory of such persons so that CompuColor/Intecolor users can contact each other to exchange ideas, chew the fat, and otherwise pursue the hobby. If you would like your name included in the directory, send us your name, the name of the dial-up facility, and your ID number. Better yet, if you're already dealing with Micronet, leave the message for us there: **Colorcue's** editors (yours truly) are now on Micronet at account number 70045,1062.

By the way, it's easy to get onto Micronet. Drop by your local Radio Shack store and purchase the "Dumb Terminal" information package (Cat. No. 26-2224). This gives you an account number and password for Compuserve's Micronet information facility and one hour of free connect time. There is also an application for permanent status which you must fill out and send in. You will also need a simple program to make your machine into a "dumb terminal". Finally, you will need a modem. If you're already dealing with Micronet, the Source, etc., why not tell us all about it via an article for these pages?

Corrections

It's just not true that there's always an error in **Colorcue**. Who said that?! Well, there's the **occasional** oversight.... Dave's Assembly Language Programming had a small omission last issue which is corrected in Part IV this month. And Ben's article on the serial to parallel interface ought to have indicated, on the schematic on page 17 of last issue, that pins 2 and 6 of the 555 should be connected. Please draw that in so you won't forget.

Doings Down Under

Ralphe Neill of the Victorian (Australia, not B.C.) Users' Group writes a "CompuColor Column" in **Australian Personal Computer** magazine. His column is an unexpected bright spot in these non-CompuColor days. It almost seems as though, now that the CompuColor II is out of production, support is starting to emerge. Witness also Tony Watson and John Newman's book **Programming Colour Graphics for CompuColor/Intecolor Computers**. The book takes you from initial power on through BASIC programming to color (and colour!) graphics, random files and even a driver program for the HIPAD digitizer. Bernie Muldowney, also of the Victorian Users' Group, is selling an assembly language tutorial series on disk. We have seen part of it, and it looks very good. \$50 (Australian) plus postage. Contact Bernie at 5 Dixon Street, Wangaratta, Victoria, 3677, Australia. And there are two active users' groups with growing program libraries. The Victorian Group can be contacted through its Secretary-Treasurer, Keith Ochiltrie, P.O. Box 420, Camberwell, Victoria, 3124. The Western Australian group's librarian is Tony Lee, 52 Cowan Rd., St. Ives, NSW, 2075. **C**

COMPUWORLD BUSINESS SOFTWARE

Compuworld, Inc. has announced a line of business software for the CompuColor II/ISC 3600 series computers. All programs require 32K user memory, one disk drive and a minimal (i.e. regular) keyboard.

ColorCalc

(CCII/ISC 3600 series, \$199.00)

ColorGraph

(ISC 3600 series only, \$150.00)

The so-called spread sheet is a traditional managerial tool which aids in the analysis of financial information. Data is presented in rows and columns and may represent sales informa-

tion, inventory, etc. As a company's budget changes, the data in the rows and columns are changed accordingly, and the entire array can be re-examined. **ColorCalc** is a computerized spread sheet which automatically totals the information you are after. The results can be handed over to another program, **ColorGraph**, which will then provide color graphic displays. Both programs are written in assembly language.

ColorCalc has 12 single-key commands, such as "L" to load data files or "D" for data entry. As many as 51 columns and 52 rows of data may be entered. A formula may be entered on the basis of which the data will then automatically be analyzed. Such a formula may operate on any column or row any number of times. And up to ten formulas may be saved. (24 formulas on 3600 machines.) **ColorCalc** operates with a triple-precision math package which is accurate to the penny up to five billion dollars. Hard copy print out is supported.

ColorGraph takes data from **ColorCalc** and draws graphs of any section of the spread sheet. You can re-scale the graphs, re-color them, re-label them, and print them -- in black and white on an Epson, for example, or in color using the PrintaColor inkjet printer.

Inventory Control

(CCII/ISC 3600 series, \$150.00)

Among the many features of this package is the ability to keep track of distributor's name, quantity on hand, quantity on order, re-order level, last re-order date, late received date, wholesale/retail pricing, and current order price. Reports may be generated by different parameters, including re-ordering, on order, and total retail and wholesale stock value at any time.

Accounts Receivable

(CCII/ISC 3600 series, \$150.00)

Designed with the average operator in mind, this package can keep track of credit ratings and credit limits. Also included are aged account status, his-

tory of total business done with any account, total monies owed at any time by all accounts, statements as often as required, and mailing labels by two parameters. On the 3600 version, this program has linkage to the **General Ledger**.

Accounts Payable

(CCII/ISC 3600 series, \$150.00)

This package supports invoice aging, with account status reporting, total monies owed, and a history of total business done with each vendor. Partial payments are supported, as well as manual and automatic check writing and check registers and mailing labels. On the 3600 series, this program has linkage to the **General Ledger**.

General Ledger

(CCII/ISC 3600 series, \$299.00)

This package is menu driven with 10 levels of account totals, account balances for current month, quarter, previous three quarters, year to date and previous year. It generates trial balance and income statements, balance sheets, special reports and account status reports. A triple precision math package is incorporated and hard copy print out is supported. (This also creates data files for **ColorCalc**. And for 3600 series machines, there is linkage to **Accounts Receivable**, **Accounts Payable**, and, in the near future, **Inventory Control**.)

Mailing List

(CCII/ISC 3600 series, \$100.00)

The Mailing List system stores data for personal and business mailing: names, addresses, phone numbers and comments. The program provides for searching on one or more of eight fields, alphabetical storage of names, support of printing one, two or three labels across a page, sorted in alphabetical or zip code order.

Contact Compuworld, Inc., 125 White Spruce Blvd., Rochester, NY, 14623 (716) 424-6260. ☐

FREPOST COMPUTERS, INC. ROM BOARD

Add-on ROM Board

Most ISC computers have a slot for additional ROM (Read Only Memory). In the CompuColor II, the area in memory from 4000 hex to 5FFF hex is reserved as a user ROM area. Therefore, if you have software you wish to be available at any time, it can go into this area.

Frepost Computers, Inc. will supply a plug in ROM board for your ISC computer. The board is ready for you to plug in EPROMs of your own, or pre-programmed EPROMs available from several sources. Since the CompuColor V6.78 system allows only one user jump vector (ESC ^), your EPROMs can be accessed from a jump instruction poked at 33215. As an added enhancement, Frepost Computers has available a replacement system EPROM to replace one of the original V6.78 system chips. This allows four software or keyboard jumps to the PROM board and adds two more user programmable jumps (like having three ESC ^). And as an added feature, you can customize your system to your requirements. Frepost Computers will even burn your program into EPROM for you.

Bank Selectable EPROM Controller

We also have a device that will plug into the CompuColor CPU board in the Add-on ROM connector just like the small board mentioned above. This device accepts commands from a program or from the keyboard and selects up to seven

more banks of ROM, all occupying the same memory addresses, 4000-5FFF. The EPROM bank selector module adds 56K of additional ROM that is software selectable in 8K byte segments. It uses 2532 type chips which are the same as your system ROMs. If you already have an add on ROM board in your machine, it can be pulled out of its sockets on the logic or CPU board. The EPROM bank select board plugs into these sockets and your present ROM board plugs into the bank selector board. There is a 50 pin bus connector that plugs into the back of the logic board. If you're already using the 50 pin bus connector for something else, we supply you with all instructions on how to solder the appropriate leads to your computer, eliminating the need for our 50 pin bus connector. (If you order it that way, you save \$10.)

The Add-on ROM board is available for \$49.95, assembled and tested, or in kit form for \$39.95, complete with sockets, capacitors and connectors. The EPROM bank selector board is \$249.95 assembled and tested. In kit form, with all sockets, capacitors and chip select parts, it is \$199.95. (Either version with 50 pin bus connector is \$10 more.) The Frepost V6.78 enhanced system PROM is available for \$29.95. Orders less than \$100, please include \$5.00 shipping and insurance. All orders include complete instructions and documentation. Write or call Frepost Computers, Inc., 431 East 20th Street 10D, New York, NY 10010. Phone: (212) 673-6476. The Source: TCI251. Micronet: 70210,374. **C**

Cueties

```
PLOT 12:X=64:Y=X:FOR Z=0 TO 5000:X=X+2-INT(RND(3)*5):Y=Y+2-INT(RND(2)*5):PLOT 2,X,Y,255:NEXT
```

HOWARD ROSEN, Inc.

Authorized ISC dealer



TALKED TO YOUR COMPUTER LATELY?

DID IT TALK BACK?

NOW IT CAN!!!

with the Amazing

TYPE - 'N - TALK

- Type & hear the written word spoken
- Simply connect to the RS232C port

To order send check for \$362.00 with your name, address, city, state, & zip. (Pa. resident please add 6% sales tax). We pay the shipping charges.

- We carry the word processor for the CCII, 3621, 3650. Your computer isn't the same, once you've added the word processor.
..... letters - reports - forms are easily prepared with the aid of the word processor.
- Servicing available for your CCII
- New hardware & software list with itemized description
- Send for new list and order form *
- Never a shipping charge for software & hardware purchases.

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

* Inquire about our free programs

3D Graphics

by
Doug Van Putte
18 Cross Bow Dr.
Rochester, NY 14624

Graphic displays on the computer are enhanced by giving depth and motion to objects. The world of animated graphics using matrix mathematics to present objects with simulated depth and motion will be introduced. I will cover the basic concepts and then describe the elementary motions that can be attained. These motions will be illustrated by a program that can be applied to any object by the reader. Those of you who find the technical treatment too superficial are referred to the references.

First, you will be given a review of the Cartesian coordinate system, followed by the object matrix which is used to define the coordinates of a point on the object. This will be followed by an introduction to the transform matrix which contains the elements of motion that can be applied to the object (matrix). The multiplication of the object matrix and the transformation matrix will then be shown to yield a new, transformed coordinate matrix from which the object can be drawn in its new position. I hope you can visualize now that a previously defined object can be moved through a series of motions by successively applying the operations above to each point on an object, followed by redrawing the object from its "new" points. The types of motion which include scaling, reflection, translation, rotation, and shearing will be shown in simple matrix notation.

CARTESIAN COORDINATE SYSTEM. The Cartesian coordinate system used exclusively here is the PLOT 2 coordinates familiar to ISC computer users. While the origin is usually in the lower left hand corner of the screen, the demo program uses an origin at the center. The X-axis is the horizontal axis, the Y-axis is the vertical axis, and the Z-axis, by convention, is perpendicular to the X and Y-axes and runs toward the viewer and into the depth of the screen. Positive values are toward the right for X, up for Y, and out of the screen for Z.

OBJECT MATRIX [P]. The object in all cases will be a series of coordinate points which are connected by lines (PLOT 242). The individual X,Y,Z coordinates for a point are stored in a column matrix P, which looks like this:

(P is a matrix with one column and three rows)	$\begin{array}{ c} X \\ Y \\ Z \end{array}$	Points are stored in a 2 dimensional array, i.e. X1=P(1,1):Y2=P(2,1):Z1=P(3,1).
--	---	---

If ten points are required to define the object, then ten column matrices, each with one column and three rows, must be created. E.G., P(1,1), P(2,1), P(3,1), ... P(1,10), P(2,10), P(3,10). The object is constructed by connecting the coordinate points P by using PLOT 242 to draw lines between each successive point.

TRANSFORMATION MATRIX [T]. This matrix is sometimes called the operator matrix because each non-zero element is a term which operates on or modifies one of the coordinates, point by point, to transform the object. For now, T will be defined as a 3*3 matrix as follows:

(T is a matrix with three columns and three rows.)	$\begin{array}{ c c c } A & B & C \\ D & E & F \\ G & H & I \end{array}$	Row 1 modifies X. Row 2 modifies Y. Row 3 modifies Z.
--	--	---

The transform matrix T can be multiplied by the object matrix P to yield a transformed object matrix P' as follows:

$$P' = T * P$$

Upon expansion, the new coordinates of P' become:

$$\begin{aligned} X' &= X * A + Y * B + Z * C, \\ Y' &= X * D + Y * E + Z * F, \text{ and} \\ Z' &= X * G + Y * H + Z * I. \end{aligned}$$

Rules of matrix order and shape have to be observed in the multiplication process. (See reference 3.) Using the (3*3) transformation matrix T, operations of scaling, reflection, rotation, and shearing can be used to manipulate the coordinates of a point P. Other operations, called translation and overall scaling, will make use of an expanded (4*4) transformation matrix. All the operations will be illustrated with a simple cube in a BASIC program later on.

SCALING. For pure scaling, T becomes

$$T = \begin{vmatrix} A & 0 & 0 \\ 0 & E & 0 \\ 0 & 0 & I \end{vmatrix}$$

And when T is multiplied by P, then

$$\begin{aligned} X' &= A * X, \\ Y' &= E * Y, \text{ and} \\ Z' &= I * Z. \end{aligned}$$

Independently then X, Y, & Z coordinates can be scaled to produce a reduction or a magnification of an object. The operation of reflection is similar.

REFLECTION. To obtain the reflection of an object in a given direction, all the signs of that coordinate direction must be reversed. For example, to obtain the mirror image of an object through the plane of the screen (X-Y), T becomes:

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{vmatrix}$$

And when T * P then

$$\begin{aligned} X' &= X, \\ Y' &= Y, \text{ and} \\ Z' &= -Z. \end{aligned}$$

SHEAR. An object is sheared by modifying one or more coordinate values of a point in a special manner. A coordinate's value is modified from the original value by an amount computed from the point's position on one or both of the other two coordinate axes. An object can be sheared in its X-direction by:

$$T = \begin{vmatrix} 1 & B & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

And when T * P, then

$$\begin{aligned} X' &= X + B * Y, \\ Y' &= Y, \text{ and} \\ Z' &= Z. \end{aligned}$$

On evaluation, this operation tilts an object by displacing the horizontal (X) values an amount B*Y. Then as a point's vertical (Y) position increases, the displacement of X increases and therefore the redrawn object will be tilted. Note that the X value can be modified by both the Y & Z position of the point at the same time by making the 'C' value non-zero. All three coordinates of a point can be sheared simultaneously by the other coordinates by making B, C, D, F, G, & H non-zero. This operation yields interesting object motions, but the operation of rotation can produce the most life-like movement.

ROTATION. An object can be rotated about any axis to produce a specific orientation in space. The transform examples, however, are restricted to the coordinate major axes. By specifying the rotation angle, an object can be rotated in one of three planes about its origin by using one of the following transformations:

X-Y PLANE (about the Z-axis):

$$T = \begin{vmatrix} \cos(a) & -\sin(a) & 0 \\ \sin(a) & \cos(a) & 0 \\ 0 & 0 & 1 \end{vmatrix}$$

And when $T * P$, then

$$\begin{aligned} X' &= X * \cos(a) + Y * \sin(a), \\ Y' &= X * \sin(a) + Y * \cos(a), \text{ and} \\ Z' &= Z. \end{aligned}$$

X-Z PLANE (about the Y-axis):

$$T = \begin{vmatrix} \cos(a) & 0 & -\sin(a) \\ 0 & 1 & 0 \\ \sin(a) & 0 & \cos(a) \end{vmatrix}$$

And when $T * P$, then

$$\begin{aligned} X' &= X * \cos(a) + Z * -\sin(a), \\ Y' &= Y, \text{ and} \\ Z' &= X * \sin(a) + Z * \cos(a). \end{aligned}$$

Y-Z PLANE (about the X-axis):

$$T = \begin{vmatrix} 1 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) \\ 0 & \sin(a) & \cos(a) \end{vmatrix}$$

And when $T * P$, then

$$\begin{aligned} X' &= X, \\ Y' &= Y * \cos(a) + Z * -\sin(a), \text{ and} \\ Z' &= Y * \sin(a) + Z * \cos(a). \end{aligned}$$

Although not within the scope of this treatment of rotation, it can be stated that:

- 1) The final position of an object rotated sequentially about different axes is a function of the order of applying the transformations.
- 2) An object may be rotated about any arbitrary point by translating that point to the origin prior to rotation and subsequently returning the point to its original position.
- 3) An object can be rotated about any arbitrary axis placed through the object.

Now for the theory and transformations for operations of Translation and Overall Scaling. This treatment will be followed by a demonstration program.

In order to present the last two operations, both the P(Object) matrix and the T(Transform) matrix must be expanded. They become:

$$P(X,Y,Z,1) = \begin{vmatrix} X \\ Y \\ Z \\ 1 \end{vmatrix} \quad T = \begin{vmatrix} A & B & C & U \\ D & E & F & V \\ G & H & I & W \\ 0 & 0 & 0 & S \end{vmatrix}$$

On multiplication, $T*P$ yields $P'(X'',Y'',Z'',S)$. On expansion, the new coordinates become:

$$\begin{aligned} X'' &= X * A + Y * B + Z * C + U, \\ Y'' &= X * D + Y * E + Z * F + V, \text{ and} \\ Z'' &= X * G + Y * H + Z * I + W. \end{aligned}$$

The transformed coordinates are now 'normalized' by dividing them by S to yield P'(X',Y',Z',1) as follows:

$$\begin{aligned} X' &= X''/S, \\ Y' &= Y''/S, \text{ and} \\ Z' &= Z''/S. \end{aligned}$$

In addition to all the operators described earlier, these new forms of the P and T matrices can be used to move and scale an object. For example:

TRANSLATION. For translation only, T becomes

$$T = \begin{vmatrix} 1 & 0 & 0 & U \\ 0 & 1 & 0 & V \\ 0 & 0 & 1 & W \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

And when T*P, then

$$\begin{aligned} X' &= X + U, \\ Y' &= Y + V, \text{ and} \\ Z' &= Z + W. \end{aligned}$$

Thus it is seen that the first three elements (U,V,W) of the fourth column are the elements of translation for each of the three coordinates (X,Y,Z), respectively. This transformation, in conjunction with the others discussed above, provide us with the means of giving realistic movement to an object.

OVERALL SCALING. For Overall Scaling only, T becomes

$$T = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & S \end{vmatrix}$$

And when T*P and the coordinates are normalized by dividing by S, then

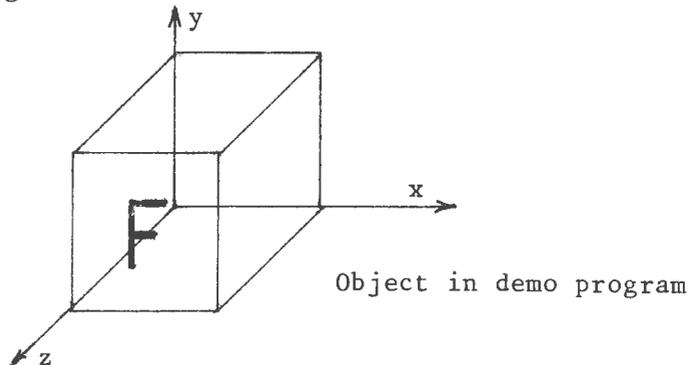
$$\begin{aligned} X' &= X/S, \\ Y' &= Y/S, \text{ and} \\ Z' &= Z/S. \end{aligned}$$

Thus, the fourth element of the last column can be used to scale the three coordinates identically.

DEMONSTRATION PROGRAM.

A few comments about the demo program are in order.

- 1) All prompted inputs are numeric, but the values need to be tailored for the type of operation. Values for each input which cause no operation are given with each prompt. The rotation angles are interpreted as degrees. Start at values close to the no-op values given to begin experimentation.
- 2) There is no internal check for exceeding the screen limits.
- 3) The program produces a composite transform matrix before the object is operated upon. To test the effect of each operation on the object, the user can change the program, or simply use the no-op values for the other operations.
- 4) The user can input his own object. Follow the program comments and don't forget to change the 'NR' value for the number of object points and the matrix dimensions.
- 5) The program is unfortunately slow. What is needed to make the program swifter is an 8080 subprogram which can be called to multiply matrices. Does anyone have one? If not, why don't one of you 8080 Wizards write one and share it.
- 6) All the lines, including the hidden lines, are drawn by the Plot subroutine. Can anyone add a hidden line subroutine?
- 7) Undoubtedly, there are the inevitable bugs. Please pass any you find along to me. **C**



REFERENCES:

1. J. Posdamer, "The Mathematics of Computer Graphics", **BYTE**, Sept., 1978, pp. 22-39.
2. J. Hungerford, "Graphic Manipulations Using Matrices", **BYTE**, Sept., 1978, pp. 156-165.
3. D. Rogers and J. Adams, Mathematical Elements for Computer Graphics (McGraw Hill, 1976).

```

100 REM ***** 3-D GRAPHICS DEMO PROGRAM *****
110 REM ***** BY D. A. VAN PUTTE *****
120 PLOT 12,15: DIM G(4,30), P(4,30), S1(4,4), S2(4,4), T(4,4): NR=24
130 I1(1)=1: I2(1)=2: I1(2)=1: I2(2)=3: I1(3)=2: I2(3)=3: DX=45: DY=45: DZ=45
140 REM * READ OBJECT POINTS FROM ARRAY (LINE 9000)
150 RESTORE : FOR C=1 TO NR: P(4,C)=1: FOR R=1 TO 3: READ P(R,C): NEXT : NEXT

160 GOSUB 530: GOSUB 600: GOSUB 500
170 REM * SCALE OPERATION-REQUIRES INPUT OF 3 NOS.
180 INPUT "SCALE X,Y,Z (1,1,1)? "; A,E,I: S1(1,1)=A: S1(2,2)=E: S1(3,3)=I: IF A=1
    AND E=1 AND I=1 THEN 210
190 GOSUB 450
200 REM * OVERALL SCALE OPERATION-REQUIRES INPUT OF 1 NO.
210 INPUT "OVERALL SCALE FACTOR S (1)? "; S: IF S=1 OR S=0 THEN 240
220 S1(4,4)=S: GOSUB 450
230 REM * SHEAR OPERATION-REQUIRES INPUT OF 6 NOS.
240 INPUT "SHEAR X(B&C), Y(D&F), Z(G&H) (0,0,0,0,0,0)? "; B,C,D,F,G,H: IF B=0 AND
    C=0 AND D=0 AND F=0 AND G=0 AND H=0 THEN 280
250 S1(1,2)=B: S1(1,3)=C: S1(2,1)=D: S1(2,3)=E: S1(3,1)=G: S1(3,2)=H
260 GOSUB 450
270 REM * ROTATION OPERATION-REQUIRES INPUT OF 3 NOS.
280 INPUT "ROTATION ANGLE IN X-Y, X-Z, Y-Z PLANES (0,0,0)? "; A(1), A(2), A(3): IF
    A(1)=0 AND A(2)=0 AND A(3)=0 THEN 310
290 GOSUB 550: GOSUB 450
300 REM * REFLECTION OPERATION-REQUIRES INPUT OF 3 NOS.
310 INPUT "REFLECTION X,Y,Z (1,1,1)? "; A,E,I: IF A=1 AND E=1 AND I=1 THEN
    350
320 S1(1,1)=A: S1(2,2)=E: S1(3,3)=I
330 GOSUB 450
340 REM * TRANSLATION OPERATION-REQUIRES INPUT OF 3 NOS.
350 INPUT "TRANSLATION X,Y,Z (0,0,0)? "; U,V,W: DD=240/(240-W): IF U=0 AND V=0
    AND W=0 THEN 380
360 S1(1,4)=U: S1(2,4)=V: S1(3,4)=W: GOSUB 450
370 FOR C=1 TO NR: P(1,C)=P(1,C)*DD: P(2,C)=P(2,C)*DD: NEXT
380 GOSUB 410: PLOT 12: GOSUB 500
390 INPUT "HIT RETURN TO RESTART? "; ZZ: PLOT 12: GOTO 150
400 REM * SUB TO MULTIPLY T MATRIX * P MATRIX
410 FOR I=1 TO 4: FOR J=1 TO NR: SM=0: FOR K=1 TO 4
420 SM=SM+T(I,K)*P(K,J): NEXT K: G(I,J)=SM: NEXT J: NEXT I
430 FOR R=1 TO 4: FOR C=1 TO NR: P(R,C)=INT(G(R,C)+.5)/S: NEXT : NEXT : RETURN
    N
440 REM * SUB TO CONSTRUCT COMPOSITE TRANSFORM MATRIX (T)
450 FOR I=1 TO 4: FOR J=1 TO 4: SM=0: FOR K=1 TO 4
460 SM=SM+S1(I,K)*T(K,J): NEXT K: IF SM^2<.0001 THEN SM=0
470 S2(I,J)=SM: NEXT J: NEXT I: FOR R=1 TO 4
480 FOR C=1 TO 4: T(R,C)=S2(R,C): NEXT C: NEXT R
490 REM * SUB TO PLOT TRANSFORMED OBJECT (P')
500 PLOT 2,P(1,1)+DX,P(2,1)+DY,242: FOR C=2 TO 18: PLOT P(1,C)+DX,P(2,C)+DY
    : NEXT : PLOT 255
510 PLOT 2,P(1,19)+DX,P(2,19)+DY,242: FOR C=20 TO 24: PLOT P(1,C)+DX,P(2,C)
    +DY: NEXT : PLOT 255: RETURN
520 REM * SUB TO INITIALIZE T MATRIX
530 FOR R=1 TO 4: FOR C=1 TO 4: T(R,C)=0: NEXT C: NEXT R: T(1,1)=1: T(2,2)=1: T
    (3,3)=1: T(4,4)=1: RETURN
540 REM * SUB TO CREATE ROTATION TRANSFORM MATRICES
550 FOR L=1 TO 3: A=A(L)*3.14159/180: IF A(L)=0 THEN 580
560 S1(I1(L), I1(L))=COS(A): S1(I1(L), I2(L))=-SIN(A)
570 S1(I2(L), I1(L))=SIN(A): S1(I2(L), I2(L))=COS(A)
580 NEXT L: RETURN
590 REM * SUB TO INITIALIZE GENERAL TRANSFORM MATRIX (S1)
600 FOR R=1 TO 4: FOR C=1 TO 4: S1(R,C)=0: NEXT C: NEXT R: S1(1,1)=1: S1(2,2)=
    1: S1(3,3)=1: S1(4,4)=1: RETURN
610 REM * ENTER COORDINATES (X,Y,Z) OF EACH DATA POINT IN SEQ.
620 REM * EXAMPLE DATA IS A 3-D CUBE W/ LETTER 'F' ON FRONT FACE
630 DATA 0,0,0,30,0,0,30,30,0,30,30,30,30,0,0,30,0,0,0
640 DATA 0,0,30,0,30,30,30,30,0,30,30,0,30,0,0,0,0,0,0
650 DATA 0,0,30,30,0,30,30,30,0,30,30,0,30,0,0,0,0,0,0
660 DATA 14,13,30,14,17,30,17,17,30,14,17,30,14,15,30,16,15,30

```

Compuwriter Word Processor

by
Howard Rosen
P.O. Box 434
Huntington Valley, PA 19006

Ten ways to use a word processor: write a letter; build a form; write a story; write a report; write a book; write a note to 15 people; write an article; write a research paper; write mailing labels; prepare a sheet of vertical and horizontal lines. The list is not complete, and I'm open to suggestions.

Do any of the above apply to you? Read on and learn the workings of the Comp-u-Writer word processor. How do you use it? Do you have to know how to program? No, you do not do any programming. Let's examine how the word processor is actually used on a day-to-day basis.

While a typewriter can be used to prepare a neat looking document, the word processor does it better and easier. Take, for example, centering the first line of a letter. Just press the key marked **CENTER** and type. The text magically starts at the very center of the line and moves left, then right, while maintaining a perfect center. A typist has to consider page width, line ending, tab setting, and indentation. All of these are child's play to the word processor. Just press the **PRINT** key and a menu with color squares appears. Each color square is associated with a setting: **BLUE** for lines per page; **YELLOW** for characters per line; **RED** for margin settings; and more. After the page is typed in, if you wish to change the number of characters per line, return to the menu and change to a different setting. The text will immediately shift to accommodate your new choice.

Tabs are set by moving a little blue square right or left along a scale at the top of the screen and locking them in place by depressing the **RETURN** key. The **TAB** key is then exactly like your typewriter, except you can change tabs if you like right in the middle of a letter to get a different effect. Suppose you've finished typing and you've found mistakes. Return to the scene of the error, delete the word or character and make your correction. Perhaps you should have made the first paragraph the third one. Use the two keys labeled **MARK BEG** and **MARK END** to place marks on either end of the section in question--watch it turn from green to red to indicate that it's marked--and then move the cursor with the four arrow keys to the exact spot where you want the paragraph to be. Then simply press the **MOVE BLOCK** key and watch how fast all is forgiven. If the marked block is to be repeated, press the **COPY BLOCK** key for each copy desired. The **ERASE BLOCK** key will totally remove the marked block from your text.

To avoid typing long words used repeatedly, you might use x or y or z in substitution, then with the **DEFINE**, **SEARCH**, and **REPLace** keys you can specifically or generally make your replacements. Anything can be underlined or made to be **boldface** by the press of a key. Any character on the keyboard will automatically repeat if held just a little longer.

Now you're ready to print. You can choose single sided, where each page is sequentially printed, or double sided, where the odd pages are printed, then

you reverse the paper for the even pages. If you wish to do two column printing then each consecutive pair of pages becomes the left and the right columns, respectively. This can be done for double sided printing as well, with the word processor keeping an eye on how things are going. If only a part of what was typed is to be printed, then use the marking technique discussed earlier and print only the marked part. Multiple copies can be printed by responding to the question "HOW MANY COPIES?" on the print menu. Documents can be saved on disk for future use. Either the whole document or just the marked portion can be saved.

The most recent version of Comp-u-Writer is version 3.5. There are two types available: the Executive and the

Mail Merge. The Mail Merge, while capable of printing personalized letters, does not support the two sided printing. Mail Merge is useful when a specific letter is needed for a number of customers, each individually addressed.

Comp-u-Writer for the Compucolor II/3600 series computers requires the 117 key keyboard and a minimum of 16K RAM. It is recommended that both 32K RAM (to support long documents) and the lower case letters (to allow normal typing) be available.

As an authorized ISC dealer, we carry the Comp-u-Writer and the entire ISC hardware and software line. We may be reached at HOWARD ROSEN, INC., P.O. Box 434, Huntington Valley, PA 19006 or by phone at (215) 464-7145. ■

CALL FOR ARTICLES

Colorcue gets its material from those who write it. It don't grow on no trees. Nevermind your bad spelling or badly grammar: that's the job of the Editors. You come up with the ideas, splash them onto paper (or disk), and we'll wrestle them into an intelligible form for publication. We can't pay you for your time--you won't become rich. But, since **Colorcue** is read all over the world (well, not **all** over, exactly), then maybe you'll become world famous.... We're looking for ideas, programs, reviews of books and/or software and/or hardware applicable to ISC machines, hardware/software modifications, user group news, and.... Well, you get the idea.

CLASSIFIED ADVERTISING

WANTED: Compucolor II system. Purchase or trade for TRS-80 Model 1. Mike Charlton, (502) 926-3021.

FOR SALE: Compucolor II, V6.78, 24K, non-working. Analog board problems. Complete with manuals, programs, etc. Ben Moser, Rt. 2, Box 550, Stanley, VA, 22851. (703) 778-2861.

FOR SALE: Heathkit microprocessor learning system. M6800 based trainer, with documentation. Excellent package for introduction to hardware. Micheal Ezzo, 692 S. Drake Apt. 0-10, Kalamazoo, MI 49009.

FOR SALE: Compucolor II, extended keyboard, 16K, with diskettes, manual, graphics. Can add modem, sound, and memory. (716) 872-2322 (after 5).

FOR SALE: Intecolor 3651. 32K machine with 5 1/4 inch disk, std keyboard. Manufactured in mid 1981. Asking \$2000. Jim Dantin, (502) 927-6921 x377 (days), or (502) 926-8333 (nights).

FOR SALE: Intecolor 3651. New in original carton. 32K, 5 1/4 inch disk. Asking \$2000. George Wilson, (404) 458-1431 (nights).

CRT Mode Plotting

by
Bob V. Smith
 498 Brown Street
 Napa, CA 94559

Using the deluxe keyboard, graphic displays can be drawn in the CRT mode. Control B enters the plot mode (just like BASIC's PLOT 2.) From then on, the ASCII values of each key press will determine what is plotted. (See the keyboard table on the next page.) The special function keys F0-F15 are used to enter the various plot submodes. In order to simplify incremental plotting, the table below can be of help.

For example, draw a triangle this way: enter the CRT mode, Control B for the plot mode, plot a point with two appropriate key strokes, enter the vector plot submode with the F2 key, plot the other end of the vector with two

more appropriate keys, and finally enter the incremental vector plot submode with the F0 key:

ESC CRT
 Control B
 Control CRT
 Shift ?
 F2
 CRT
 Shift ?
 F0

Now form a triangle by holding the left end of the vector stationary and moving the right end up (North). According to the table, that is the 2 key. **C**

SHF = Shift
 CTL = Control
 COM = Command
 (shift &
 control)

Right
 Side
 (B)

Left side (A)

	N	NE	E	SE	S	SW	W	NW	HOLD
N	SHF 2	COM 2	COM B	COM R	CTL R	R	B	SHF B	2
NE	SHF *	COM *	COM J	COM Z	CTL Z	Z	J	SHF J	:
E	SHF 8	COM 8	COM H	COM X	CTL X	X	H	SHF H	8
SE	SHF 9	COM 9	COM I	COM Y	CTL Y	Y	I	SHF I	9
S	SHF 1	COM 1	COM A	COM Q	CTL Q	Q	A	SHF A	1
SW	SHF 5	COM 5	COM E	COM U	CTL U	U	E	SHF E	5
W	SHF 4	COM 4	COM D	COM T	CTL T	T	D	SHF D	4
NW	SHF 6	COM 6	COM F	COM V	CTL V	V	F	SHF F	6
HOLD	/	CTL /	COM O	COM CRT	CTL CRT	CRT	O	SHF O	

For Y bar graphs,
 A is bottom of line.

Keyboard Table. Each row lists ASCII values produced by the indicated keys (plus Control, Command or Shift, if necessary).

CONTROL	SHIFT	NOTHING	NOTHING	SHIFT	COMMAND	COMMAND	CONTROL	CONTROL	SHIFT	COMMAND	NOTHING					
0	@	32	0	64	@	96	@	128	@	160	0	192	F0	224	F0	
1	A	33	1	65	A	97	A	129	A	161	1	193	F1	225	F1	
2	B	34	2	66	B	98	B	130	B	162	2	194	F2	226	F2	
3	C	35	3	67	C	99	C	131	C	163	3	195	F3	227	F3	
4	D	36	4	68	D	100	D	132	D	164	4	196	F4	228	F4	
5	E	37	5	69	E	101	E	133	E	165	5	197	F5	229	F5	
6	F	38	6	70	F	102	F	134	F	166	6	198	F6	230	F6	
7	G	39	7	71	G	103	G	135	G	167	7	199	F7	231	F7	
8	H	40	8	72	H	104	H	136	H	168	8	200	F8	232	F8	
9	I	41	9	73	I	105	I	137	I	169	9	201	F9	233	F9	
10	J	42	*	74	J	106	J	138	J	170	*	202	F10	234	F10	
11	K	43	+	75	K	107	K	139	K	171	+	203	F11	235	F11	
12	L	44	,	76	L	108	L	140	L	172	,	204	F12	236	F12	
13	M	45	=	77	M	109	M	141	M	173	=	205	F13	237	F13	
14	N	46	.	78	N	110	N	142	N	174	.	206	F14	238	F14	
15	O	47	/	79	O	111	O	143	O	175	/	207	F15	239	F15	
16	P	48	0	80	P	112	P	144	P	176	0	208	F0	240	F0	
17	Q	49	1	81	Q	113	Q	145	Q	177	1	209	F1	241	F1	
18	R	50	2	82	R	114	R	146	R	178	2	210	F2	242	F2	
19	S	51	3	83	S	115	S	147	S	179	3	211	F3	243	F3	
20	T	52	4	84	T	116	T	148	T	180	4	212	F4	244	F4	
21	U	53	5	85	U	117	U	149	U	181	5	213	F5	245	F5	
22	V	54	6	86	V	118	V	150	V	182	6	214	F6	246	F6	
23	W	55	7	87	W	119	W	151	W	183	7	215	F7	247	F7	
24	X	56	8	88	X	120	X	152	X	184	8	216	F8	248	F8	
25	Y	57	9	89	Y	121	Y	153	Y	185	9	217	F9	249	F9	
26	Z	58	:	90	Z	122	Z	154	Z	186	:	218	F10	250	F10	
27	[59	;	91	[123	[155	[187	;	219	F11	251	F11	
28	\	60	<	92	\	124	\	156	\	188	<	220	F12	252	F12	
29]	61	=	93]	125]	157]	189	=	221	F13	253	F13	
30	^	62	>	94	^	126	^	158	^	190	>	222	F14	254	F14	
31	_	63	?	95	_	127	_	159	_	191	?	223	F15	255	F15	

BUSINESS SOFTWARE FOR SALE

- 1982 INCOME TAX PREPARER (for Federal and New York State).
- COMPLETE CASH REGISTER PROGRAM (Includes invoice print, order, inventory count, cash/credit total).
- PAYROLL SYSTEM.
- QSORT PROGRAM
- CCII INTERACTIVE OPERATING SYSTEM

MAU Corporation

5 Eldridge Street, Store North
 New York, New York 10002
 Tel. (212) 431-1277

Assembly Language Programming

by

David B. Suits

PART IV: The Status Flags and the Stack

Well, I goofed last time. Some of you saw the mistake and made the appropriate corrections. The little program I gave you last time to draw a line around the screen will unfortunately only plot four points at the corners of the screen. That's because I left out the incremental plot mode introduction number, 242, which should be inserted just after the 12,2,0,0 in the string of bytes which the program used. In addition, there will now be 14 bytes in the string instead of 13, and so the counter must be initialized to 14 (=0EH) at address 8203H.

The Status Flags

Besides the normal 8080 registers (Accumulator, B, C, etc.), there is an eight bit group of Status Flags, each of which can be set (or 'on', =1) or reset (or 'off', =0), according to the operations of certain instructions. Although the flag register is eight bits wide, there are only five flags; the other three bits are unused. The five flags are the Sign, Zero, Auxiliary Carry, Parity, and Carry Flags. Their

Status Flags:

S	Z	AC		P	C
---	---	----	--	---	---

settings are often used automatically by other instructions. For example, the DCR <reg> instruction which we examined last time will affect all the flags except the Carry flag. Probably the most useful of the flags it affects is the Zero flag, which will be set if the result of DCR <reg> is zero, and reset otherwise. Then a conditional instruction such as JNZ <addr> (Jump if Not

Zero) will automatically test the status of the Zero flag. Thus, the sequence

```
DCR <reg>
JNZ <address>
```

is a very common one in 8080 programming. There are other conditional jump instructions, each of which tests the status of some flag:

Hex	Instruction	Meaning
DA	JC <address>	Jump if carry (carry flag=1)
FA	JM <address>	Jump if minus (sign flag=1)
D2	JNC <address>	Jump if no carry (carry flag=0)
C2	JNZ <address>	Jump if not zero (zero flag=0)
F2	JP <address>	Jump if plus (sign flag=0)
EA	JPE <address>	Jump if parity even (parity flag=1)
E2	JPO <address>	Jump if parity odd (parity flag=0)
CA	JZ <address>	Jump if zero (zero flag=1)

In the previous installment we made use of the CALL instruction. The CALL is really a jump-to-subroutine instruction. You might expect, then, that there would be a set of conditional jump-to-subroutine instructions. And you would be right:

Hex	Instruction	Meaning
DC	CC <address>	Call if carry (carry flag=1)
FC	CM <address>	Call if minus (sign flag=1)
D4	CNC <address>	Call if no carry (carry flag=0)
C4	CNZ <address>	Call if not zero (zero flag=0)
F4	CP <address>	Call if plus (sign flag=0)
EC	CPE <address>	Call if parity even (parity flag=1)
E4	CPO <address>	Call if parity odd (parity flag=0)
CC	CZ <address>	Call if zero (zero flag=1)

Finally, there are corresponding RET (return from subroutine) instructions which are executed only if a flag is of a certain value:

Hex	Instruction	Meaning
D8	RC	Return if carry (carry flag=1)
F8	RM	Return if minus (sign flag=1)
D0	RNC	Return if no carry (carry flag=0)
C0	RNZ	Return if not zero (zero flag=0)
F0	RP	Return if plus (sign flag=0)
E8	RPE	Return if parity even (parity flag=1)
E0	RPO	Return if parity odd (parity flag=0)
C8	RZ	Return if zero (zero flag=1)

Notice that if a flag has been affected (set or reset) by some instruction, it will retain that value until and unless some other instruction is executed which affects it. For example, the INX <reg pair> (increment a register pair) instruction affects none of the flags. So this sequence:

```
DCR C
INX H
INX H
JNZ <address>
```

would have exactly the same effect as

```
INX H
INX H
DCR C
JNZ <address>
```

The Compare Instruction

The contents of the Accumulator may be compared to the contents of another register by means of the **CMP <reg>** instruction. The **CMP <reg>** instruction will not change the contents of either the Accumulator or the other register, but it will affect the statuses of all the flags. However, we will usually be concerned with only two of them: the Zero flag and the Carry flag. The **CMP <reg>** instruction will set the Zero flag only if the two numbers (the contents of the two registers) are equal. Otherwise, the Zero flag will be reset. Thus, a typical loop structure which we used last time,

```
.
.
.
DCR C ;Is C=0?
JNZ <address> ;Not Yet.
. ;Yes.
.
.
```

could alternatively be re-coded in this rather cumbersome form:

```
.
.
.
MVI A,0 ;Accumulator = 0.
DCR C ;Decrement counter.
CMP C ;Is C=Accumulator (=0)?
JNZ <address> ;Not yet.
. ;Yes.
.
.
```

Obviously, such a program, although workable, is not very clever. Here is a clever use of **CMP**:

```
CMP A
```

This instruction compares register A to itself. Thus, the Zero flag is necessarily set, since A is necessarily equal to A. This might be a useful way of setting the Zero flag unconditionally--in case you ever need to do that.

Not only can the contents of a register be compared to A, but a byte of **immediate** data may also be compared to A. Remember that **MOV A,<reg>** will move (i.e., copy) the contents of <reg> into the Accumulator, whereas **MVI A,<num>** will load the Accumulator with the number specified, which is called a byte of immediate data. Similarly, the instruction **CPI <num>** will compare <num> to the contents of the Accumulator. So we could write the loop structure above in still another (but equally cumbersome) way:

```
.
.
.
DCR C ;Decrement counter.
MOV A,C ;Get counter value into A.
CPI 0 ;Is it =0?
JNZ <address> ;Not yet.
. ;Yes.
.
.
```

The **CMP** and **CPI** instructions affect the Zero flag. But they affect another flag as well, the Carry flag. The Carry flag is used for many purposes, and many instructions affect it. Just why and how the Carry flag is affected by the compare instructions is a more involved topic than we have space for here. But we can note one rule of thumb for making use of the result: if the contents of the register (or the byte of immediate data) being compared to the contents of the Accumulator are **greater** than the contents of the Accumulator, then the Carry flag is set (i.e., =1); otherwise the Carry flag is reset (=0). Suppose, for example, that the Accumulator contains 0AH (=10 decimal):

Accum.	Instruction	Carry Fla.
0AH	CPI 9	0 (no carry)
0AH	CPI 0AH	0 (no carry)
0AH	CPI 0BH	1 (carry)

Thus, the instruction CPI <num> (or CMP <reg>) can be interpreted as a test for **greater than**: is <num> greater than the contents of the Accumulator? (Or, is the number in <reg> greater than the contents of the Accumulator?) If so, the Carry flag is set. If not, then there is no carry. There is no instruction which tests for a **less than** relation, although we can do the same thing by a series of conditional jumps. For example:

```

CMP B          ;Compare contents of B
               ; with contents of A.
JC <address>  ;Jump if B>A.
JZ <address>  ;Jump if B=A.
               ;At this point we know that B<A.

```

Another Way to Put Stuff Onto the Screen

In the last installment we developed a handy way of putting a lot of information on the screen. We did this by constructing a loop. Here it is again:

```

MVI C,0DH     ;Put count of bytes into C.
LXI H,9000H   ;HL point to start of bytes.
LOOP: MOV A,M  ;Get a byte.
      CALL 33H ;'Plot' it.
      INX H    ;Point to next byte.
      DCR C    ;Done with all bytes?
      JNZ LOOP ;Not yet.
      .       ;Yes. All Done.
      .
      .

```

Notice that I have taken the liberty of calling the address in the conditional jump "LOOP". If the MVI C,0DH instruction is at 8200H, then JNZ LOOP would translate as JNZ 8205H. But I don't want to have to figure out all those addresses each time I give you an example program, so I'll just **label** a certain spot in the program and then you'll know what I mean by JMP LABEL, or JNC LABEL, and so on. This is the convenience that an assembler allows you. (The MLDP does not.) When you write something like "LOOP:" out to the left (always followed by a colon), the assembler understands that it means whatever address that is. Then, when you write something like JNZ

LOOP, it will translate "LOOP" into whatever address the label "LOOP:" was at. You can have as many labels as you wish, but no two labels may be identical.

Anyway (I got off the track), by using the compare instruction we can now generalize things a bit. Notice that, in order for the program above to operate properly, you first have to load the C register (or whatever register you use for the counter) with the number of bytes in the string. But suppose you don't know (or are too lazy to count) how many bytes there are? In that case, you specify that a certain number will represent the termination byte--it will indicate the end of the string. For example, let's pick the number 239 to represent the end of the string of bytes. Now our program doesn't have to know how many bytes to print--it will just keep CALLing 33H until it finds 239.

Thus:

```

                LXI H,9000H   ;HL point to start of string.
LOOP:          MOV A,M       ;Get a byte.
                CPI 239      ;=239 (terminal byte)?
                JZ EXIT      ;Yes. Jump out of loop.
                CALL 33H     ;No. 'Plot' the byte.
                INX H        ;Point to next byte.
                JMP LOOP     ;Back for more.
EXIT:          .
                .
                .

```

The data at address 9000H will be a series of numbers to be plotted (just as in BASIC's DATA statement), and will end with the number 239. The disadvantage of this scheme, however, is that you cannot PLOT 239. Of course, if you wanted, for some reason, to plot 239, then you could pick some other number to represent the terminal byte. Zero, perhaps:

```

                LXI H,9000H   ;HL point to start of data.
LOOP:          MOV A,M       ;Get a byte.
                CPI 0        ;Is it =0 (terminal byte)?
                JZ EXIT      ;Yes. Jump out of loop.
                CALL 33H     ;No. 'Plot' the byte.
                INX H        ;Point to next byte.
                JMP LOOP     ;Back for more.
EXIT:          .
                .
                .

```

But in this case you couldn't plot a zero.

In BASIC, the PLOT statement handles not only graphics (point plot, bar graphs, etc.), but also letters and digits. You need an ASCII table to tell you what number represents what character. (An ASCII table is provided for you in your Programming Manual. There is also one included with the Machine Language Debug Package Manual.) Thus, PLOT 65 is the same as PRINT "A". And so

```
MVI A,65      ;Put decimal 65 into A.
CALL 33H     ;Print an 'A'.
```

will be the assembly language equivalent. Since the letters, digits, punctuation marks and special characters printed on the screen range from ASCII 32 through 127, a number out of that range, such as 239, can be used as a terminal byte for a string of printable characters. It so happens that there is already such a routine in your computer's ROM, and it expects 239 as the terminal byte (which is why I chose it for my example). Get out your MLDP program and examine the routine which starts at address 33F4H (for V6.78 machines; 182AH for V8.79 and V9.80). Let's name this routine OSTR (for Out STRing). It is easy to use. Just set up HL to point to the first byte of the string of numbers to be plotted and then CALL OSTR. Thus,

```
LXI H,9000H
CALL OSTR
```

If you have stored these bytes at 9000H,

Address	Contents	
	(hex)	(decimal)
9000	49	73
9001	54	84
9002	20	32
9003	57	87
9004	4F	79
9005	52	82
9006	4C	76
9007	53	83
9008	21	33
9009	0C	13
900A	0A	10
900B	EF	239

then the result will be

IT WORKS!

Try the program using the MLDP. (Remember to set a BREAKPOINT after the CALL instruction so that the program will stop and transfer control back to the MLDP.)

CALL and RETURN

We've had enough experience now with the CALL instruction to feel comfortable using it. But what does it do? It does the very same thing as the JMP instruction, but with one clever addition: the address of the next instruction after the CALL will be saved in a reserved area of memory called the **stack**. The subroutine which is CALLED will be able to return to the part of the program which CALLED it by using the RET instruction. That instruction, too, does the same thing as a JMP with a clever addition. Unlike JMP or CALL, RET does not specify an address to return to. How does it know where to go? Simple: it merely jumps to the address contained in the stack area. Let's look at an example.

ADDRESS	CONTENTS	INSTRUCTION
8500	21	LXI H,7000H
8501	00	
8502	90	
8503	0E	MVI C,40H
8504	40	
8505	3E	MVI A,20H
8506	20	
8507	0CD	CALL 8832H
8508	32	
8509	88	
850A	78	MOV A,B
.	.	.
.	.	.
.	.	.
8832	77	MOV M,A
8833	23	INX H
8834	0D	DCR C
8835	0C2	JNZ 8832H
8836	32	
8837	88	
8838	0C9	RET

This little program has the effect of storing 20H (=32 decimal) at 40H (=64 decimal) memory locations beginning at address 7000H. When the CALL 8832H instruction is encountered at address 8507H, the address of the instruction following the CALL 8832H is stored "on the stack" (as they say). That address is 850AH. Then a jump is made to address

8832H where the program continues. It loops around until the contents of the C register become zero, and then it RETURNS; that is, a jump is made to the address contained on the stack, namely, 850AH. Pretty neat, eh?

The Stack and its Pointer

The stack itself is merely a designated section of memory. The programmer may designate which section of memory is to be used as the stack area. The 8080 has a number of instructions which, either explicitly or implicitly, store or retrieve data in that area. The CALL and RET instructions, for examples (plus all their conditional variations, such as CNZ, CC, RZ, etc.) make implicit use of the stack. On the other hand, some instructions, such as **PUSH <reg pair>** and **POP <reg pair>**, which I'll discuss in a moment, make explicit use of the stack.

The stack may hold lots of data at once: it is not limited to storing only one address (or, more generally, two bytes). The stack is sometimes more descriptively called a "push down stack", or sometimes a "first-in, last-out" stack. The common analogy is a stack of plates in a cafeteria: the cafeteria staff adds a clean plate to the stack, which sinks down a bit on a spring loaded holder. A customer comes along and takes a plate (the **top** plate, please; don't be difficult), and the whole stack rises up a bit. Just how many plates you can put on the stack depends on the size of the holder. And just how many plates can be taken off the stack depends on how many plates are on it. The nature of the physical universe determines which plate is the top one.

The analogy is not bad, but don't get carried away by it. The computer's stack doesn't move at all. Rather, there is a **pointer** to the most recent addition to it. And instead of rising up, like the stack of plates, the computer's stack grows downwards (where "down" means toward lower addresses; so you can't make the stack grow upwards by

turning your machine upside down). More accurately, the stack pointer is adjusted downward when a new addition to the stack is made, and it is adjusted upward when something is retrieved from the stack.

Now I'm going to confuse you. I have been writing programs (such as the example above) downwards toward the bottom of the page; but the addresses increase--they go upwards. That's quite ordinary: you do the same when writing a BASIC program:

```
10 REM
20 A=A+1
30 .
.
```

But now I want to illustrate a downward growing stack, and I am tempted to make the lower addresses actually be lower on the printed page, like this:

```
9327
9326
9325
.
```

But I won't do that. So I'm going to illustrate a downward growing stack by a list of upward growing addresses which go downward on the printed page. Got that? Nevermind. You would probably have been better off had I not mentioned this at all. Just remember that up is down and down is up, and that there is a gross equivocation going on. [Ed note: At this point the author succumbed to a fit of conversion hysteria and had to be locked up for two weeks inside a mass storage drum filled with octal notation. Fortunately, he recovered just before this month's deadline and was able to finish this article.]

Back to the stack pointer. Only two bytes (which could represent data or an address)--no more, no less--are put onto or taken off the stack during a stack operation. That is, although the stack area might be as big as you please, the operation of adding something to or retrieving something from the stack always involves exactly two bytes. One of the 8080's registers is called the **Stack Pointer (SP)**. It is a two byte register which always contains the ad-

dress of the most recent two byte addition to the stack. Suppose, for example, that the stack begins at 9327H and grows down from there:

Stack Pointer	Address	Contents
	9324	x
9327	9325	x
	9326	x
	9327	x

If a CALL instruction is encountered in a program, the address of the next instruction is PUSHed onto the stack, and the stack pointer is adjusted down by two bytes. In the example program given earlier, the CALL 8832H would have this effect: decrement the SP; store the high byte of the address of the next instruction after the CALL in the memory location addressed by the SP; decrement the SP again; store the low byte of that next instruction in the memory location pointed to by the SP; then jump to the address given in the CALL instruction.

Stack Pointer	Address	Contents
	9324	x
9325	9325	0A
	9326	85
	9327	x

The RET instruction retrieves (POPs) an address from the stack: get the low byte from the memory location addressed by SP; increment the SP; get the high byte; increment the SP; jump to the address so retrieved.

Stack Pointer	Address	Contents
	9324	x
9327	9325	0A
	9326	85
	9327	x

Notice that the contents of the stack are not changed by the RET instruction. Rather, the SP is merely repositioned, i.e., points to a new address.

PUSH and POP

Explicit (as opposed to implicit) manipulation of the stack pointer and the stack's contents is possible with

some of the 8080's instructions. The two most common are **PUSH <reg pair>** and **POP <reg pair>**. During a PUSH instruction the stack pointer is adjusted down two bytes and the two byte register pair is copied onto the stack. The contents of the register pair are not changed. During a POP operation, the two bytes on the stack are copied into the specified register pair and the SP is adjusted up two bytes. Thus,

```
LXI H,2395H
PUSH H
```

has the effect of storing 2395H on the stack and adjusting the SP downward two bytes. And

```
POP H
```

loads the register pair HL with the two bytes on the stack and adjusts the stack pointer back up two bytes. Notice that the PUSH and POP instructions don't care what is in a register pair or on the stack. Moreover, there is no record kept about where the information on the stack came from. Thus,

```
LXI H,2395H
PUSH H
POP B
```

will (1) load HL with 2395H, (2) copy the contents of HL onto the stack and adjust SP down, and then (3) copy the contents of the stack into BC and adjust SP back up. The final result is that SP will be where it was to start with and both HL and BC will contain 2395H. If you want to get two bytes off the stack and still keep them on the stack, then

```
POP <reg. pair>
PUSH <reg. pair>
```

can be used.

One of the main functions of PUSH and POP is the temporary saving of register contents. If you want to make use of DE, for example, but you also do not want to lose what's presently there, then here's how to do it:

```
PUSH D ;Save DE on stack.
blah
blah
blah
POP D ;Retrieve DE.
```

Quite often a subroutine will make extensive use of the registers, and in order to avoid destroying important data being held in those registers, the subroutine might save some or all the registers on the stack and retrieve them just before RETURNing:

```
SBR0UT: PUSH PSW      ;Save all registers.
        PUSH B
        PUSH D
        PUSH H
        blah          ;blah
        blah
        blah
        POP H         ;Restore all registers.
        POP D
        POP B
        POP PSW
        RET
```

(The PSW, remember, is the one byte Flags register along with the Accumulator.) Notice that the registers must be restored in the opposite order they were saved, otherwise the contents of some of the registers would be interchanged. (Sometimes that is done on purpose.) Notice also that the number of POPs must equal the number of PUSHes. Why? Because

when the subroutine is CALLED, a return address is automatically PUSHed onto the stack. Then the subroutine PUSHes some registers onto the stack. In order to get that return address back again, all those two byte additions to the stack must be cleared away first. (That's why we call it a first-in last-out stack.) If you POP too few, or POP too many, what you get as the return address with the RET instruction might be very surprising.

POP QUIZ: What will this series of instructions do?

```
LXI H,8A94H
PUSH H
RET
```

Answer: It will have the same effect as JMP 8A94H.

NEXT TIME: Manipulating the stack pointer; more ways of getting data to and from memory; addition and subtraction; getting data from the keyboard. **C**

8 K R A M B O A R D

(For V6.78 Compucolors only. V8.79 available soon.)

This 8K of additional RAM is addressed at 4000H-5FFFH, the space unused by your CompuColor and available for PROM. Now you can add additional RAM instead, allowing you to increase the maximum RAM of your machine to 40K. Installation is easy, requiring a small modification to the logic board. It is compatible with the Frepost Computers, Inc. Bank Select ROM Board (see notice in this issue).

US\$65.00 plus \$2.50 postage and handling

Tom Devlin
3809 Airport Road
Waterford, MI 48095

For more information or a copy of the installation instructions, send a SASE. Also see the article in the next **Colorcue**.

Add 16K RAM

TO YOUR 16K COMPUCOLOR 11 (V6.78, V8.79 & 3621)

for only \$138 (u.s.)

- * Completely assembled and tested.
- * No soldering required. Just plug in.
- * Full installation instructions included.
- * All RAM chips are in sockets (8).
- * Spare RAM CHIP included.
- * 90 Day warranty.
- * Price includes air mail costs. (Aust.=\$120, Canada =\$158)

PPI PROGRAM PACKAGE INSTALLERS,
8 Hillcrest Drive,
DARLINGTON,
WESTERN AUSTRALIA 6070

* Add lower case *

TO YOUR COMPUCOLOR 11 (V6.78, V8.79 & 3621)

for only \$40 (u.s.)

- * Completely assembled and tested.
- * No soldering required. Just plug in.
- * Full installation instructions included.
- * Both 2708 EPROMS are in sockets.
- * Switchable between Lower case and graphics. (switch incl.)
- * 90 Day warranty.
- * Price includes air mail costs. (Aust.=\$36, Canada=\$47)

PPI PROGRAM PACKAGE INSTALLERS,
8 Hillcrest Drive,
DARLINGTON,
WESTERN AUSTRALIA 6070

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 V7.89 software; software modifications.

MULTI-ISSUES at \$3.50 each

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

POSTAGE

US and Canada -- First Class postage included.

Europe, S. America -- add \$1.00 per item for air, or
\$.40 per item for surface.

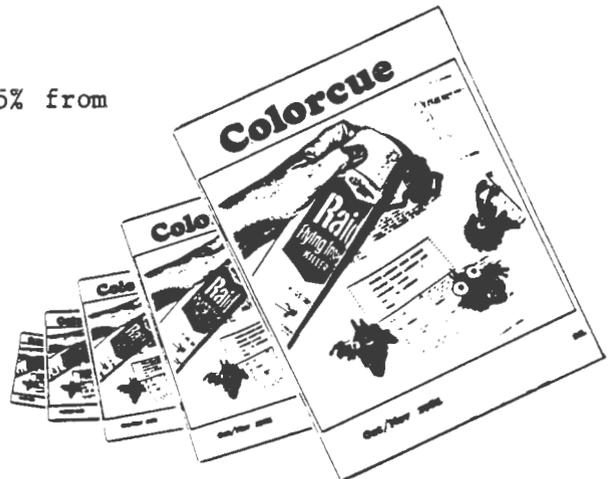
Asia, Africa, Middle East -- add \$1.40 per item for air, or
\$.60 per item for surface.

DISCOUNT

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

ORDER FROM:

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618



BULK RATE
U.S. POSTAGE
PAID

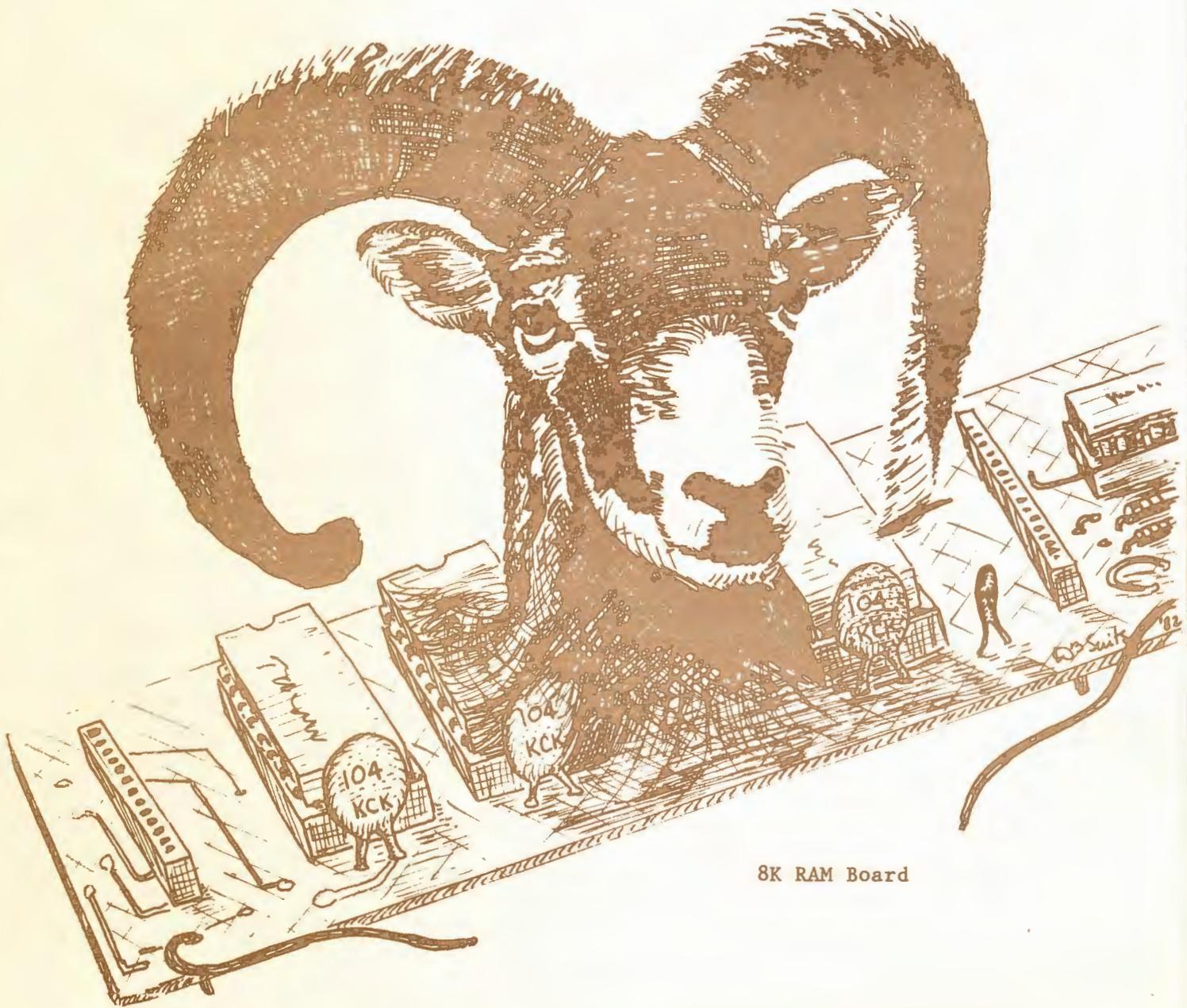
Rochester, N. Y.
Permit No. 415

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

Address Correction Requested

An  **Publication**

Colorcue



8K RAM Board

April/May 1982

\$2.

Colorcue

A Bi-monthly Publication by and for
Intecolor and CompuColor Users

Editors:

Ben Barlow
David B. Suits

CompuServe: 70045,1062

April/May, 1982
Volume 4, Number 5

- 3 **Editors' Notes**
 - 5 **8K RAM Board for the CompuColor II, by Tom Devlin**
Expand your machine to 40K
 - 12 **Classified Ads**
 - 13 **In and Out of the CompuColor II, by Jane Devlin**
How to take the back off--and put it on again
 - 17 **Combine Record Documentation with Record Access,**
by Alan D. Matzger
Improve your random file routines
 - 18 **About Your Subscription**
Keep COLORCUE alive!
 - 19 **Assembly Language Programming, by David B. Suits**
Part V: Creating an INPUT routine
-

Advertisers: You will find our advertising policies attractive. Write for details.

Authors: This is a user-oriented and supported publication. Your articles/tips/hints are required to make it go. Send your articles or write for information.

Colorcue is published bi-monthly by Intelligent Systems Corporation, with editorial offices in Rochester, New York. Editorial and subscription correspondence should be addressed to the Editors, Colorcue, 161 Brookside Dr., Rochester, NY 14618. Product related correspondence should be addressed to ISC, 225 Technology Park, Norcross, GA 30092, ATTN: Susan Sheridan. Opinions expressed in by-line articles are not necessarily those of the editors or of ISC. Hardware/software items are checked to the best of our abilities but are NOT guaranteed.

Editors' Notes

Aboard a Proud Ship

Some CompuColor/Intecolor owners, for various reasons, have abandoned their first machine and have struck up friendships with the Atari 800 or Apple II microcomputers. (I have yet to hear of an ISC computer owner switching to TRS-80, or even to the Color Computer.) One may wonder about the wisdom of such a switch: what those other machines offer that the CompuColor/Intecolor does not is, to my mind, so miniscule as to not make up for their serious deficiencies, not the least of which is their little toy keyboards. And I suspect that the greater resolution available for graphics on the Apple II is hardly a sufficient motive for discarding the otherwise superior display which the ISC machines have. Nor is the 6502 microprocessor, which the Atari and Apple II computers use, a step up from the old klunker 8080.

No, it is not, I suspect, the hardware which entices an ISC computer owner to abandon ship; it is, rather, the wealth of independent software and hardware vendors who flood the market with unimaginably diverse products for use with the Atari and Apple II computers. And if you're considering joining the ranks of the producers-sellers, the market for Atari and Apple II software is still very strong. How can a budding author hope to make **that** kind of money in the restricted CompuColor/Intecolor market?

ISC fans have been tempted away even more by the recent introduction of the IBM Personal Computer. I admit that the PC appears to be a very fine machine. But does it really represent very much of a step up from the CompuColor/Intecolor? Sure, there's an 8088, but it grinds along at a blindingly slow pace. It is instructive to read Vol I, No. 1 of **PC Magazine**, a new publication dedicated to the IBM PC. Its editor, Jim Edlin,

is pictured with two computers: the new IBM PC is in the background, and in the foreground is a ... a CompuColor II! "I have", Jim says, "what I believe was the best personal computer of the pre-IBM era, though you've probably never heard of it. It's called a CompuColor II...." After expressing some doubts about the PC in its present form, Edlin says, "I'm not quite ready to put my CompuColor II away. But I can see it won't be long." He anticipates an evolution of the IBM PC, by independent vendors if not by IBM itself.

Still and all, I wonder if it isn't mere hype to talk about "the pre-IBM era", as though the IBM PC is really a significant advance. For my own part, I am not at all interested in the new machines which--good as they surely are--offer not very much more than popularity over the CompuColor II. I will not be ready to put away my CompuColor II until I finish the design and construction of my own "step up". Don't hold your breath; it will be a 68000-based high resolution color graphics system. The sheer size of the thing boggles my mind: the display refresh memory alone will occupy 153K bytes.

No, it will be quite some time before I find a better machine to move up to. In the meantime, my CompuColor II, for all its foibles and follies, does me right proud every time.

--DBS

New Publication

The U.K. CompuColor Users' Group has launched publication of their newsletter, **CompUKolour**. April was their first issue. It was 40 pages, including: Bill Donkin's lengthy discussion of how the directory and BASIC programs (or "programmes", if you prefer) are stored on disk; notes on programs in the group's library; notes on disk drive speed and alignment adjustments; an index to **Colorcue** from Volume 1 through Volume 3; and Dale Dewey's article on the TMS 5501. The Secretary/Librarian is Bill Donkin. Contact the group at 19 Harwood Avenue, Bromley, Kent, BR1 3DX, England.



RENAISSANCE MARKETING ANNOUNCES
 LOW COST BUSINESS PROGRAMS
 FOR YOUR COMPUCOLOR MODEL 4, MODEL 5

* GENERAL LEDGER (16K,32K) \$ 59.95

-
- (1) - ACCOUNT PROGRAM : DISPLAY ACCOUNT DATA , LIST ALL ACCOUNTS , ADD ACCOUNT , DELETE ACCOUNT , CHANGE ACCOUNT DATA.
 - (2) - JOURNAL PROGRAM : ENTER JOURNAL DATA , DISPLAY JOURNAL DATA BY ENTRY # , CHANGE ENTRY DATA.
 - (3) - PROOF PROGRAM : DISPLAY OR PRINT JOURNAL ENTRY PROOF SHEETS WITH TITLES.
 - (4) - POST PROGRAM : APPLIES JOURNAL ENTRIES TO ACCOUNTS.
 - (5) - REPORT PROGRAM : PRINT BALANCE SHEET , PRINT INCOME STATEMENT WITH TITLES.

INCLUDES : EASY TO UNDERSTAND INSTRUCTIONS, SAMPLE ACCOUNTS, SAMPLE ENTRY FORMS, SAMPLE PRINTOUTS.

* INVENTORY CONTROL (16K,32K) \$ 34.95

GIVES THE FOLLOWING REPORTS.

- (1) - DISPLAYS ITEM DATA BY QUANTITY
- (2) - PRINT OR DISPLAY ALL ITEMS ON FILE.
- (3) - DISPLAY ITEM DATA BY CLASS CODE.
- (4) - PRINT OR DISPLAY ITEM DATA BY VENDOR.
- (5) - UPDATING SECTION
 - (A) - ADD NEW ITEM
 - (B) - UPDATE ITEM QUANTITY
 - (C) - CHANGE ITEM DATA
 - (D) - DELETE ITEM
- (6) - DISPLAY ITEM DATA BY ITEM #.

PROVIDES QUANTITIES AND DATA FOR 750 ITEMS OR MODELS.

* INVOICE CONTROL (16K, 32K) \$ 49.95

INVOICE CONTROL PROGRAM PROVIDES DATA FOR 768 INVOICES PER DISK PER SIDE. GIVES ON A DAILY BASIS MONTH-TO-DATE, YEAR-TO-DATE INVOICE INFORMATION ON TOTAL SALES, ACCOUNTS RECEIVABLE, AMOUNT PAID, TAX DUE, FREIGHT CHARGES, INVOICE COST AND PROFIT.

DISPLAYS = LIST ALL INVOICES, SEARCH BY CUSTOMER, INVOICE DATA, SEARCH BY P.O. #, LIST OF TOTALS (TOTAL SALES, SALES TAX, FREIGHT AND COST).
 PRINTS = ALL INVOICE DATA (MONTHLY, YEARLY), A/R DATA (BY CUSTOMER, OR ALL INVOICES MONTH-TO-DATE, YEAR-TO-DATE), INVOICE TOTALS (M-T-D, Y-T-D AMOUNTS PER INVOICE AND TOTALS FOR SALES, TAX, FREIGHT, COST AND PROFIT).

CALL OR WRITE FOR ADDITIONAL INFO.

OPTIONAL INVOICE PRINTING USES NEBS TRACTOR INVOICE FORMS..... \$ 19.95
 OPTIONAL INVENTORY CONTROL INTERACTIVE WITH INVOICE PRINTING.....\$ 29.95

MAXELL MINI DISK FOR COMPUCOLOR. BOX OF 10\$ 34.95
 30% OFF ALL COMPUCOLOR CORP. SOFTWARE IN STOCK. CALL

RENAISSANCE MKT. IN NJ. ADD 5% SALES TAX
 7 SO. PIERSON RD.
 MAPLEWOOD, NJ. 07040
 201-762-0585

SHIP TO _____

[] INVOICE CONTROL \$ 49.95
 [] GENERAL LEDGER \$ 59.95
 [] INVENTORY CONTROL \$ 34.95
 [] MAXELL DISK BOX QTY. _____

TERMS: PAYMENT WITH ORDER FREIGHT: PREPAID IN USA

* AVAIL. FOR INTERCOLOR MAY, JUNE 1982 TAKING ORDERS NOW.

8K RAM Board for the CompuColor II

by Tom Devlin
3809 Airport Road
Waterford, MI 48095

For some time I had been casting a wistful eye at the 4000-5FFF "Future Space" section of the CompuColor II memory map. ISC left this area open to give us the option of installing 8K of EPROM for special programs. They mounted connectors J9 on the logic board to mate with an EPROM board and programed PROM UB3 to supply chip select signals to this board during memory read cycles. Unfortunately very little use has been made of this feature.

My first thought upon discovering this unused area was that it would be a great spot for some RAM. This would be far more flexible than EPROM in that with RAM the program could be changed at will or it could be used for temporary storage if the application required it. One of the nicest features of placing RAM in this area would be that BASIC wouldn't know it existed so you could (ESC)-W without worrying about your utility program.

My search for suitable RAM chips turned up the TOSHIBA 2016, a 2K x 8, low power, static device. Four of these provided the full 8K needed and occupied the same physical space as the 2716 EPROMS used on the CompuColor add-on board.

The only problem was the address decoding. ISC had intended this area for EPROM and had made no provision for accessing the chips during a write cycle. Rather than replace the UF3 system decoder PROM and the UB3 chip select PROM, I decided to disconnect the appropriate outputs of the UB3

PROM and run the A12-A15 address lines through the same J9 connector pins. This was easy since A12-A14 were already present on pins 11-13 of UB3 and A15 was available on pin 36 of the 8080 CPU just a short distance away. These four lines and the A0-A11 lines already present on J9 allowed access anywhere in the 64K memory range. While I was mainly interested in the 4000-5FFF area I left the door open for future expansion into the screen memory area via pads 0 thru V on the board. If all goes well this will be a part of another **Colorcue** article.

At this point I had almost everything needed. The only thing left was to pick up the active low memory read (MRD) and write (MWR) strobes. Two wires from convenient locations on the logic board to the new RAM board completed the modifications.

The schematic shows the final circuit. I decided to make the board accept single 5 volt supply (Intel type) EPROMS as well as the 2016 RAM chips. Each board will accept 8K of 2016 RAM or 2716 EPROM or 16K of 2732 EPROM in two switch selectable 8K banks. The schematic shows the jumper and switch settings for each type of memory chip. I also made provision for stacking these cards and selecting between them with a switch. Jumpers A-B and C-D allow for easier stacking by re-routing the MRD and MWR signals from the lowest (bottom-most) board to the upper one through connector pins 13 and 14. If you use this

option, you **MUST** cut off pins 13 and 14 on the bottom of the lower board or serious damage to the computer may result because the logic board uses these pins for +12 and -5 volt power.

Address decoding and chip selection are handled by IC-1 (74LS138), a three line binary to 1 of 8 decoder. Assuming that IC-1 is enabled by a logic zero (ground) through one of the two diodes, one of its outputs will be low through a 2K or 4K block within the 4000-7FFF range. The size of the block has to match the size of the memory chip installed and is set by jumpering M-L or M-N. (If IC-1 is not enabled, or we are outside the 4000-7FFF range, all eight outputs will remain high.)

Pin 18 of any of the three types of memory chips is the active low chip enable (CE). If this pin is pulled low, the chip "wakes up" and awaits further orders. If pin 20, the active low output enable (OE), is then pulled low by MRD, the memory chip will output a byte onto the data bus. In the case of the 2016 RAM chip if pin 21, the active low write input (W), is pulled low by the MWR strobe, the enabled chip will take the byte currently on the bus and store it internally. If CE goes low, but neither the MRD or MWR follow (as might happen during an I/O cycle) the enabled chip will do nothing.

Pin 21 has other functions on the EPROM memories. The 2716 uses it for the programming voltage (VPP). For normal operation this pin must be held at +5 volts. The 2732 uses pin 21 for the All address line to double the storage capacity to 4k bytes.

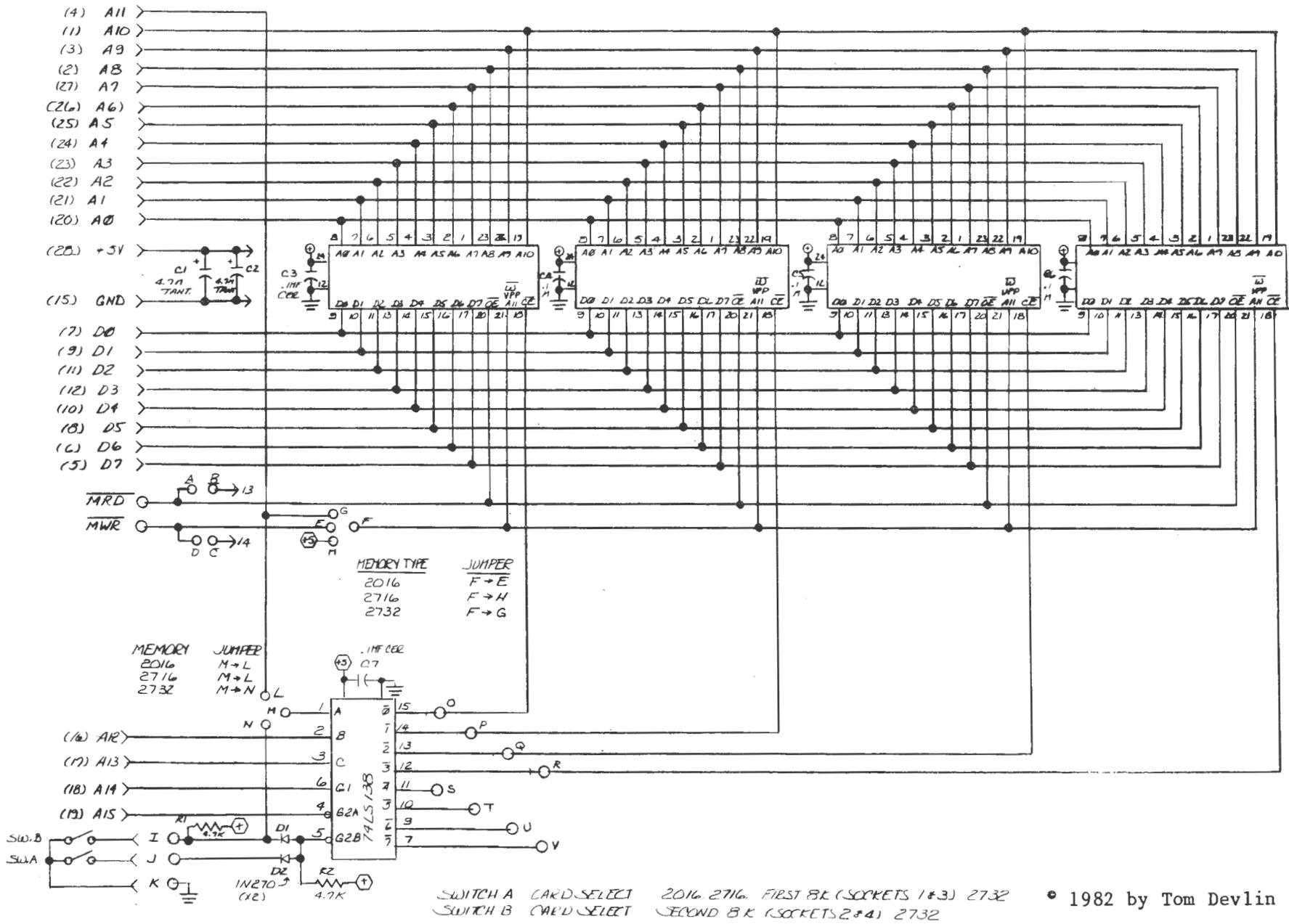
Resistors R1 and R2 simply pull the IC-1 inputs to a logic one (+5 volts) when the switches are open. The 1N270 diodes specified are germanium devices chosen for their lower forward voltage drop compared to the more common 1N914 silicon types. 1N914s should work fine with the switches (in fact D2 is not needed if you use switches for control) but the lower drop of the 1N270s will give some extra margin for possible future card selection under software control. The capacitors are for power supply by-passing.

Installation on the V6.78 units is straight-forward. Just follow the REV.3 instructions. The situation is a little more complex with the V8.79 types: the REV.3 logic board was changed to a newer REV.4 board part way through production. Also, the initial run of V8.79 UA5 FCS ROMs must have had a bug because CompuColor patched around one section of this ROM to an EPROM located on the add-on EPROM board (the only time this board was ever used) and changed the UB3 PROM to reflect the change. If you have one of these units with the add-on board installed you will have to update to the later UA5 ROM (PN 100695) and replace the UB3 PROM so you can remove and discard this EPROM board and use the J9 connectors for the RAM card. I can supply a preprogrammed 2532 EPROM to replace the UA5 ROM and the proper UB3 PROM if needed.

The REV.4 logic board used in the final units is mostly the same as the REV.3 board (a few changes to the RS232 circuitry and the handshake mod added), but for some reason ISC switched from the 16 pin UB3 PROM used on the older boards to a 20 pin part. I have been unable to find a REV.4 board to make installation drawings from. If anyone can supply a good photograph or drawing of this board I will try to get the information into a future **Colorcue**.

If you elect to wire wrap a RAM card, the PC layouts and assembly drawing should, with the schematic, give you all the information needed. The 14 pin header assemblies are a little tough to find, but you can make your own by cutting a 28 pin wire wrap socket in half lengthwise. Just be sure that the socket you pick has large enough holes to allow you to stack another board on top. Also, on some sockets the pins are offset slightly from the holes. If yours are like this, make sure you keep the hole center-to-center spacing at 3.9" to make stacking easier. (The CompuColor J9 spacing varies a little from unit to unit. The .65" pins on the headers we use are long enough to flex slightly to make up for this. I would recommend using this length.)

If you do decide to roll your own,



would you please drop me a line and tell me how you made out?

Installing the RAM Card on Rev. 3 Logic Boards

A word of warning. STATIC CHARGES KILL MOS ICs. Just walking across the floor can produce enough of an electrical buildup to totally destroy MOS parts. The RAM chips are just as sensitive as other parts of your machine, so remember this warning whenever you work on your computer. To avoid this potential problem touch the metal frame of the disk drive to neutralize any charge you may have accumulated. Do this as soon as the back is off.

(1) Disconnect the A.C. power cord and all connectors from the rear of the computer.

(2) Remove the back (3 screws top, 1 bottom). Be careful not to hit the neck of the CRT during this step.

(3) Being careful to mark socket placement, remove the three cable connectors and the sixteen pin flat cable for the disk drive from the left side of the logic board (bottom circuit board). Remove the logic board.

(4) Locate and remove PROM UB3 (82S123). (Figure 1A.) Bend pins 1,2,3, and 4 out from the body of the chip. Make sure that the pins don't touch each other or anything else when the chip is replaced in the socket. If UB3 is not socketed you can cut these pins off close to the board with a sharp pair of wire cutters.

(5) Turn the logic board upside down and locate the socket pins for UB3. Using four lengths of insulated jumper wire, make the following connections (see Figure 2):

- () pin 2 to pin 13
- () pin 3 to pin 12
- () pin 4 to pin 11
- () pin 1 to UA2 socket pin 36 (8080, A15 output).

(6) Turn the logic board rightside up. Plug the RAM board into connectors J9. The end with the small parts goes to the rear (edge card connector end) of the logic board. Support the logic board from the bottom to keep it from flexing during this step.

(7) Connect a wire from the MWR pad on the RAM board to pin 8 of J8 at the front end of the logic board. You can connect directly to pin 8 of the 16K add-on RAM card if you have it, otherwise use a wire wrap pin. See Figure 3.

(8) Locate the feed-thru pad shown in Figure 1. Connect a wire from the MRD pad on the RAM board to this point.

(9) Double check your work. Pay close attention to the UB3 and UA2 chips to be sure that you have not shorted two adjacent pins together when you soldered the wires to them.

(10) Re-assemble the computer. Remember to reconnect all of the cables removed during the disassembly. The logic board is supported at the rear by two tabs that fit into slots in the rear cover. Be sure to get these in place.

With the power switch off reconnect the keyboard and power cords. Take a deep breath and turn the power on. If the computer does not act normally, turn the power off immediately. Otherwise relax and wait for the screen to light up.

Problems can be localized by removing the RAM card from the computer. (You do not have to "unmodify" the logic board.) If everything then works normally, you either have a bad RAM card or else you have connected the MRD or MWR wires to the wrong places.

If the display "jitters" or the colors don't quite line up don't panic; you've probably just moved the horizontal centering control. (The small trim pot at the left rear edge of the logic board.) Re-adjust it with a small screwdriver. (The display will move in steps as the control is turned. Set the control for the best picture midway between two steps.)

If all is well so far, put a disk into the drive, enter FCS, and

```
FCS>SAVE RTEST 0-1FFF
```

to put 8K of fairly random data onto the disk. Now try to load this data into the RAM card with

```
FCS>LOAD RTEST.PRG;01 4000
```

If you don't get any EMEM error messages, you are home free!

Software

Some software is already available for the 4000H-5FFFH area. Quality Software Associates (21 Dersingham Cr., Thornhill, Ontario, Canada, L3T 4P5) has a greatly expanded version of 'THE' BASIC EDITOR for use with the RAM card. It uses the full 8K and is a joy to work with. If you have already purchased 'THE' editor, send them your disk and \$6.00 and they will run you a copy. This program will be included in all future disks sold. You can also relocate the Compucolor MONITOR or MLDP programs using the MENUS supplied with them. (Line 710 of the MENU program prevents ORGing below 8200H, so just delete it.) I have included a program to reORG version 3.20 of FREDI to 4000H.

The RAM card is compatible with Frepost Computers' bank switching E-PROM system. Frepost has instructions available. If, however, you install the RAM card and later decide to install the Frepost system, you will have to return the logic board to its original condition before proceeding.

The RAM card may also be purchased assembled and tested for \$65.00 plus \$2.50 postage and handling. (US funds drawn on a US bank. Make checks payable to Tom Devlin.) All ICs are socketed, the printed circuit board is tin-lead plated epoxy-glass, and all parts are of high quality. It is covered by a 90 day parts and labor warranty. It is currently available for all V6.78 Compucolor IIs. Owners of V8.79 units will have to look inside your machines before ordering. If your logic board has a 16 pin UB3 PROM, you have the same Rev. 3 logic board as in V6.78

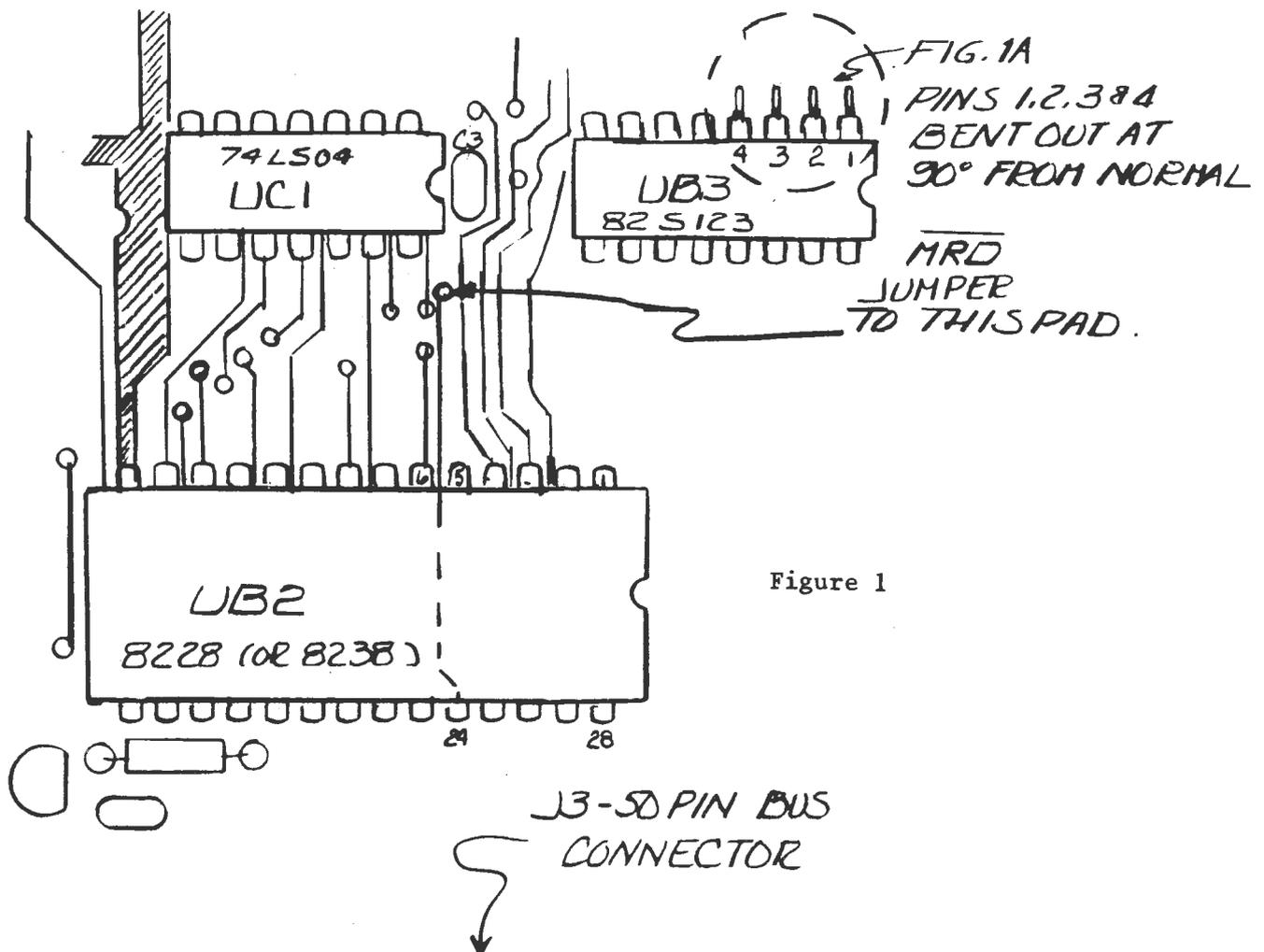


Figure 1

systems. If you also have an EPROM board with one EPROM on it, you will have to replace the UA5 masked ROM and UB3 PROM to let you remove and discard this EPROM board to free up the J9 connectors for the RAM card. We have a UA5 EPROM and UB3 PROM replacement kit for \$15.00 additional when ordered with the RAM card. V8.79 units with the Rev. 3 logic board and no EPROM board already have the cor-

rect UA5 and UB3 chips installed and are ready for installation of the RAM card. If your machine has a 20 pin UB3 PROM, you have a Rev. 4 logic board. These require no extra parts, but we have been unable to find a Rev. 4 board to make the installation drawings from. Perhaps we can make this information available in a future article. ■

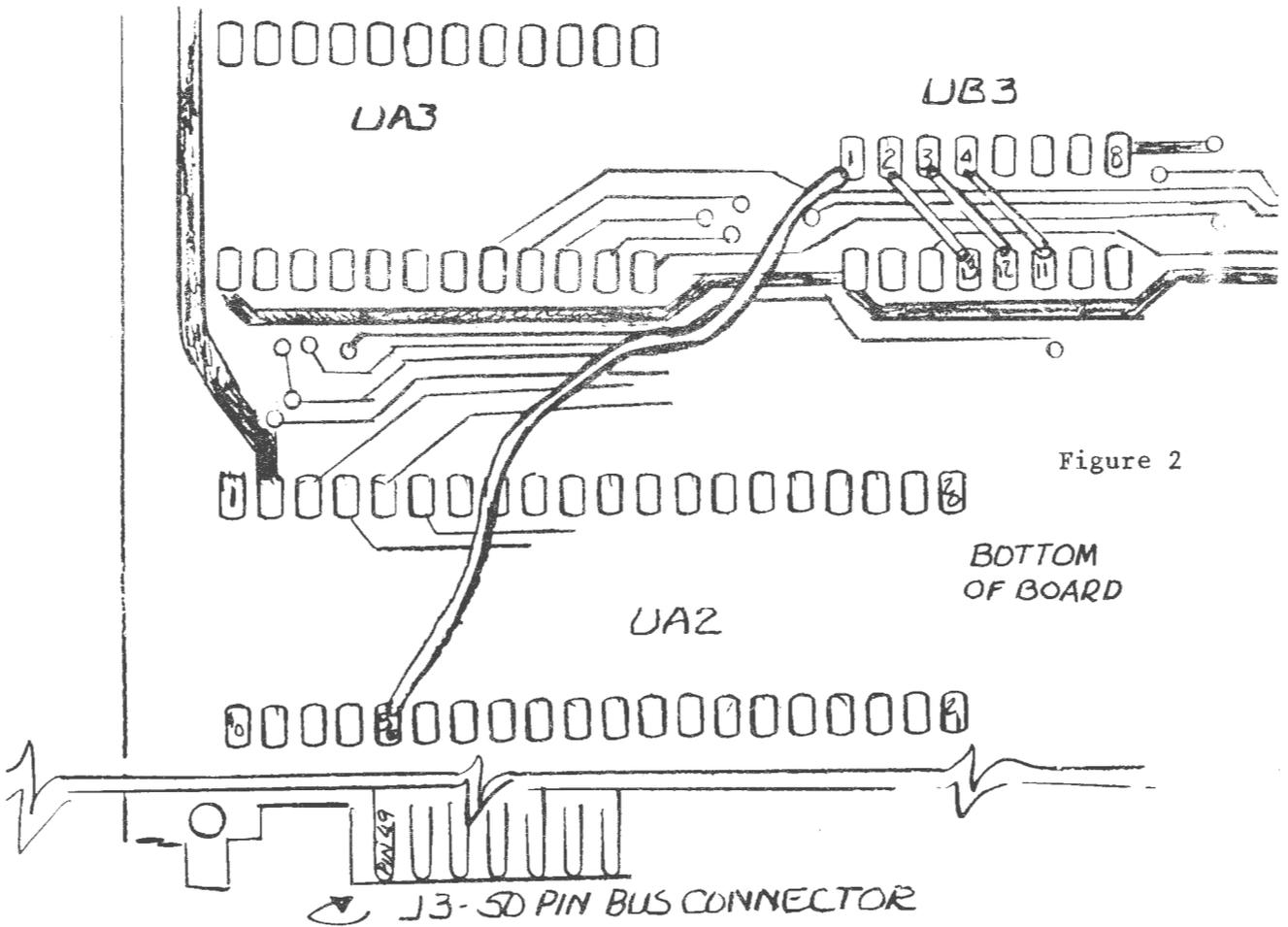


Figure 2

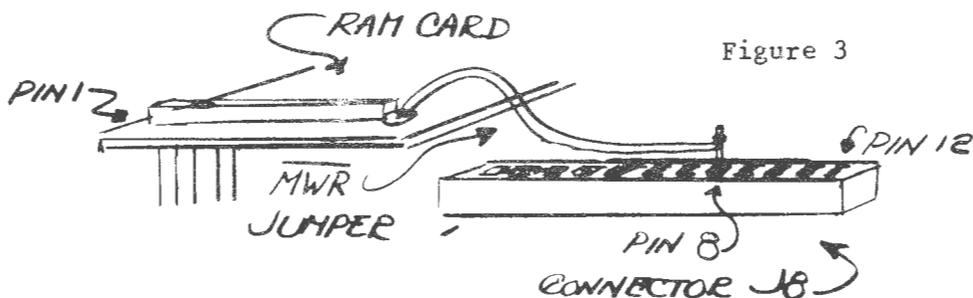
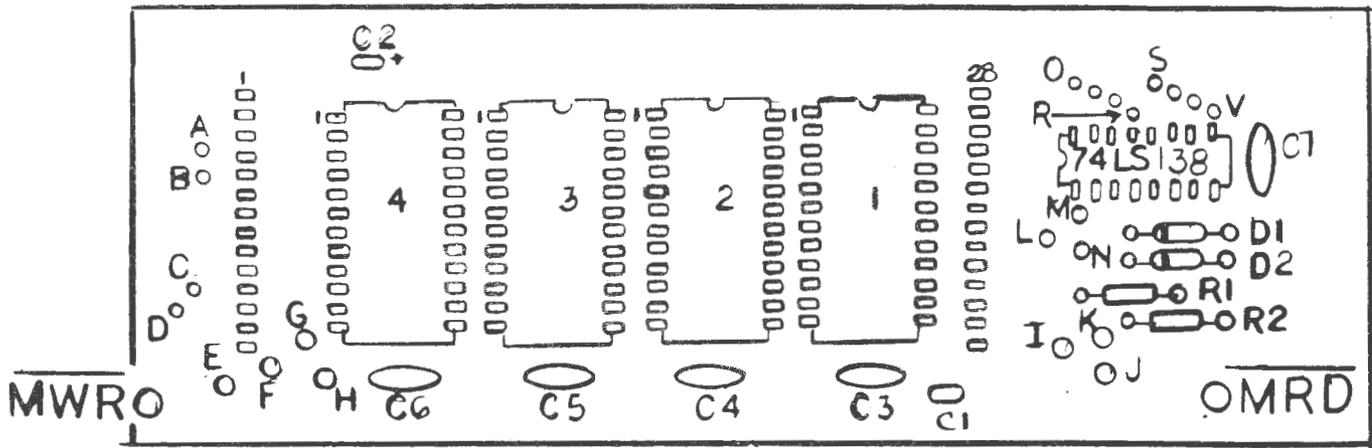


Figure 3



© 1982 by Tom Devlin

```

0 REM          'REORG' BY TOM DEVLIN 1982
1 REM          PROGRAM TO REORG 'FREDI'. REQUIRES 16K. WORKS FOR
2 REM          V3.20. WILL HAVE TO BE MODIFIED FOR OTHER VERSIONS.
3 REM          RESET TOP OF MEMORY WITH AN <ESC>-W WHEN DONE.
8
9 REM          MOVE TOP OF MEMORY DOWN
10 POKE 32940,191:POKE 32941,158:CLEAR 50
19
20 PLOT 12,6,2,15
30 INPUT "INSERT YOUR BASIC EDITING DISK AND HIT RETURN";A$
40 PRINT :PRINT "DO YOU HAVE AN <ESC> [P] JUMP TO 4000H?"
50 INPUT "      ( ALL V8.79 UNITS DO ) ";A$:PRINT
60 PRINT "THIS WILL TAKE ABOUT TWO MINUTES"
68
69 REM          GET BOTH VERSIONS INTO 16K AT THE SAME TIME
70 PLOT 27,4:PRINT "LOAD FRED16.PRG;01":PLOT 27,27
80 PLOT 27,4:PRINT "LOAD FRED32.PRG;01,9EC0":PLOT 27,27
88
89 REM          SPOT FCS ERROR VIA CCI CODE
90 IF (PEEK(33231) AND 7)=1 THEN 10
99
100 LO=44928:REM 0AF80H, FIRST BYTE 16K VERSION
110 HI=40640:REM 09EC0H, FIRST BYTE 32K VERSION
120 RC=16768:REM 04180H, FIRST BYTE RAM CARD VERSION
130 SZ=4189:REM 0105DH, PROGRAM SIZE
138
139 REM          MOVE 'FREDI' DOWN, CHANGING HIGH ADDRESS BYTES
140 FOR I=0 TO SZ
150   POKE RC+I,PEEK(LO+I)
160   IF PEEK(HI+I)-PEEK(LO+I)<>64 THEN 180
170   POKE RC+I,PEEK(LO+I)-110
180 NEXT
188
189 REM          SEE IF WE NEED TO USE <ESC> [^]
190 IF ASC(A$)<>89 THEN 250:REM YES
198
199 REM          DELETE USER JUMP SETUP (FILL WITH NOPS)
200 FOR M=16847 TO 16857
210   POKE M,0

```

```

220 NEXT
230 POKE 19521,80:REM CHANGE [^] TO [P] IN SIGN-ON MESSAGE
240 GOTO 260
248
249 REM SET USER JUMP TO 04000H IF NO <ESC> [P]
250 POKE 16848,0:POKE 16849,64
257
258 REM SET UP JUMP AT 04000H WHEN RUN
259 REM USING OLD TOP OF MEMORY ADJUST SPACE
260 DATA 62,195,50,0,64,33,15,66,34,1,64,0
270 FOR M=16864 TO 16875
280 READ D
290 POKE M,D
300 NEXT
308
309 REM CHANGE THE SIGN-ON MESSAGE TO 4000H, COLOR TO CYAN
310 DATA 6,52,48,48,48,72
320 FOR M=19468 TO 19473
330 READ D
340 POKE M,D
350 NEXT
358
359 REM SAVE 'FRED4' TO DISK AND RUN IT. (HIM?)
360 PLOT 27,4:PRINT "SAVE FRED4 4180 105D":PLOT 27,27
370 PLOT 27,4:PRINT "RUN FRED4"

```

Call for Information

An interesting use of ISC computers was devised by Mr. Shepard of Rochester (present whereabouts unknown) who marketed the machines with a horse race handicapping system. His disappearance leaves those owners completely out of the saddle. If you're one of those owners, **Colorcue** can serve as an information exchange for you. We will be happy to publish information (if someone will provide it for us) or, if you wish, publish names of users interested in contacting each other. Let us hear from you.

CLASSIFIED ADVERTISING

FOR SALE: Intecolor 3651. New in original carton. 32K, 5 1/4 inch disk. Disks (no games). Joystick option. \$2800 (list \$3800). George Wilson (404) 458-1431 (nights).

FOR SALE: CompuColor II, V6.78, 16K, 71 key keyboard, manual set, disks, including assembler, Compuwriter, Debug, games. Asking \$1000. Charles Lovejoy (800) 225-2465 x1365, weekdays 9-5.

FOR SALE: CompuColor II, V8.79, 32K, extended keyboard, modem, manual set, disks, games. Great condition. \$1200 or best offer. Pete Gammon (404) 393-9690.

In and Out of the Compucolor II

by Jane Devlin
3809 Airport Road
Waterford, MI 48095

Will someone explain what I'm doing, sitting here with a screwdriver in either hand and the Compucolor crouched in front of me like a threatening beastie? I'm more at home with knitting needles, crochet hooks or quilting blocks, so the idea seems to be that if I can get into and out of this computer without major damage to it, myself, or the immediate environment, then ANYONE can. Nervous? Who's nervous? ME! (I wonder if he'll promise to count to ten if he hears a scream and a crash? That should give me a good running start!)

First, clear a space to work (as I went along, I found it had to be about three times the size of the computer to allow the back to swing out of the way and the logic board to be pulled free). If you're working on anything that you'd like to keep scratch-free, cover it with a cotton sheet or towel, or newspapers. Nothing synthetic, please! The Compucolor II has MOS chips in it, and while with reasonable care there's no danger to them in taking the unit apart, a stray static charge might fry them. I'm told it's a good idea to get all the tools you'll need together (to just take the computer apart and put it back together, all you'll need is one regular screwdriver and one phillips head) and then stay put while you're working. (No fair scuffing your feet on the carpet first).

Disconnect the AC cord from the back of the set by grasping the connector (not the cord!) and wiggling it up and down while pulling. Find the screws holding the back on; there are three phillips heads across the top and a nut with a screwdriver slot at the very bottom. This last one is

at the end of a groove that resembles the trench in Star Wars and you need either a long screwdriver so the handle extends past the back or a lot of patience. You can tilt the set forward where it will remain fairly steady while you remove the bottom screw (Fig. 1). Remember to put the screws somewhere where you can find them again.

Now, before you remove anything else, note the two tiny green PC board tabs sticking through the back on either side of the edge connectors. These will be important in reassembling the computer. Tilt the unit back

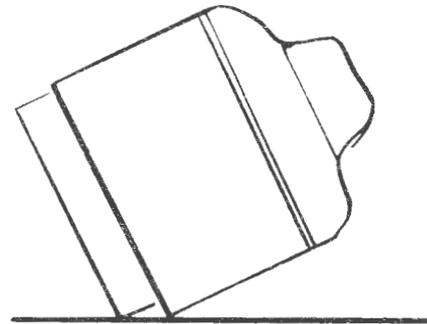


Figure 1

down on its base and look down the ventilation slits in the neck cover (that's the part that sticks out the farthest). You'll see a small board attached to the neck of a big, black glass bottle--the CRT. As you work the back off in a straight backward motion, be careful not to put any pressure on that small board and thus on the neck of the CRT. As you pull

the back free of those two PC tabs, there will be a slight jerk as the logic board drops a little; don't panic, you didn't break anything. You now have the back in your hands, but it is still connected to the computer by an 'umbilical cord' of wires. Swing the back gently to the left and it will be out of the way without disconnecting any more wires than necessary.

You can now see the interior of the computer. To the novice (like me) it's a confusing jumble of metal boxes, P.C. board, and wires running everywhere. To your left as you look into the set is a rectangular metal box with a cylindrical projection and a gray ribbon cable running off the P.C. board attached to the box's left side (Fig. 2,A). This is the working end of your disk drive. The big, black, glass 'bottle' in the center is the CRT, as I mentioned before (Fig. 2,B). Leave it alone. CRTs are capable of storing jolts that can make things very uncomfortable. This one has a 'bleeder' to drain high charges, but why take chances? On the far right (and facing to the right), partially protected by an aluminum frame that doubles as a heat sink for three power transistors (those are the little oval things that vaguely resemble a streamlined flying saucer), there is an intriguing board with all sorts of pots that seem to beg to be twitched or tweedled. DON'T--unless you like non-working computers, sleepless nights and gray hair. This is the infamous analog board, and it's awfully easy to blow up, so leave it alone!

(At this point, it's a good idea to touch the disk drive case or some other metal inside the set so you and the computer will be at the same electrical potential and thus protect the circuitry.)

The board sitting on the bottom of the case with three edge connectors pointing right at you is the logic board. This is the one that holds all the chips that make the computer compute, and it's here that most modifications are installed. For the sake of simplicity, I'm going to refer to the side with the three edge connectors as the rear of the board

(because it's in the rear of the set) and the side disappearing into the recesses under the CRT as the front of the board.

There are three wired connectors to the left side and the other end of the ribbon cable from the disk drive in a DIP socket just beside them. These four connectors have to be disconnected before the board can be pulled out, but before you start yanking, take the time to mark the connectors with masking tape, magic marker or anything handy so you know where to put them back. It can save a lot of hassle in the long run if they're keyed for replacement. (Two of ours are, and two aren't.) Carefully work the connectors free and pull the logic board toward you. (Now see why I said you would need a lot of space to work in?)

If you have a 32K machine, there will be a funny looking (it's upside down) small card on long spindley pins toward the right front quarter of the logic board. This is the 16K add-on RAM card and it is placed in two connector sockets known collectively as J8. To the right of this area is another set of widely spaced connector sockets (J9). If you have an EPROM board already installed, it will be in the J9 sockets and will look about the same as the 16K RAM card.

The chips have been designated sequentially from the right rear corner. The letter designation reads from right to left across the board and the numbers from the rear to the front. Therefore, the chip in the right rear corner is UA1, and in the front left corner you will find UG11. It makes finding chips on the board quite simple. (But when it comes to the schematic, it's everyone for themselves!)

Now for the fun part--getting everything back the way you found it. If you look at the front edge of the logic board between J8 and J9, you will see a small notch in the P.C. board. This was designed to fit onto one of the plastic fins molded into the case just under the CRT. Reconnect the four connectors, being careful to get them in the right places, and slip the board into the case. Don't expect

to be able to see to guide that notch onto the fin; you can't, and it has to be done by feel and blind luck. When it's engaged, the board will not move sideways. Let the board rest on the two rubber knobs on the bottom of the case.

Tilt the computer forward again and take a deep breath. Getting the back on has been known to drive people crazy. I struggled with trying to get everything aligned for almost ten minutes before I gave up and asked for help. Tom made it look easy, but then he's had a lot of practice. What I hadn't noticed was that there are two small wedges on the bottom edge of the back that have to fit into two notches in the case. You can sort of

look through the ventilation slits to keep track of what you're doing. Once those wedges are engaged, the back slips easily over the neck and onto the rim. The two tabs on the logic board have to be fit into their holes. As the board is slightly raised when they are in place, you have to lift it gently with a pencil or some other non-conductive object so the back can slip the final quarter inch into place. (Don't put too much pressure on the P.C. board or you may crack it.) Find where you hid the screws, tighten them in, replace the line cord and you're done!

As for me, I have an afghan, two sweaters and a quilt to finish. ■

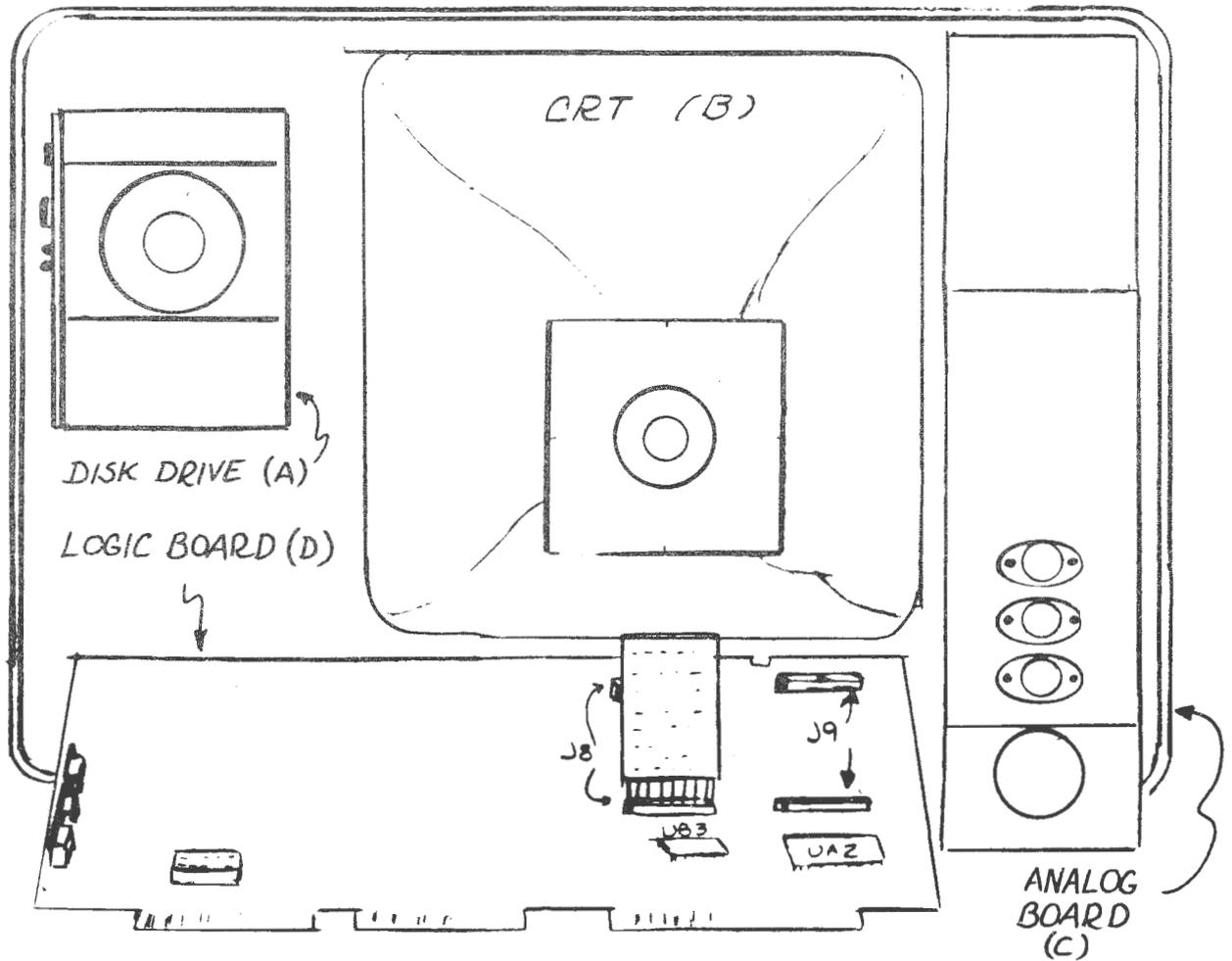


Figure 2

VERSATILE

EASY TO USE

ECONOMICAL



wiley

TRACE SET

LENS DESIGN PROGRAMS

FOR THE COMPUCOLOR II

The optical design aids which we offer now for the COMPUCOLOR have evolved from 20 years of experience in developing lens programs for the IBM 704, 7094, GE 235, 400, 635, ModComp II, and HP 1000 computers. Those of you that already have a COMPUCOLOR know that it is hard to beat in what you get for the money. These programs have been written to give you the benefit of this Computer Explosion. We currently offer a set of programs, two for the First Order Optics and one for the Rigorous Ray Tracing with spot diagrams. We plan to offer an ongoing series of programs and updates.

The current programs are written for a COMPUCOLOR with 16K memory and disc. The programs are provided on disc with a tutorial users manual in a notebook binder. The manuals are written with the beginner in mind so that a high school student can use them. The programs are written with the professional in mind so that anyone can do practical optical evaluation and design with a small investment in the computer and software.

FIRST and EFL are first order evaluation programs which take the input of the lens description in radii, spacings, and indices and compute the image positions and magnifications and plot a representation of the optical system on the screen. EFL computes the Front, Back and Effective Focal Length and shows the nodes on the Screen-plot. Mirrors and Catadioptrics are handled by all programs.

TRACE is a program to trace rays rigorously through spherical and conic section surfaces with optional decentrations. The resulting array of rays can be plotted on the screen as a spot diagram and listed by coordinates. The size and position of the best focus is computed, and images of three weighted colors can be evaluated. The inputs and outputs are simple and straightforward. In an hour of learning, you can be well on your way to constructive lens evaluation and design.

TRACE SET includes: FIRST, EFL, and TRACE programs Price \$250.00 (U.S. dollars)

ENCLOSE check or money order, Corporate Purchase Orders accepted.

Price subject to change without notice.

Combine Record Documentation with Record Access

by Alan D. Matzger
960 Guerrero St.
San Francisco, CA 94110

Most business applications involve keeping records, updating them from time to time and generating reports from them. Such applications involve several to many programs, each of which does something with the fields in a record--reading or modifying or writing them back. A while ago I found I was getting lost inside a record, forgetting or confusing which field was in what location. For a time I kept a formal, handwritten list of what was what in there. Things were a bit better, but lately I've hit on a method whereby such documentation is incorporated into the body of a program and is marvelously useful.

Recall the general format for the GET and PUT statements: PUT file <,record <,first>>; nexpr, sexpr [byte count]. I guess I didn't read all that too carefully at first, because I thought that all except the "nexpr" and "sexpr" had to be constants. My early programs were overflowing with GET and PUT statements for each and every field accessed, all with the appropriate numeric constants.

But one day in a flight of fancy I played with variables in place of constants and lo!, they worked splendidly. All at once, field access became a matter of a few subroutines and lots of DATA statements. And those data statements, appropriately REMmed, provided internal documentation for the byte-by-byte contents of the record. In what follows the variable FL is the file number used when the FILE "R" ... opened it; R is the record number; SB is the starting byte of the field; A\$ or A is the receiving

variable for string or number; and L is the length (byte count) of the field.

So now the DATA statement contains the file number (if appropriate), the starting byte and field length. Thus,

```
100 DATA 1,1,25;REM NAME
110 DATA 1,26,20;REM ADDRESS
120 DATA 1,46,15;REM CITY & STATE
.
.
.
200 DATA 2,5,4;REM AMOUNT OWED
```

The accessing subroutine might be:

```
400 READ FL,SB,L:IF L=4 THEN
    GET FL,R,SB;A:GOTO 420
410 GET FL,R,SB;A$(L)
420 RETURN
```

Be careful that none of the string fields are four bytes long; should one be, then use a different value for L to denote a numeric field. (Then be sure to remember that the actual field length is indeed four so that the SB of the next field is correct.)

The calling program RESTORES the proper line number for the field desired. It will also somehow have a value for R, the record number. (Here is perhaps the only advantage of BASIC's having only global variables.) You then simply GOSUB 400 to get the desired field. Thus, to print the City and State field,

```
870 RESTORE 120:GOSUB 400:
    PRINT A$
```

The PUTting subroutine may or may not need a READ statement. Mine usually don't because I GET or "pretend" to GET whenever I work on a field. The values for FL, R, SB and L remain unchanged. By "pretending" I mean that I keep other wanted information in the DATA statement, such as the X and Y coordinates of the field's position on the screen. So even though there may be nothing in the record to GET, I'll pretend and at the same time get the several field parameters.

My updating programs do usually include the X and Y coordinates and sometimes the color code; appropriate PLOT statements in the subroutine make for a pretty screen. The report programs have appropriate values for TAB positions. These extra pieces of information are READ into other vari-

ables in the one READ statement.

Having discovered that variables work as well as constants in GET and PUT statements, I think my programs are a whole lot cleaner and surely are closer to being self-documenting. But then, I suppose you knew that all along.

P.S. As long as we're talking about GETting and PUTting, here's something I found disconcerting the first time I ran across it. The LENGTH of the receiving string variable equals the byte count used in the GET or PUT. A two character string GETten by A\$(20) is 20 characters long--the two plus 18 trailing spaces. It took me a while to figure out why my print lines were overflowing. **■**

**** About Your Subscription ****

Most of our subscribers' subscriptions will come up for renewal after July. (Check your mailing label: the number indicates your last issue number. The June/July issue is issue number 6.)

We need your subscriptions to continue publication.

Since its beginnings in 1978, **Colorcue** has been financed by Intelligent Systems Corporation. That financial assistance will terminate with the June/July issue. Whether we will be able to continue publishing **Colorcue** will depend on whether you renew your subscription. There is a critical number of subscribers, below which **Colorcue** will not have the funds to continue. Let us know of your desire to see **Colorcue** flourish. Send us your renewal now so that we may know in advance where we stand and so that you can help guarantee the continued publication of what we believe to be an outstanding magazine for Intecolor/Compucolor users.

Subscription for one year (six issues) is \$12 in U.S., Canada and Mexico; \$24 elsewhere. Please make check or money order in U.S. funds payable to "Colorcue".

At the same time, why not take this opportunity to let us know what kinds of information you would like to see in **Colorcue** during the coming year? Would you prefer more hardware oriented articles? Tutorials? Programs? Applications? Games? Perhaps you have something specific in mind.

Assembly Language Programming

by

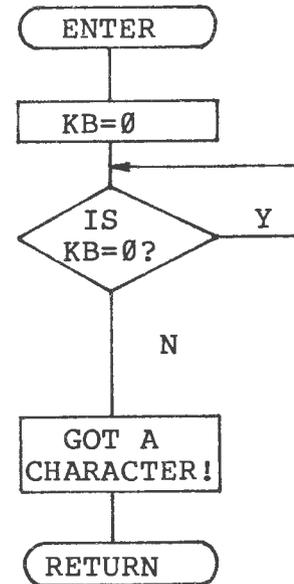
David B. Suits

Part V: Creating an INPUT Routine

We have discussed several ways of getting information from memory onto the screen. Let's take up the subject of getting information from the keyboard into memory. The very first thing we need to know is how to get one character from the keyboard.

Suppose you were working in BASIC but without the INPUT statement. How would you get data from the keyboard? If you can answer that question, then you can probably devise at least one perfectly useful method in assembly language. Under normal conditions, when a key is pressed, its code is stored in location 81FEH (33278 decimal), and then it is output to the screen. The codes range from 0 (control @) and 1 (AUTO) through 17 (red) and 18 (green), and then through the printable ASCII characters, 32 (space), 33 (!), and finally up through the special function keys, 240 (F0) through 255 (F15). The routine which reads the keyboard and stores the result in 81FEH if a key was pressed is a ROM routine which is automatically executed many times per second. That is, it interrupts your program, does its thing, and then returns control to your program. This happens in BASIC and in assembly language, and it occurs so fast that

you are usually unaware of it. There are, however, several methods of altering this routine so as to make it more adaptable to your particular needs. We'll get to that later. Right now, let's look at the quick and dirty method, which can be illustrated with the following flow chart. A simple BASIC subroutine is given in Listing



1. An equivalent assembly language program is given in Listing 2.

Last time I explained that labels may be used in place of constants (such as addresses) in an assembly language program. Thus, the instruction "JZ LOOP" is interpreted by the

```
1000 KB=33278
1010 POKE KB,0
1020 A=PEEK(KB):IF A=0 THEN 1020
1030 REM GOT IT!
1040 RETURN
```

Listing 1. A simple BASIC program to wait for a key press.

assembler as "Jump, if the zero flag = 1, to the instruction which is at the address LOOP:". But suppose I had not included the "LOOP:" label off to the left in Listing 2. How would the assembler know where to make the program jump to in response to "JZ LOOP"? It wouldn't. Similarly with KBCHAR. The instruction LXI H,KBCHAR will be translated as "put the two byte number represented by KBCHAR into the HL register pair". But what two byte number is that? It is the EQU instruction which gives the assembler that information. It says, in effect, that whenever the word "KBCHAR" appears, make it EQUAL (or EQUIVALENT) to--or, better yet, EQUATE it with 81FEH. (It is roughly similar to an assignment statement in BASIC, as in line 1000 of Listing 1; whenever KB appears, 33278 is meant.) By the way, since you can load the HL pair with any two byte number, and since all addresses are two byte numbers, you can even load HL with the value of LOOP. That is, the label "LOOP:" is merely an address in the program. So if we wanted to, we could have all sorts of instructions which referred to it. For example

```
JMP LOOP
LXI H,LOOP
CALL LOOP
```

and so on. Although you are not allowed to have duplicate labels, you may refer to any label you please as often as you please. Notice also that labels which specify the location of an instruction in your program (such as

GETCHA and LOOP) must always be followed by a colon. But assignment (that is, EQUATE) statements, such as "KBCHAR EQU 81FEH", have no colon; if you put one in, the assembler will signal an error. That's because KBCHAR does not indicate the location of an instruction in your program. Rather, it is merely a convenient way of writing the number 81FEH (or whatever number it is which you EQUATE to KBCHAR). That convenience makes your programs easier to read: "LXI H,KBCHAR" is a bit more meaningful than "LXI H,81FEH". This also makes things easier to change in case you need to do so. Most, if not all, of the EQUATE statements are traditionally grouped together near the beginning of the program or routine. If you were to learn that, say, 81FEH is not after all the correct number, then you need only change the appropriate EQUATE statement, and the assembler will make all the required corrections for you.

The EQUATE statement is one of several instructions which are sometimes called **pseudo-ops** because they don't end up in the final machine language program created by the assembler. Every time KBCHAR appears, it is translated into 81FEH. But the instruction "KBCHAR EQU 81FEH" itself disappears; it is an instruction **to the assembler** to act in a certain way. Thus, if the assembler were to assemble the get a character subroutine in Listing 2 starting at, say, 9000H, the machine language code it would produce would be the string of numbers in the Contents column below:

```

;Subroutine to get a character from the
; keyboard. NOTE: this is a workable,
; although not very efficient, way of
; doing it.

KBCHAR EQU 81FEH

GETCHA: LXI H,KBCHAR ;Load HL with 81FEH.
        MVI M,0 ;Set keyboard char=0.
LOOP:   MOV A,M ;See what's there.
        CPI 0 ;If zero, keep looping.
        JZ LOOP ;I.e., wait for key.
        RET ;Return with key in A.
```

Listing 2. An assembly language subroutine to get one character from the keyboard.

Address	Contents	
9000	21H	LXI H
9001	0FEH	81FEH
9002	81H	
9003	36H	MVI M
9004	00H	0
9005	7EH	MOV A,M
9006	0FEH	CPI
9007	00H	0
9008	0C2H	JNZ
9009	05H	9005H
900A	90H	
900B	0C9H	RET

Another important pseudo-op is the **ORG** (ORiGin) statement. It instructs the assembler that you wish the following series of instructions to be located in memory beginning at the address specified. A popular place for assembly language programs for ISC machines to begin is address 8200H (because that's where you go with ESC T). So you'll find a number of programs beginning with "ORG 8200H". Often there is only one ORG in the whole program, but occasionally there are more. Perhaps, for example, you want the program to live at 8200H, but for some reason you want a particular subroutine located in very high memory:

```

ORG 8200H
.
.
.
;main program
.
.
.
ORG 0FAC0H
.
.
.
;a subroutine
.
.
.
END

```

Finally, the **END** pseudo-op merely tells the assembler to stop assembling. It must be the last line in your assembly language program. Moreover, it must be followed by a carriage return (to signal the end of the line on which the END occurs). Failure to

include the carriage return is a common source of frustration for beginners; the assembler will insist that it didn't find the END statement, even though it's right there in front of its nose.

Logical OR

Back to reading the keyboard. (Forgive me for these frequent, but important, digressions.) I want to make the get a character routine more elegant. Several new 8080 instructions can help. The first is the **ORA <reg>** instruction. This takes the contents of <reg> and logically ORs them with the contents of the Accumulator; the result ends up in the Accumulator. (The contents of <reg> are unaltered.) What is a logical OR? It is simple. It says, for each bit in <reg> and the corresponding bit in the Accumulator, if either one is a 1, then the result is a 1 for that bit. Otherwise (i.e., if they're both 0), then the result is 0 for that bit. Some examples are given in Figure 1. Notice that the Accumulator may be logically ORed with itself. The result will be itself. Why bother? Ah! You see, the **ORA <reg>** instruction affects the carry, sign, parity and zero flags, and it is the zero flag which I am interested in here. If the Accumulator has all zeros, then **ORA A** sets the zero flag. Otherwise, the zero flag is not set. That is, we now have a nondestructive test to see if the Accumulator is zero. Since it is a one byte instruction, it can replace the more cumbersome **CPI 0** instruction which was used in the original get a character routine:

```

ORA  A      ;A=0?
JZ   _____ ;Yes.
.         ;No.
.
.

```

This is really no big deal, of course. We save one byte of space and negligible execution time. But I find this way of doing things to be more elegant and actually easier to read and understand. (There is also an **ORI <num>--OR immediate--** instruction which logically ORs the byte of immediate data with

the contents of the Accumulator.)

STA and LDA

A more important refinement to the get a character routine is possible by understanding alternative methods of moving data to and from the Accumulator. Instead of accessing a memory location indirectly via the address in HL, we can get to a memory location directly with the instructions **LDA <addr>**, which loads (i.e., copies) the contents of <addr> into the Accumulator, and **STA <adr>**, which stores (i.e., copies) the contents of

the Accumulator into memory location <addr>. The get a character routine will now look like Listing 3, assuming **KBCHAR EQU 81FEH**.

Exclusive OR

There's one more new instruction to investigate: the logical exclusive OR. The **XRA <reg>** instruction takes the byte in <reg> and exclusive ORs it with the byte in the Accumulator. (The contents of <reg> are unaltered.) What is an exclusive OR? It says for each bit in <reg> and the corresponding bit in the Accumulator, if exactly

1 0 1 1 0 0 1 1	Initial contents of Acc.
1 0 0 0 1 0 1 0	Contents of B.
1 0 1 1 1 0 1 1	Contents of Acc. after ORA B.
0 0 0 0 1 0 0 1	Initial Contents of Acc.
0 0 0 0 0 1 0 1	Contents of H.
0 0 0 0 1 1 0 1	Contents of Acc. after ORA H.
0 0 0 0 0 1 1 0	Initial contents of Acc.
0 0 0 0 0 1 1 0	Contents of Acc. after ORA A.

Figure 1. Examples of ORA <reg>.

1 0 1 1 0 0 1 1	Initial Contents of Acc.
1 0 0 0 1 0 1 0	Contents of B.
0 0 1 1 1 0 0 1	Contents of Acc. after XRA B.
0 0 0 0 1 0 0 1	Initial contents of Acc.
0 0 0 0 0 1 0 1	Contents of H.
0 0 0 0 1 1 0 0	Contents of Acc. after XRA H.
0 0 0 0 0 1 1 0	Initial contents of Acc.
0 0 0 0 0 0 0 0	Contents of Acc. after XRA A.

Figure 2. Examples of XRA <reg>.

```
GETCHA: MVI  A,0      ;A=0
        STA  KBCHAR  ;Contents of KBCHAR=0.
LOOP:   LDA  KBCHAR  ;Get byte at KBCHAR.
        ORA  A       ;=0?
        JZ   LOOP    ;Yes. Wait for a key.
        RET
```

Listing 3. The modified get a character routine.

one of them is 1, then that bit of the result is 1. Otherwise (i.e., if they're both 1 or both 0) the result is 0 for that bit. Examples are given in Figure 2. Notice that the Accumulator may be logically XRAed with itself. The result will necessarily be zero. Like the ORI <num> instruction, there is an XRI <num> instruction; it exclusively ORs the byte of immediate data with the contents of the Accumulator. And, like the ORA <reg> and ORI <num> instructions, the exclusive OR instructions affect the carry, sign, parity and zero flags. (Unlike the inclusive OR instructions, the exclusive OR instructions also affect the auxiliary carry flag.) Often the XRA A instruction is used instead of MVI A,0, simply as an elegant way of setting the Accumulator to zero. (But remember that it will affect the flags, whereas MVI A,0 will not.) Now we can write the get a character routine as in Listing 4.

Well, that routine is fine for getting a single character. But what about getting a whole string of characters? Let's write a routine which will act something like BASIC's INPUT statement. First, however, we must make yet another digression.

CHRINT

As I mentioned before, there is a keyboard scanning routine in ROM which is automatically executed a number of times every second. If a key is pressed, its code is put into 81FEH, the character is sent to the screen (echoed), and then the routine returns to your program. It is the part where the character is echoed on the screen that we want to interrupt so as to be able to disallow certain keystrokes. If, for example, the AUTO key is pressed, we might want to be able to ignore it. Or perhaps we want to ignore the down arrow or erase page

key, or.... A popular routine to do just this may be found in many assembly language programs for ISC machines. The location 81FFH holds a byte which the keyboard scanning routine looks at in order to see whether it should echo a character which has just come in from the keyboard. Now, if we could allow a character to come in from the keyboard and be stored in 81FEH but fool the computer at just the right time into thinking that a character hasn't come in, then we will effectively have cut off the echoing of characters. It so happens that location 81FFH is used as a keyboard character flag: if the contents of 81FFH=0, then the character won't be echoed on the screen. Somehow we will have to add these instructions at the appropriate spot:

```
XRA A
STA 81FFH
```

Fortunately, there is a "jump vector" at location 81DFH (33247 decimal). Ordinarily the contents of 81DFH are such that the keyboard scanning routine looks at that memory location and goes off to echo the character on the screen. We can interrupt that process at this point by putting the number 1FH (31 decimal) into this spot. Now the keyboard scanning routine, when it finds 1FH, will JMP to location 81C5H instead of its usual spot. What is at location 81C5H? We'll put still another JMP instruction there so that the routine will jmp to our own routine, namely, XRA A and STA 81FFH.

All this must sound a bit Rube Goldberg, but it is a versatile scheme which allows us a good deal of control over what happens. It is important to remember, however, that all this is occurring many times per second--interrupting our regular program in order

```
GETCHA: XRA A ;A=0.
        STA KBCHAR ;Contents of KBCHAR=0.
LOOP:   LDA KBCHAR ;Get contents of KBCHAR.
        ORA A ;Wait until
        JZ LOOP ; a key is pressed.
        RET ;Return with char in Acc.
```

Listing 4. Thrice modified get a character routine.

to do so. So we must be careful that our interrupt routine does not alter the flags or the contents of any register. We ought to make sure to save and later restore anything which will be affected. Hence, the CHaRacter INTerrupt routine, since it uses the XRA A instruction, must first save the Accumulator and flags. (See Listing 5.)

But first, how do we get the keyboard interrupt routine to jump to CHRINT instead of to its own echoing routine? Put the jump vector number 1FH into location 81DFH. That's easy:

```
MVI A,1FH
STA 81DFH
```

Now put a JMP to the CHRINT routine into locations 81C5H-81C7H. That is, we want:

```
81C5H  JMP
81C6H  ?
81C7H  ?
```

The JMP is easy. The 8080 JMP instruction in 0C3H (195 decimal). So:

```
MVI A,0C3H  ;='JMP'.
STA 81C5H
```

Now we need only put the low byte of CHRINT into 81C6H and the high byte into 81C7H. But what are those bytes? That is, what is the address of CHRINT? If it begins at, say, 9000H, then the low byte is 00H and the high byte is 90H. But is CHRINT really to be found at 9000H? Not necessarily. It is wherever the assembler puts it. We could force the assembler to put it there with an ORG statement, of course:

```
ORG 9000H
CHRINT: PUSH PSW
XRA A
STA 81FFH
POP PSW
RET
```

```
ORG 8200H
.
.
.
;Your program.
```

But we run a terrible risk: perhaps the program at 8200H will be long enough to overwrite 9000H and beyond, thus destroying our CHRINT routine. We could always put CHRINT way, way up in high memory, just to make sure.... But there's a far easier way to handle all of this.

What will LXI H,CHRINT do? It will load the HL pair with the address of the label CHRINT, whatever that happens to be. We don't have to know the address; the assembler figures that out for us, just as it also figures out where to jump to with something like JMP LOOP. Now that we have CHRINT's address in HL, we need to put it at 81C6H-81C7H. We could do it as in Listing 6. But there's a more elegant way.

SHLD and LHLD

The contents of the HL pair can be copied into memory with the SHLD <addr> instruction. (Store HL Direct.) The contents of L are stored at <addr>, and then the contents of H are stored at <addr>+1. Similarly, LHLD <addr> (Load HL Direct) will load L with the contents of <addr> and then load H with the contents of <addr>+1.

Just what we need! We have the address of CHRINT in HL, and we know where we want to put that address:

```
KBFLAG EQU 81FFH ;Location of keyboard char flag.

CHRINT: PUSH PSW ;Save Acc. and flags.
XRA A ;Acc. =0.
STA KBFLAG ;Pretend that no key was pressed.
POP PSW ;Restore Acc. and flags.
RET
```

Listing 5. The character interrupt routine.

```
LXI H,CHRINT
SHLD 81C6H
```

We can now write the routine which sets up the jump to the CHRINT routine. (Listing 7.)

Don't get confused between the LXI H and LHLD instructions. LXI H,<addr> will put <addr> into HL. LHLD <addr> will load the **contents** of <addr> and <addr>+1 into HL. Some examples are given in Figure 3.

The Input Routine

Using CHRINT is easy. At the beginning of your program, set up the jump vector 1FH and the jump to CHRINT. Once that has been done, CHRINT will have been spliced into the normal read-keyboard-echo-to-screen routine. Nothing will appear on the screen

unless we explicitly put it there. CHRINT, along with something like the get a character routine, form the basis of our input routine.

Just what do we want the input routine to do? Let's specify that it is to recognize only uppercase ASCII characters, from " " through "]". A backspace (left arrow) will cause the previous character to be erased, and a carriage return will end the input. Let's also specify that no more than 128 characters will be allowed.

Next time

Try your hand at writing a full fledged assembly language program for the input routine. I'll show you my version in the next issue. **C**

```
LXI H,CHRINT ;Get address of CHRINT.
MOV A,L ;Low byte in Acc.
STA 81C6H
MOV A,H ;High byte in Acc.
STA 81C7H
```

Listing 6. Setting up the jump to CHRINT. First version.

```
MVI A,1FH ;Jump vector 31.
STA 81DFH
MVI A,0C3H ;='JMP'.
STA 81C5H
LXI H,CHRINT
SHLD 81C6H
```

Listing 7. Setting up the jump to CHRINT. Final version.

Before LXI H,9000H			After LXI H,9000H		
<u>Address</u>	<u>Contents</u>	<u>HL</u>	<u>Address</u>	<u>Contents</u>	<u>HL</u>
9000H	0AH	73A4H	9000H	0AH	9000H
9001H	3DH		9001H	3DH	
Before LHLD 9000H			After LHLD 9000H		
<u>Address</u>	<u>Contents</u>	<u>HL</u>	<u>Address</u>	<u>Contents</u>	<u>HL</u>
9000H	0AH	73A4H	9000H	0AH	3D0AH
9001H	3DH		9001H	3DH	
Before SHLD 9000H			After SHLD 9000H		
<u>Address</u>	<u>Contents</u>	<u>HL</u>	<u>Address</u>	<u>Contents</u>	<u>HL</u>
9000H	0AH	73A4H	9000H	0A4H	73A4H
9001H	3DH		9001H	73H	

Figure 3. Examples of the LXI H, LHLD and SHLD instructions.

LARGEST SOFTWARE CENTER FOR COMPUCOLOR + INTECOLOR



INTELLIGENT COMPUTER SYSTEMS INC.

CHECK THESE EXAMPLES OF HIGH QUALITY SOFTWARE LISTED IN OUR CATALOG!

CTE SCREEN EDITOR

Order number: CT833
Price: US\$ 40.00

The new editor is a high speed machine language editor program which will move block, copy, delete or print text. CTE will change lower case characters to upper case yellow characters for those who do not have lower case installed. CTE will allow the insertion of cyan colored control characters (these characters are not counted as text, so printing will be normal) in the text for printer use. CTE has a global search-and-replace function on all ASCII characters 0-127 in both forward and reverse.

CTE has an active character, line page counter on the screen at all times and is updated on every key press or use of HOME key. CTE has all single key commands plus any keyboard can be used. CTE will allow you to set your printer's baud rate and print all the text or marked text only. CTE has a fast typematic keyboard action. CTE will set up it's text buffer to the memory size of your computer or honor the memory size question from BASIC at 80AC hex.

CTE is an excellent low cost wordprocessor.

DEBUG

Order number: CT841
PRICE: US\$ 30.00

DEBUG is a machine language EDITOR, DISASSEMBLER, HEX AND ASII DUMP, MASTER DEBUGGER, and more. Use any serial printer (with user selected print format), select Baud rates, print ASII/HEX dumps (in compressed or standard mode) to the printer & CRT with a floating address header for every 60 lines (printer) or 24 lines (CRT), move, copy, compare, verify memory blocks, disassemble programs, test all memory locations, search for a string of data (1-16) bytes long etc.

TERMII

Order number: CT821
Price: US\$ 70.00

TERMII is the most powerful communications program for the CCII and Intecolor. It includes: up/down loading of disk files (all types, with byte for byte verification); dumb terminal communications up to 9600 baud; send/receive and buffer all types of terminal outputs (binary, text, random files, pictures, etc.) at 110 to 4800 baud, print/save all buffered data to the disk, display all in-coming & out-going data to the CRT in HEX and/or ASCII.

You can also set upper/lower case character colors. Half/full duplex, parity ODD/EVEN/NONE, turn on/off ECHO BACK feature, Xon/Xoff protocol, and special menu HELP command. All commands can be toggled on/off. All keyboard type-outs are in one color and port inputs in another color (both selectable). Select large or small character mode, and many more features too numerous to mention.

in Australia: For Intecolor: Color Computer Systems, ATTN: Don Sforcina
58 Valley Road, Hornsby, NSW 2077, Phone: 02-476-2480

in Australia: For CompuColor: Trevor Taylor
1/6 McIntosh St., Chatswood, NSW 2067

in Germany: ICS, Attn: Heinz Sork,
Graue Burgstrasse 18, 5303 Bornheim 4, Phone: 02227-3242

in Canada: ICS, 21 Dersingham Cres., Thornhill, Ontario L3T 4P5
Phone: 416-889-8557 SOURCE: TCM101

in USA: ICS, 12117 Comanche Trail, Huntsville, AL 35803
Phone: 205-881-3800 SOURCE: TCB610

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

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

Address Correction Requested

An  Publication

Colorcue

RUBIK'S CUBE

```

F1
R1U1F3U3
B3U1R3U3
L1
D2R3D1R1
L3D2L1D2R1D3R3
B1D3B3
D1L1D3L3
R1D3R3B1R3B3R1
R1D3R3B1R3B3R1
B1D3B3L1B3L3B1
F1D2F3D3F1D2F3R1F3R3F1
D1
B1U1D3R1U1D3F3U2D2B3D1R1U1D3F1U1D3L3U2D2R3D3
B1U1D3R1U1D3F3U2D2B3D3F1U1D3L3U1D3B1U2D2F3D1
D2F1U2D2B1U3D1L3U3D1F3D2
L1U1D3B1U1D3R3U2D2L3D1F3U2D2B1U1D3R3U1D3F1D3
R2U1D3F2U3D1
    
```



RUBIK'S CUBE

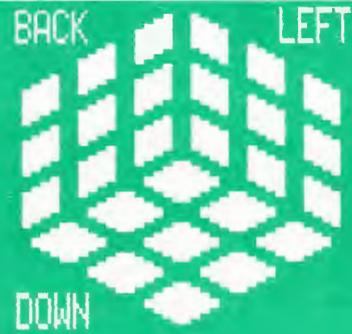
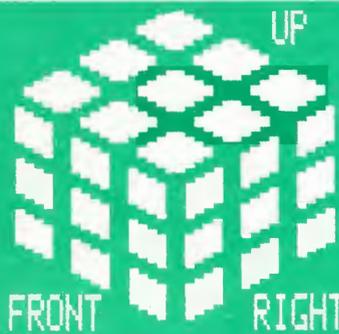
```

1 - INSTRUCTIONS
2 - MIX & PLAY
3 - MATCH & SOLVE
    
```

SELECT 2

DISPLAY EACH MOVE < Y/N >? Y

HOW MANY MIXES 2



RUBIK'S CUBE

```

1 - INSTRUCTIONS
2 - MIX & PLAY
3 - MATCH & SOLVE
    
```

SELECT

EVERY OTHER MAGAZINE WE'VE SEEN HAS HAD RUBIK'S CUBE ON IT'S COVER AND WE THOUGHT WE SHOULD, TOO.



Colorcue

**A Bi-monthly Publication by and for
Intecolor and Compucolor Users**

Editors:

Ben Barlow
David B. Suits

Compuserve: 70045,1062

June/July, 1982
Volume 4, Number 6

- 3 Editors' Notes**
 - 4 New Products from ISC**
 - 5 Rubik's Cube Demystified, by Roger Safford**
Have the computer solve it
 - 9 Assembly Language Screen Dump to MX80 Printer**
by Steve Reddoch
A fast version of a program originally in BASIC
 - 13 CALL Subroutine Linkage, by Ben Barlow**
Getting two languages together
 - 17 Keyboard Reading in BASIC, by Steve Perrigo**
The INS and OUTS of keyboard scanning
 - 20 Assembly Language Programming, by David B. Suits**
Part VI: The Input Routine
 - 25 Classified Colorcue Index, by James A. Kavanagh**
A comprehensive CCI code

 - 27 About Your Subscription**
-

Advertisers: You will find our advertising policies attractive. Write for details.

Authors: This is a user-oriented and supported publication. Your articles/tips/hints are required to make it go. Send your articles or write for information.

Colorcue is published bi-monthly by Intelligent Systems Corporation, with editorial offices in Rochester, New York. Editorial and subscription correspondence should be addressed to the Editors, Colorcue, 161 Brookside Dr., Rochester, NY 14618. Product related correspondence should be addressed to ISC, 225 Technology Park, Norcross, GA 30092, ATTN: Susan Sheridan. Opinions expressed in by-line articles are not necessarily those of the editors or of ISC. Hardware/software items are checked to the best of our abilities but are NOT guaranteed.

Editors' Notes

As we mentioned in the last issue, this issue is the last issue subsidized by ISC. From this point on, **Colorcue's** continued existence will depend on you. We need a certain number of renewals to make its publication worthwhile--not to make a huge profit, but to keep the enterprise from becoming a public service. Subscriptions have begun to roll in on a regular basis. We thank those early re-uppers (we'll hold your checks until we reach our magic number); your response has been gratifying. The suggestions for articles that we've received along with the renewals will be an excellent source for planning next year's content. Keep them coming.

Compucolor users interested in starting a user group in the northern New Jersey area should contact Amod Patwardhan, 21 Beachmont Terr., N. Caldwell, NJ 07006 (201) 226-8619.

The CUWEST user group has a fully working Scrabble game available for the cost of a disk and airmail postage. The game, written by Chris Teo, holds a dictionary of over 700 two and three letter words. The computer's moves take only about a minute. Send a check for \$9 payable to CUWEST to Doug Grant, CUWEST librarian, 2 Brookside Ave., South Perth, Western Australia 6151.

Doug Van Putte, who wrote the 3-D Graphics article in the Feb/Mar issue, suggests these changes to his program:

```
285 FOR L=1 TO 3:A=A(L)*3.14159/180:
    IF A(L)=0 THEN 295
290 GOSUB 560: GOSUB 450
295 NEXT L
550 delete this line
580 RETURN
```

A new version of 'THE' BASIC Editor has been released by Quality Software Associates, 21 Dersingham Crescent, Thornhill, Ontario, Canada L3T 4P5. (See original review in the Dec/Jan issue of **Color-**

cue). The new version features typamatic keys, improved cursor movement, and many other improvements. If ordered in EPROM or for the Devlin RAM board (see Apr/May **Colorcue**), 'THE' Editor will also contain a full set of FCS commands, an erase line command, an origin command to move a BASIC program, and a print command for programs and variables. A super program made even better!

We'd like to have some articles on languages. We have been using BASIC and assembly language for years. But what of the less well-known languages available? A tiny-c interpreter is available; so is FORTH and, of course, FORTRAN, about which we will have a series of articles in the coming months. Whatever happened to PILOT? Do you remember rumors of a compiled language from ISC called ALGAE? It was an interesting approach to a block structured, procedure-oriented language, but apparently it was never officially released. Anyone for LISP? It is available on a few microcomputers, but is anyone running LISP on an ISC machine? (One of your anxious editors is drooling at the thought.) There are rumors of a BASIC compiler (actually, two of them) in the works. One in Germany and the other in California. Will their authors and/or users please stand up? By the way, there's a money maker! Suppose you wrote a BASIC compiler, even a simplified, stripped down version. Double precision integers instead of floating point numbers. Maybe omit trig functions and random file handling. You'd still have a very marketable product. How much would you sell it for? Let's make it cheap, so as to encourage lots of sales. \$50? I'll take one for that price. (At \$75 you'd be testing the patience of my poor bank account, and at \$100 I would, reluctantly, walk away.) Now, how many can you sell? That's difficult to estimate, but at \$50 surely you'd sell at least five or six **hundred** of them. Let's see... \$50 x 600 = \$30,000. There's a tidy sum! We have had thoughts of just such a project for some time now, but it has always been pushed to the back burner. Perhaps, if no one else produces one (after all, a compiler is a difficult beast to create), we'll get back to work on it. Consider this a challenge. First one there wins. **■**

NEW PRODUCTS FROM ISC

TERMINAL SOFTWARE

ISC has released FlexWare, a terminal firmware package for ISC 8000, 8300 and 8900 machines. FlexWare's three asynchronous communications protocol modes with parameters definable from the keyboard or host allow ISC terminals to be interfaced to a variety of hosts. FlexWare's editing functions are accessed through ANSI sequences, and color via ISO standard color sequences. FlexWare's built-in expansion capabilities allow the user to interface his own firmware extensions and special functions by simply adding EPROMs.

Optional function keys can consist of keystrokes in local or on-line modes. Function keys can also be loaded with executable machine language patches. FlexWare has a disk I/O option that allows simultaneous capture of incoming data. The CRT display may be subdivided into as many as 32 discrete regions, each with its own roll mode, colors and character size. Single piece price is \$250.

GRAPHICS SOFTWARE

ISC has released IGS (Intelligent Graphics System), an advanced color graphics programming language with over 100 graphics commands which can be issued by any host computer. IGS may be used alone or driven by other computer languages. Features include seven type fonts, automatic labelling, scaling and rotation of grids and axes, rotate and zoom, and real-time plotting. IGS runs on ISC 8000R and 8300R series terminals and computers. Introductory price for IGS and 8301R terminal is \$3490.

TEKTRONIX 4014 EMULATOR

ICS's 8001R terminal with 8 color dot-addressable display is available with Tektronix 4014 emulation package. The ISC terminal emulates the Tektronix storage tube display by using the "shrink mode", in which the terminal receives vectors and fits them to its own 480 x 384 resolution. Color is added to the data by using the intensity bit in the 4014 data string. The emulation package also supports the screen dump to Printronix printers and Printacolor 8-color graphics printers. The 100 piece price is \$3275.

DEC VT52/VT100 EMULATOR

This software package allows the ISC terminal to behave like VT52 or VT100 in DEC environments. Color coding is used to emulate the VT100's reverse video, underline and bold. 100 piece price, including terminal, is \$2465.

IBM 3275 EMULATOR

ISC's 8001 terminal with 13" or 19" display allows for an 80 character by 24 line display with a 25th line for status. Software is now available so that the terminal may be attached to a host with binary synchronous communication (BSC). Color control is provided by the 3270 data stream. 100 piece price for terminal and software is \$4055.

UL-APPROVED TERMINAL

ISC has received Underwriters' Laboratory approval for a new version of its model 8001G 19" color terminal. The new terminal, the 8001G/82, meets UL 478 (electronic data processing) and FCC Class A regulations. While the digital circuitry of the 8001G/82 remains unchanged from the original 8001G, the analog circuitry has been completely redesigned, including a new switching power supply and switch-selectable 115 or 220 input voltage. 100 Piece price is \$2870.

NEW TERMINAL

The IS/2405 has an 80 column, 24 line color alphanumeric and graphics display with a single-board design based on the 8085 microprocessor. There are 12 function keys providing 36 programmable functions, each of which can store up to 40 characters. The terminal has a resident set-up mode for selecting physical I/O parameters which is retained by 2K of CMOS RAM even during power off. The IS/2405 also has 4K of screen RAM and 8K of EPROM for the operating system and an RS-232C printer port and a communications port that can be configured for RS-232C or current loop interface at 19.2K baud. The standard single-piece price is \$1995, but before October 1, 1982, the price is \$1200.

For information on any of the above products, contact Marketing Services, Intelligent Systems Corporation, 225 Technology Park, Norcross, GA 30092, (404) 449-5961. **C**

Rubik's Cube Demystified

by Roger Safford
15 Safford Ave.
Perry, NY 14530

The Rubik's Cube fad may be on the down cycle, but fortunately programming for fun is not. The program listing which follows is not just for the eight year old down the street who has no problem solving the cube in 37.5 seconds. He assuredly will find it a challenge trying to match up the colors with the handicap of not being able to tumble the Rubik's Cube in his hands. This program is also for those who don't have an eight year old to show them how to solve the Cube, and for those who followed the array of patterns described in the book but ended up with a colorful mess not unlike the

one they started with. Unlike the book, however, this is not a set of patterns. You enter the color orientation matching that of the Cube in your hands, and the computer will print a list of moves which will solve the cube. If there is no solution (if someone has moved the stickers or disassembled it), the program will let you know. The program will also let you try to solve it yourself by entering single or multiple move commands.

The program RUBIKS uses approximately 13K. Have fun. [Yes, and have fun typing it in! We suggest a team of eight year olds be hired for the job. --Eds.]

```

0 REM
10 REM RUBIKS CUBE : ROGER E. SAFFORD 06/15/1982
20 REM 15 SAFFORD AVE.
30 REM PERRY, N.Y. 14530
40 REM
100 CLEAR 1000:GOTO 1000
103
104 REM ----- SUBROUTINE TO DRAW CUBES
105 Z=Z+1:PLOT 6,C(Z),2,X,Y,242,X,Y-6:RETURN
110 Z=Z+1:PLOT 6,C(Z),2,X,Y,242,X,Y-1:PRINT A$:RETURN
115 Z=0:FOR T=1 TO 3:FOR U=1 TO 3
120 X=(T+U)*6+39:Y=112+(U-T)*6:GOSUB 110
125 X=T*6+40:Y=81+U*10-T*6:GOSUB 105:PRINT B$
130 X=T*6+50:Y=50+U*10+T*6:GOSUB 105:PRINT C$
135 X=T*6+86:Y=78+U*10+T*6:GOSUB 105:PRINT C$
140 X=T*6+104:Y=39+U*10-T*6:GOSUB 105:PRINT B$
145 X=(T+U)*6+85:Y=92+(U-T)*6:GOSUB 110
150 NEXT U,T:RETURN
153
154 REM ----- ROTATIONAL DATA
155 DATA 9,33,45,21,44,43,10,24,15,51
160 DATA 39,3,52,49,4,42,38,37,16,6
165 DATA 7,31,43,19,35,34,33,32,1,13
170 DATA 49,37,50,53,52,51,14,17,16,15
175 DATA 32,44,20,8,19,9,48,47,2,14
180 DATA 50,38,1,15,42,41,3,54,53,37
185 DATA 11,35,47,23,46,7,8,36,5,17
190 DATA 53,41,40,13,14,54,18,52,1,2
195 DATA 6,18,54,42,12,36,48,24,22,23
200 DATA 20,21,2,3,4,5,30,39,40,41
205 DATA 4,16,52,40,10,34,46,22,11,12
210 DATA 45,31,5,6,51,13,17,18,39,49
213
214 REM ----- ROTATION SUBROUTINE
215 FOR I=1 TO 5:READ P,Q,R,S:ON U GOTO 220,230,225
220 T=C(S):C(S)=C(R):C(R)=C(Q):C(Q)=C(P):C(P)=T:GOTO 235
225 T=C(P):C(P)=C(Q):C(Q)=C(R):C(R)=C(S):C(S)=T:GOTO 235
230 T=C(S):C(S)=C(Q):C(Q)=T:T=C(P):C(P)=C(R):C(R)=T
235 NEXT I:RETURN
240 GOSUB 115:FOR I=1 TO 1000:NEXT I
245 D=LEN(D$)-2:IF D<0 THEN RETURN
250 M$=LEFT$(D$,1):U=VAL(MID$(D$,2,1)):D$=MID$(D$,3,D)
255 IF M$="D" THEN RESTORE 195:GOTO 290
260 IF M$="U" THEN RESTORE 165:GOTO 290
265 IF M$="R" THEN RESTORE 155:GOTO 290
270 IF M$="L" THEN RESTORE 185:GOTO 290
275 IF M$="F" THEN RESTORE 175:GOTO 290
280 IF M$="B" THEN RESTORE 205:GOTO 290
283 IF M$="Q" THEN RUN
285 GOTO 295
290 IF U<0 AND U/4 THEN GOSUB 215:ON F GOTO 245,240
295 PLOT 3,0,17:PRINT M$:RIGHT$(STR$(U),1);
300 PRINT " IS AN INVALID MOVE "
305 FOR I=1 TO 2000:NEXT I:RETURN

```

```

9
309 REM ----- CORNER IDENT. DATA
310 DATA 6, 39, 4, 18, 40, 5, 42, 38, 3, 54, 41, 2
315 DATA 1, 14, 53, 13, 17, 52, 49, 16, 51, 37, 15, 50
318
319 REM ----- EDGIE IDENT. DATA
320 DATA 8, 47, 9, 44, 10, 45, 11, 46, 36, 23, 12, 22
325 DATA 24, 21, 48, 20, 7, 35, 31, 34, 43, 33, 19, 32
328
329 REM ----- MOVE BUILD SUBROUTINES TO LINE 905
330 D$="D2"
335 D$=D$+I$+"1D2"+I$+"3"
340 D$=D$+"D2"
345 D$=D$+F$+"3D1"+F$+"1":RETURN
350 D$=F$+"3D3"+F$+"1"
355 D$=D$+"D2"
360 D$=D$+I$+"1D3"+I$+"3":RETURN
365 D$="D3"
370 D$=D$+I$+"1"
375 D$=D$+"U1"+H$+"3U3":RETURN
380 D$="D3"
385 D$=D$+I$+"2":RETURN
390 D$="D1":GOTO 360
395 D$="D3":GOTO 360
400 D$=F$+"3"
405 D$=D$+I$+"3":RETURN
410 D$="U3"+F$+"3U1":RETURN
415 D$=G$+"1"
420 D$=D$+"U2"+G$+"3U2":RETURN
425 D$="D1":GOTO 385
430 D$=H$+"1D2"+H$+"3":GOTO 360
435 D$=G$+"3D2"+G$+"1":GOTO 355
440 D$=H$+"3D3"+H$+"1":GOTO 360
445 D$=I$+"3D2"+I$+"1":GOTO 360
450 D$="D1":GOTO 335
455 D$=F$+"3"
460 D$=D$+"U3"+F$+"1U1":RETURN
465 D$="D2":GOTO 370
470 D$="D1":GOTO 370
475 D$="D2":GOTO 385
480 D$=H$+"1":GOTO 375
485 D$=G$+"1"
490 D$=D$+"U1"+H$+"1U3":RETURN
495 D$=H$+"1"
500 D$=D$+I$+"1":RETURN
505 D$=I$+"3":GOTO 375
510 D$=G$+"3D2"+G$+"1":GOTO 345
515 D$="D1":GOTO 345
520 D$="D3":GOTO 345
525 D$="D3":GOTO 335
530 D$="U2"+G$+"1U2"
535 RETURN
540 D$=G$+"1D1"+G$+"3":GOTO 355
545 D$=F$+"1D2"+F$+"3":GOTO 345
550 D$=G$+"1D1"+G$+"3":GOTO 345
555 D$=H$+"1D2"+H$+"3":GOTO 340
560 D$=I$+"1D1"+I$+"3":GOTO 340
565 D$="":ON CM GOTO 535, 390, 360, 395, 355, 330, 535, 405, 410, 420
570 ON CM-10 GOTO 375, 425, 430, 350, 435, 440, 445, 450, 455, 365, 465
575 ON CM-21 GOTO 470, 370, 475, 535, 535, 535, 535, 535, 535, 400, 400
580 ON CM-32 GOTO 485, 495, 505, 385, 510, 340, 515, 345, 520, 525, 415
585 ON CM-43 GOTO 530, 490, 500, 460, 380, 540, 545, 550, 555, 560, 335
590 D$="":ON CM GOTO 430, 395, 390, 360, 395, 525, 480, 375, 405, 410
595 ON CM-10 GOTO 420, 475, 540, 445, 350, 435, 440, 330, 535, 370, 365

```

```

600 ON CM-21 GOTO 465, 470, 380, 535, 535, 535, 535, 535, 535, 415, 505
605 ON CM-32 GOTO 400, 485, 495, 425, 535, 520, 340, 515, 345, 335, 455
610 ON CM-43 GOTO 460, 530, 490, 500, 385, 510, 560, 545, 550, 555, 450
615 D$="":ON CM GOTO 540, 395, 355, 390, 360, 335, 415, 420, 375, 405
620 ON CM-10 GOTO 410, 380, 510, 440, 445, 350, 435, 525, 480, 470, 370
625 ON CM-21 GOTO 365, 465, 385, 535, 535, 535, 535, 535, 455, 495
630 ON CM-32 GOTO 505, 400, 485, 475, 430, 345, 520, 340, 515, 450, 535
635 ON CM-43 GOTO 500, 460, 530, 490, 425, 535, 555, 560, 545, 550, 330
640 D$="":ON CM GOTO 510, 360, 395, 355, 390, 450, 455, 410, 420, 375
645 ON CM-10 GOTO 405, 385, 535, 435, 440, 445, 350, 335, 415, 465, 470
650 ON CM-21 GOTO 370, 365, 425, 535, 535, 535, 535, 535, 485
655 ON CM-32 GOTO 495, 505, 400, 380, 540, 515, 345, 520, 340, 330, 480
660 ON CM-43 GOTO 490, 500, 460, 530, 475, 430, 550, 555, 560, 545, 525
665 D$="":ON S GOTO 690, 690, 695, 685, 695, 695, 695, 685
670 ON S-8 GOTO 685, 695, 685, 690, 680, 690, 695, 695, 695
675 ON S-17 GOTO 685, 685, 695, 695, 685, 685, 690, 695, 680
680 F$="B":G$="L":GOTO 700
685 F$="L":G$="F":GOTO 700
690 F$="F":G$="R":GOTO 700
695 F$="R":G$="B"
700 D$=F$+"1D3"
705 D$=D$+F$+"3"+G$+"1"+F$+"3"+G$+"3"+F$+"1"
710 RETURN
715 D$=F$+"1D2"+F$+"3D3"+F$+"1D2":GOTO 705
720 D$="":R=2:ON CM-7 GOTO 765, 770, 775, 780, 760
725 R=1:ON CM-19 GOTO 750, 755, 760, 745
730 R=2:IF CM=24GOTO 755
735 IF CM=36GOTO 745
740 ON CM-43 GOTO 785, 790, 795, 800, 750
745 D$="L1U1D3B1U1D3R3U2D2L3":ON R GOTO 775, 790
750 D$="F1U1D3L1U1D3B3U2D2F3":ON R GOTO 780, 795
755 D$="R1U1D3F1U1D3L3U2D2R3":ON R GOTO 765, 800
760 D$="B1U1D3R1U1D3F3U2D2B3":ON R GOTO 770, 785
765 D$=D$+J$+"F1U1D3L1U1D3B3U2D2F3"+K$:RETURN
770 D$=D$+J$+"D1R1U1D3F1U1D3L3U2D2R3D3"+K$:RETURN
775 D$=D$+J$+"F3U2D2B1U1D3R3U1D3F1"+K$:RETURN
780 D$=D$+J$+"F1U1D3L2U1D3B2U2D2F3"+K$:RETURN
785 D$=D$+J$+"F1U1D3L3U1D3B1U2D2F3"+K$:RETURN
790 D$=D$+J$+"F1U3D1R2U3D1L2U2D2F3"+K$:RETURN
795 D$=D$+J$+"F1U2D2B1U3D1L3U3D1F3"+K$:RETURN
800 D$=D$+J$+"D3L1U1D3B3U1D3R1U2D2L3D1"+K$:RETURN
805 ON R-1 GOTO 810, 815, 820, 825, 830, 835, 840, 845, 850, 855, 860
810 D$="R2U3D1R2U1D3":RETURN
815 D$="B2U1D3R2U3D1":RETURN
820 D$="U2D2F2U2D2B2":RETURN
825 D$="F2U3D1R2U1D3":RETURN
830 D$="L2U3D1F2U1D3":RETURN
835 D$="R2U1D3F2U3D1":RETURN
840 D$="B2U3D1L2U1D3":RETURN
845 D$="R2L2U1D3F2B2U3D1":RETURN
850 D$="F2U1D3L2U3D1":RETURN
855 D$="L2U1D3B2U3D1":RETURN
860 D$="R2U2D2L2U2D2":RETURN
865 D$="R3L1U1R3L1F1R3L1D1R2L2U1R3L1F1R3L1D1R3L1B2":RETURN
870 D$="R1L1U1R3L1F1R3L1D1R2L2U1R3L1F1R3L1D1R3L1B2R2":RETURN
875 D$="B3F1U1B3F1L1B3F1U1B2F2D1B3F1L1B3F1U1B3F1R2":RETURN
880 D$="B3F1U1B3F1R1B3F1D1B2F2U1B3F1R1B3F1D1B3F1L2":RETURN
885 D$="R3L3U1R3L1F1R3L1D1R2L2U1R3L1F1R3L1D1R3L1B2L2":RETURN
890 D$=""
895 D$=D$+"R3L1D1R3L1B1R3L1U1R2L2D1R3L1B1"
900 D$=D$+"R3L1U1R3L1F2":RETURN
905 GOSUB 865:PLOT 3, 0, WW, 6, 7:WW=WW+1:PRINT D$:GOTO 895

```

```

999 REM ----- INITIALIZATION ( START )
1000 DIM C(54)
1005 A$=CHR$(240)+"eeeDVVV"+CHR$(255)
1010 B$=CHR$(240)+"fff"+CHR$(255)
1015 C$=CHR$(240)+"UUU"+CHR$(255)
1020 FOR Z=1 TO 49 STEP 6:C(Z)=3
1025 C(Z+1)=1:C(Z+2)=7:C(Z+3)=5:C(Z+4)=4
1030 C(Z+5)=2:NEXT Z
1035 PLOT 27,24,6,6,14,12
1040 PLOT 3,35,1:PRINT "UP          BACK          LEFT"
1045 PLOT 3,21,14:PRINT "FRONT        RIGHT       DOWN"
1050 PLOT 3,2,2,6,3:PRINT "R U B I K ' S"
1055 PLOT 3,11,4,6,4:PRINT "C U B E":GOSUB 115
1060 PLOT 15,6,2,3,0,7:PRINT "1 - INSTRUCTIONS"
1065 PRINT "2 - MIX & PLAY":PRINT "3 - MATCH & SOLVE"
1070 PRINT :INPUT "   SELECT ";D$
1075 IF D$="1"GOTO 2200
1080 IF D$=""OR"2" AND D$=""OR"3"GOTO 1035
1085 PLOT 3,0,16
1090 INPUT "DISPLAY EACH MOVE ( Y/N ) ? ";E$
1095 E=1:IF E$="Y" THEN E=2
1100 IF D$="3"GOTO 1300
1198
1199 REM ----- MIX AND PLAY ROUTINE
1200 INPUT "HOW MANY MIXES ";U
1205 M$="RUBLDF":D$="":F=1:IF U>31 THEN U=31
1210 FOR J=1 TO U:D=INT(RND(8)*6+1)
1215 D$=D$+MID$(M$,D,1)
1220 D$=D$+RIGHT$(STR$(INT(RND(8)*3+1)),1)
1225 NEXT J:GOSUB 245:GOSUB 115
1230 PLOT 3,0,17,11,28,11
1235 PLOT 3,4,16:INPUT "ENTER MOVE ( TYPE 'QUIT' TO END ) ";D$
1240 F=E:GOSUB 245:IF F=1 THEN GOSUB 115
1245 GOTO 1230
1298
1299 REM ----- MATCH AND SOLVE ROUTINE
1300 GOSUB 1805:WW=16
1303
1304 REM ----- SOLVE TOP EDGIES
1305 F=E:F$="R":G$="B":H$="L":I$="F":FOR EG=1 TO 4
1310 RESTORE 320:EX=C(25+EG)*C(25):FOR EY=1 TO 12:READ R,S
1315 IF EX()C(R)*C(S) THEN NEXT EY:EY=C(25):GOTO 2000
1320 EY=12:NEXT EY:CM=R:IF C(S)=C(25) THEN CM=S
1325 ON EG GOSUB 590,615,640,565:IF D$=""GOTO 1335
1330 PLOT 3,0,7+EG,6,3:PRINT D$:GOSUB 245:GOSUB 115
1335 D$=F$:F$=G$:G$=H$:H$=I$:I$=D$:NEXT EG
1338
1339 REM ----- SOLVE TOP CORNERS
1340 FOR EG=1 TO 4:RESTORE 310:EX=C(25)*C(26)*C(29)
1345 IF EG(4) THEN EX=C(25)*C(25+EG)*C(26+EG)
1350 FOR EY=1 TO 8:READ R,S,T:Q=C(R)*C(S)*C(T)
1355 IF EX()Q THEN NEXT EY:EY=C(25):GOTO 2045
1360 EY=8:NEXT EY:CM=T:IF C(R)=C(25) THEN CM=R
1365 IF C(S)=C(25) THEN CM=S
1370 ON EG GOSUB 590,615,640,565:IF D$=""GOTO 1380
1375 PLOT 3,0,11+EG,6,6:PRINT D$:GOSUB 245:GOSUB 115
1380 IF C(EG+25)()C(EG+49) THEN EY=C(25):GOTO 2040
1385 D$=F$:F$=G$:G$=H$:H$=I$:I$=D$:NEXT EG
1388
1389 REM ----- CHECK FOR VALID CORNER COLORS
1390 FOR EG=1 TO 4:RESTORE 310:EX=C(30)*C(26)*C(29)
1395 IF EG(4) THEN EX=C(30)*C(25+EG)*C(26+EG)
1400 FOR EY=1 TO 4:READ R,S,T:Q=C(R)*C(S)*C(T)

```

```

1405 IF EX()Q THEN NEXT EY:EY=C(30):GOTO 2025
1410 EY=4:NEXT EY,EG
1413
1414 REM ----- PATTERN TO FLIP BOTTOM CORNERS
1415 S=0:RESTORE 310:FOR I=0 TO 2:FOR J=0 TO 2
1420 READ R:IF C(R)=C(30) THEN S=S+3*I*J
1425 NEXT J,I:IF S=0GOTO 1445
1430 PLOT 3,0,WW,6,2:WW=WW+1:GOSUB 665:PRINT D$
1435 IF WW>20 THEN PLOT 3,0,28:PRINT "NO SOLUTION":GOTO 2060
1440 GOSUB 245:GOSUB 115:GOTO 1415
1445 FOR CM=2 TO 5:IF C(CM)=C(CM+36)GOTO 1460
1450 NEXT CM:GOSUB 715:PLOT 3,0,WW,6,2:WW=WW+1:PRINT D$
1455 GOSUB 245:GOSUB 115:GOTO 1445
1460 FOR J=26 TO 29:IF C(CM)()C(J) THEN NEXT J
1465 EG=J-CM-24:IF EG<0 THEN EG=EG+4
1470 ON CM-2 GOTO 1485,1490,1495
1475 IF C(4)=C(40)GOTO 1510
1480 F$="B":G$="L":GOTO 1500
1485 F$="L":G$="F":GOTO 1500
1490 F$="F":G$="R":GOTO 1500
1495 F$="R":G$="B"
1500 GOSUB 715:PLOT 3,0,WW,6,2:WW=WW+1:PRINT D$
1505 GOSUB 245:GOSUB 115
1510 IF EG=0GOTO 1525
1515 D$="D"+RIGHT$(STR$(EG),1):PLOT 3,0,WW,6,2:WW=WW+1
1520 PRINT D$:GOSUB 245:GOSUB 115
1523
1524 REM ----- FINAL CORNER CUBIE CHECK
1525 FOR EG=1 TO 4:IF C(EG+37)=C(25+EG) THEN NEXT :GOTO 1540
1530 EY=C(26+EG):IF EG=4 THEN EY=C(26)
1535 EX=C(30)*C(25+EG)*EY:EY=C(30):GOTO 2020
1538
1539 REM ----- SOLVE BOTTOM EDGIES
1540 FOR EG=1 TO 4:RESTORE 320:EX=C(25+EG)*C(30)
1545 FOR EY=1 TO 8:READ R,S
1550 IF EX()C(R)*C(S) THEN NEXT EY:EY=C(30):GOTO 2000
1555 EY=8:NEXT EY:CM=R:IF C(S)=C(30) THEN CM=S
1560 R=INT(EG*EG*EG/3-EG*EG*1.5+EG/6+5.1)
1565 IF CM=12*RGOTO 1590
1570 J$="":IF EG(1) THEN J$="D"+RIGHT$(STR$(5-EG),1)
1575 K$="":IF EG(1) THEN K$="D"+RIGHT$(STR$(EG-1),1)
1580 GOSUB 720:IF D$=""GOTO 1590
1585 PLOT 3,0,WW,6,5:WW=WW+1:PRINT D$:GOSUB 245:GOSUB 115
1590 NEXT EG
1593
1594 REM ----- CHECK FOR VALID EDGIE COLORS
1595 RESTORE 320:Q=C(26)*C(29):FOR I=0 TO 3:READ R,S
1600 IF Q()C(R)*C(S) THEN NEXT I:EY=C(29):EG=1:GOTO 2000
1605 Q=I:I=3:NEXT I:FOR EG=1 TO 3:RESTORE 320
1610 P=C(EG+25)*C(EG+26):FOR EY=1 TO 4:READ R,S
1615 IF P()C(R)*C(S) THEN NEXT EY:EY=C(EG+26):GOTO 2000
1620 EY=4:NEXT EY,EG
1623
1624 REM ----- POSITION CENTER EDGIES
1625 RESTORE 320:P=C(26)*C(27):FOR I=0 TO 3:READ R,S
1630 IF P()C(R)*C(S) THEN NEXT I:END
1635 R=Q*3+I:IF I<Q THEN R=R+1
1640 I=3:NEXT I:IF R=1GOTO 1655
1645 GOSUB 805
1650 PLOT 3,0,WW,6,7:WW=WW+1:PRINT D$:GOSUB 245:GOSUB 115
1653
1654 REM ----- FLIP CENTER EDGIES
1655 R=1:IF C(8)()C(26) THEN R=R+4

```

```

1660 IF C(9) (>)C(27) THEN R=R+2
1665 IF C(10) (>)C(28) THEN R=R+1
1670 IF R=16GOTO 1695
1675 ON R-1 GOSUB 865,870,875,880,885,890,905
1680 PLOT 3,0,WW:IF LEN(D$)>50 THEN PRINT RIGHT$(D$,42)
1685 PLOT 6,7:IF LEN(D$)<50 THEN PRINT D$
1690 GOSUB 245:GOSUB 115
1695 FOR EG=26 TO 29:IF C(EG) (>)C(EG+10)GOTO 2060
1700 NEXT EG
1705 PLOT 3,10,30:PRINT "ALL DONE"
1800 INPUT "HIT RETURN WHEN READY ";WW$:RUN
1803
1804 REM ----- SUBROUTINE TO SET CUBIE COLORS
1805 PLOT 3,4,17:PRINT "ENTER COLORS VIA COLORED KEYS"
1810 FOR S=0 TO 5:FOR T=0 TO 2:FOR U=0 TO 2:Z=S+U*6+T*18
1815 ON S GOTO 1825,1830,1835,1845,1855
1820 X=(T+U)*6+51:Y=112+(U-T)*6:GOTO 1860
1825 X=T*6+46:Y=85+U*10-T*6:GOTO 1850
1830 X=T*6+64:Y=76+U*10+T*6:GOTO 1840
1835 X=T*6+92:Y=94+U*10+T*6
1840 GOSUB 1870:GOSUB 105:PRINT C$:GOTO 1865
1845 X=T*6+110:Y=103+U*10-T*6
1850 GOSUB 1870:GOSUB 105:PRINT B$:GOTO 1865
1855 X=(T+U)*6+97:Y=82+(U-T)*6
1860 GOSUB 1870:GOSUB 110
1865 NEXT U,T,S:GOTO 1900
1870 X1=INT(X/2)-1:Y1=32-INT(Y/4):PLOT 3,X1,Y1
1875 PLOT 6,0,32,26:INPUT " ";WW$
1880 IF WW$="0" THEN RETURN
1885 WW=ASC(LEFT$(WW$,1))-16:IF WW>7 THEN WW=WW-64
1888 IF WW<1 OR WW>7GOTO 1870
1890 IF WW<>6 THEN C(Z+1)=WW:RETURN
1895 GOTO 1870
1900 PLOT 3,4,17:INPUT "ANY FURTHER CHANGES ( Y/N ) ";WW$
1905 IF LEFT$(WW$,1)="Y" THEN PLOT 11,29,11:GOTO 1805
1910 FOR EG=7 TO 15:PLOT 3,0,EG:PRINT SPC(20):PRINT :NEXT EG
1915 FOR EG=16 TO 31:PLOT 3,0,EG,11:NEXT EG:RETURN
1998
1999 REM ----- ERROR MESSAGES FOR INVALID CUBES
2000 PLOT 3,0,29,6,66:PRINT "CANNOT FIND";
2005 GOSUB 2070:EY=C(25+EG):GOSUB 2070:PRINT " EDGIE"
2010 PLOT 6,2:PRINT "SOMEONE SWITCHED THE STICKERS"
2015 GOTO 1800
2020 PLOT 3,0,29,6,66:PRINT "SWAPPED COLORS AT";:GOTO 2030
2025 PLOT 3,0,29,6,66:PRINT "CANNOT FIND";:EY=C(30)
2030 GOSUB 2070:EY=C(25+EG):GOSUB 2070
2035 EY=EX/C(30)/EY+.1:GOSUB 2070:PRINT " CORNER":GOTO 2010
2040 PLOT 3,0,29,6,66:PRINT "SWAPPED COLORS AT";:GOTO 2050
2045 PLOT 3,0,29,6,66:PRINT "CANNOT FIND";:EY=C(25)
2050 GOSUB 2070:EY=C(25+EG):GOSUB 2070
2055 EY=EX/C(25)/EY+.1:GOSUB 2070:PRINT " CORNER":GOTO 2010
2060 PLOT 3,0,29,6,66:PRINT "YOUR CUBE MUST BE DISASSEMBLED"
2065 PLOT 6,2:PRINT "AND PUT TOGETHER CORRECTLY!":GOTO 1800
2070 ON EY-1 GOTO 2080,2085,2090,2095,2100,2105
2075 PRINT " RED";:RETURN
2080 PRINT " GREEN";:RETURN
2085 PRINT " YELLOW";:RETURN
2090 PRINT " BLUE";:RETURN
2095 PRINT " PURPLE";:RETURN
2100 PRINT " INVALID";:RETURN
2105 PRINT " WHITE";:RETURN

```

```

2198
2199 REM ----- INSTRUCTIONS
2200 PLOT 3,0,16:PRINT "MODES OF PLAY"
2210 PRINT :PRINT " SELECT MIX & PLAY AND I'LL MIX THE CUBE FO
R YOU TO SOLVE."
2220 PRINT :PRINT " SELECT MATCH & SOLVE AND YOU CAN MATCH MY
DISPLAYS TO"
2230 PRINT " YOUR HAND HELD CUBE THEN I'LL GIVE YOU A LIST OF
MOVES"
2240 PRINT " WHICH WILL SOLVE THE CUBE FROM THAT POSITION OR L
ET YOU"
2250 PRINT " KNOW YOU HAVE AN INVALID CUBE!"
2260 PLOT 3,12,30:INPUT "HIT RETURN TO CONTINUE ";WW$
2270 PLOT 3,0,16:PRINT "HOW TO ENTER MOVES"
2280 PRINT :PRINT " MOVES ARE ENCODED BY USING THE FIRST LETTE
R OF FACE TO BE"
2290 PRINT " ROTATED. THIS IS FOLLOWED BY A NUMBER FROM ONE (
1) TO "
2300 PRINT " THREE (3) TO INDICATE THE NUMBER OF QUARTER TURNS
IN A "
2310 PLOT 11:PRINT " CLOCKWISE DIRECTION AS YOU FACE THAT SIDE
"
2320 PLOT 11:PRINT :PLOT 11
2330 PRINT " FOR EXAMPLE: F3 TURNS THE FRONT FACE THREE (3) QU
ARTER"
2340 PRINT " TURNS. ( WHICH EQUALS ONE CCW TURN )":PRINT
2350 PRINT " MULTIPLE MOVES MAY BE WRITTEN AS FOLLOWS:"
2360 PRINT " D1U3R2D3U1 ( NOTICE NO SPACES! )"
2370 PLOT 3,12,30:INPUT "HIT RETURN TO CONTINUE ";WW$
2380 FOR I=30 TO 16 STEP -1:PLOT 3,0,I,11:NEXT I
2390 PRINT "MISC. FUNCTIONS":PRINT
2400 PRINT " A 'YES' RESPONSE TO THE DISPLAY EACH MOVE QUERY W
ILL DO"
2410 PRINT " JUST THAT AND ALLOW YOU TO SEE EACH MOVE. HOWEVE
R, WHEN"
2420 PRINT " DONE IN MATCH AND SOLVE MODE, MY SOLUTION SPEED I
S "
2430 PRINT " GREATLY REDUCED.":PRINT
2440 PRINT " IN MATCH & SOLVE MODE COLORS ARE SET BY PRESSING
THE"
2450 PRINT " COLOR KEY PAD (OR THE COLORED KEYS OF THE STD. KE
YBOARD), "
2455 PRINT " FOLLOWED BY RETURN. THE BLACK CURSOR WILL INDICAT
E THE"
2460 PRINT " CHANGE LOCATION.
2470 PLOT 3,12,30:INPUT "HIT RETURN TO START ";WW$:RUN

```

Is Your Subscription Expiring?

Assembly Language Screen Dump To MX80 Printer

by Steve Reddoch
1158 Via Bolzano
Santa Barbara, CA 93111

After typing in the BASIC language version of the screen dump program (Aug./Sept. **Colorcue**), I typed RUN and waited. And waited. BASIC was just too slow, or I'm too impatient. In any case, I sat down right then and rewrote the program in Assembly Language, and after working out the bugs, it's as fast as the original was slow.

The program follows the original to the letter (almost). The BASIC statements are included as comments, and serve to guide the reader, even one unfamiliar with the 8080 instructions, through the code. The translation looks easy. It was, but it's misleading. Much use was made of the fact that many of the numbers never get greater than 127 (one byte) and there are no floating point numbers. Not all BASIC programs would lend themselves to such a simple translation. Since most of the multiply instructions are by powers of 2, rotate instructions are used. The only tricky instruction is at line 9520, but

some study will lead to an understanding of its logic, if you're so inclined.

The program is set up to be CALLED from a BASIC program, and it occupies the top of memory. When RUN, it is brought into memory and protects itself by changing BASIC's end of memory pointer to just ahead of itself. It sets the CALL vector to point to the CALL entry point, sets the baud rate and returns to FCS, and then to the BASIC CALLer. When CALLED, the label named CALL: is given control, and the program begins to translate the screen (plot characters only) into Epson graphics characters in the standard character set, and print them line by line. Screen plot blocks are 2 dots wide and 4 dots high; Epson blocks are 2 dots wide and three dots high. When printed, the image looks a little higher than when on the screen, but that shouldn't prove objectionable. A sample BASIC test program is shown which exercises the routine. **■**

```
9 REM
BRING IN ASSEMBLY LANGUAGE ROUTINE
10 PLOT 27,4:PRINT "RUN EPSPLT":PLOT 27,27:CLEAR 1000
11 REM
PUT INTO VECTOR PLOT MODE
20 PLOT 12,2,242
21 REM
PLOT 50 RANDOM VECTORS
30 FOR X= 0 TO 50
40 PLOT RND (2)* 127,RND (2)* 127
50 NEXT
51 REM
PRINT SCREEN
60 Z= CALL (0)
```

```

;
; ASSEMBLY LANGUAGE SCREEN DUMP FOR MX-80 PRINTER
;
; BY STEVE W REDDOCH
;
; ASSEMBLY LANGUAGE VERSION OF A BASIC PROGRAM BY MARK
; FAIRBROTHER PUBLISHED IN AUG/SEPT 81 COLORCUE
; CALLABLE FROM BASIC PROGRAM BY:
; 10PLOT 27,4:"RUN SCRDMF.FRG":PLOT 27,27
; 11CLEAR 1000 (OR WHATEVER)
;
;
;
;
; 200Z=CALL(0):REM PLOT WHATEVER IS ON SCREEN
;
;-----
; PROGRAM LISTING IS FOR MACRO ASSEMBLER
;-----
;
; ENTRY START ;FOR REG. ASSMBLR, MAKE INTO ORG STMT
; AT DESIRED LOCATION E.G.
;
; ORG A000H
;
SCREEN EQU 7000H ;28672
;
LO EQU 3392H ;17C8H FOR 8.79
OSTR EQU 33F4H ;182AH FOR 8.79
;
LOFL EQU 81F9H ;81F9H
KEYBD EQU 3EB3H ;0911H
CALLV EQU 8202H ;8202H
MAXBAS EQU 80ACH ;80ACH
;
; LINKAGE ROUTINE
; WHEN PROGRAM IS RUN FROM BASIC, THIS ROUTINE ESTABLISHES
; THE NECESSARY LINKAGES AND POINTERS. IT ALSO SETS THE
; BAUD RATE OF THE RS232 PORT.
;
START: PUSH PSW
      PUSH H ;SAVE NEEDED REGISTERS
      PUSH D
;
      LXI H,START-1
      SHLD MAXBAS ;SET END OF BASIC IN FRONT OF PGM
      MVI A,(JMP)
      LXI H,CALL ;SET UP CALL VECTOR WITH
      STA CALLV ;JMP INSTRUCTION AND
      SHLD CALLV+1 ;ADDRESS OF DUMP ROUTINE
;
      LXI H,MSG0
      CALL OSTR ;SET BAUD RATE
;
      POP D
      POP H
      POP PSW ;RETORE REGS
      MVI B,0 ;INDICATE "NO ERRORS" TO FCS
      RET ;RETURN TO FCS
;
MSG0: DB 3,64,0 ;GET RID OF CURSOR
      DB 15 ;SMALL LETTERS
      DB 27,18 ;(ESC) (R)
      DB 7 ;BAUD RATE *** FOR DIFFERENT RATES,
      DB 239 ;END DATA SEE BELOW

```

```

;BAUD RATES:
;1 - 110 BAUD
;2 - 150
;3 - 300
;4 - 1200
;5 - 2400
;6 - 4800
;7 - 9600
;
; CALLED ROUTINE, ENTERED FROM BASIC BY
; Z=CALL (0) .....CALL VALUE NOT USED
;
CALL: PUSH PSW
      PUSH H
      PUSH B ;SAVE NEEDED REGS (AND UNEEDED AS WELL)
;
      LDA LOFL
      PUSH PSW ;SAVE LOFL VALUE
      MVI A,14
      STA LOFL ;NEW LOFLAG TO DIRECT ALL OUTPUT TO PTR
      MVI A,199
      OUT 8 ;LOCK OUT KEYBOARD (LOWER CASE Y'S)
      JMP BEGIN
;
EXIT: POP PSW
      STA LOFL ;RESTORE LOFL TO PREV VALUE
      MVI A,207
      OUT 8 ;UNLOCK THE KEYBOARD
      POP B
      POP H
      POP PSW ;RESTORE REGISTERS
      RET ;RETURN TO BASIC PGM
;
; MAIN ROUTINE
;
;9020 FOR Y = 0 (TO 126 STEP 3)
BEGIN: XRA A
      STA VY
S9020 EQU $
;9030 PLOT 14
      MVI A,14
      CALL LO
;9040 FOR X = 0 (TO 126 STEP 2)
      XRA A
      STA VX
S9040 EQU $
      CALL CI ;CHECK FOR INTERRUPT
      CPI 10
      JZ EXIT ;IF ONE, QUIT.
;9050 CH = 160
      MVI A,160
      STA VCH
;
      M = 1
      MVI A,1
      STA VM
;
      IM = 2
      MVI A,2
      STA VIM
;9060 IF Y = 126 THEN IM = 1
      LDA VY
      CPI 126
      JNZ S9060E
      MVI A,1
      STA VIM

```

```

S9060E EQU $
;9070 FOR I = 0 (TO IM)
      XRA A
      STA VI
S9070 EQU $
;9080 FOR J = 0 (TO 1)
      XRA A
      STA VJ
S9080 EQU $
;9090 SX = X + J
      LDA VX
      MOV B,A
      LDA VJ
      ADD B
      STA VSX
;9100 SY = Y + I
      LDA VY
      MOV B,A
      LDA VI
      ADD B
      STA VSY
;9110 GOSUB 9500
      CALL S9500
;9120 IF PT = -1 THEN M = M * 4: GOTO 9160
      LDA VPT
      CPI -1
      JNZ S9120E
      LDA VM
      RAL
      RAL ;ASSUME TOP 2 BITS CAN NEVER BE ON
      STA VM
      JMP S9160
S9120E EQU $
;9130 IF PT > 0 THEN CH = CH + M
      LDA VFT
      CPI 0
      JZ S9140 ;IF PT = 0
      JC S9140 ;IF PT < 0
      LDA VM
      MOV B,A
      LDA VCH
      ADD B
      STA VCH
;9140 M = M * 2
S9140: LDA VM
      ADD A ;DOUBLE IT
      STA VM
;9150 NEXT J (TO 1)
      LDA VJ
      INR A
      STA VJ
      CPI 2 ;COMPARE TO 1 STEP OVER LIMIT
      JNZ S9080 ;IF LESS, GO BACK TO TOP OF J LOOP
;9160 NEXT I (TO IM)
S9160: LDA VI
      INR A
      STA VI
      MOV B,A
      LDA VIM
      CMP B ;COMPARE VIM WITH VI
      JNC S9070 ;IF VIM >= VI (IF VI <= VIM)
;9170 PRINT CHR$(CH);
      LDA VCH
      CALL LO

```

```

;9180 NEXT X (TO 126 STEP 2)
      LDA VX
      INR A
      INR A ;BY 2
      STA VX
      CPI 128 ;COMPARE TO ONE STEP PAST LIMIT
      JNZ S9040 ;AND RESUME LOOP IF NOT OVER
;9190 PRINT
      MVI A,13
      CALL LO
      MVI A,10
      CALL LO
;9200 NEXT Y (TO 126 STEP 3)
      LDA VY
      INR A
      INR A ;INC BY 3
      STA VY
      CPI 129 ;COMP WITH 1 STEP PAST,
      JNZ S9020 ;GO IF LESS
;9210 PRINT CHR$(12)
      MVI A,12 ;FORMFEED
      CALL LO
      JMP EXIT ;LEAVE.
;9220-9240 REPLACED BY EXIT
S9500 EQU $
;9510 PT = 0
      XRA A
      STA VPT
;9520 AD = SCREEN+ 2* INT(SX/2) + 128* INT(SY/4)
      LXI H,SCREEN
      LXI D,0
      LDA VSX
      RAR ;SX /2
      ANI 07FH ;MAKE SURE TOP BIT'S OFF
      RLC ;2*INT(SX/2) IN A
      MOV E,A
      DAD D ;SCREEN + 2*INT(SX/2) NOW IN H
      LDA VSY
      RAR
      RAR ;SY/4
      ANI 03FH ;INT( MAKE SURE TOP 2 BITS OFF)
      RRC ;THIS IS THE SAME AS SHIFTING LEFT 7
      MVI E,0 ;BITS. WE DO IT BY SHIFTING RIGHT 1,
      JNC S9520A ;WITH SHIFTED BIT INTO CARRY, PUTTING
      MVI E,080H ;THE CARRY BIT INTO E, AND THE RIGHT
S9520A: ANI 07FH ;SHIFTED BYTE (WITH CLEARED TOP BIT)
      MOV D,A ;INTO D.
      DAD D ;ADD INTO ADDRESS
;9530 DA = PEEK (AD)
      MOV A,M ;PICK UP BYTE WHOSE ADDR JUST FOUND
      STA VDA
      CL = PEEK (AD + 1)
      INX H
      MOV A,M ;PICK UP CCI BYTE
;9540 IF CL >127 THEN 9570:REM IF DA IS A PLOT BLOCK
      ANI 128 ;CHECK FOR PLOT BIT IN CCI BYTE
      JNZ S9570 ;IT WAS ON
;9550 PT = -1
      MVI A,-1
      STA VPT ;IT WAS NOT.
;9560 RETURN
      RET
;9570 DO = 4* (SX AND 1) + (SY AND 3)
S9570: LDA VSX

```

```

ANI    1      ;SX AND 1
RLC
RLC      ;4 * (SX AND 1)
MOV     B,A
LDA     VSY
ANI     3      ;SY AND 3
ADD     B      ;A NOW IS 4 *(SX AND 1) + (SY AND 3)
;9580 PT = MS(DO) AND DA
MOV     E,A
MVI     D,0    ;DE NOW HAS DO IN IT
LXI     H,SQUTAB ;POINT TABLE
DAD     D      ;ADD IN DO OFFSET,
MOV     A,M    ;PICK UP BYTE,
MOV     B,A
LDA     VDA
ANA     B      ;A NOW HAS MS(DO) AND DA
STA     VPT
;9590 RETURN
RET

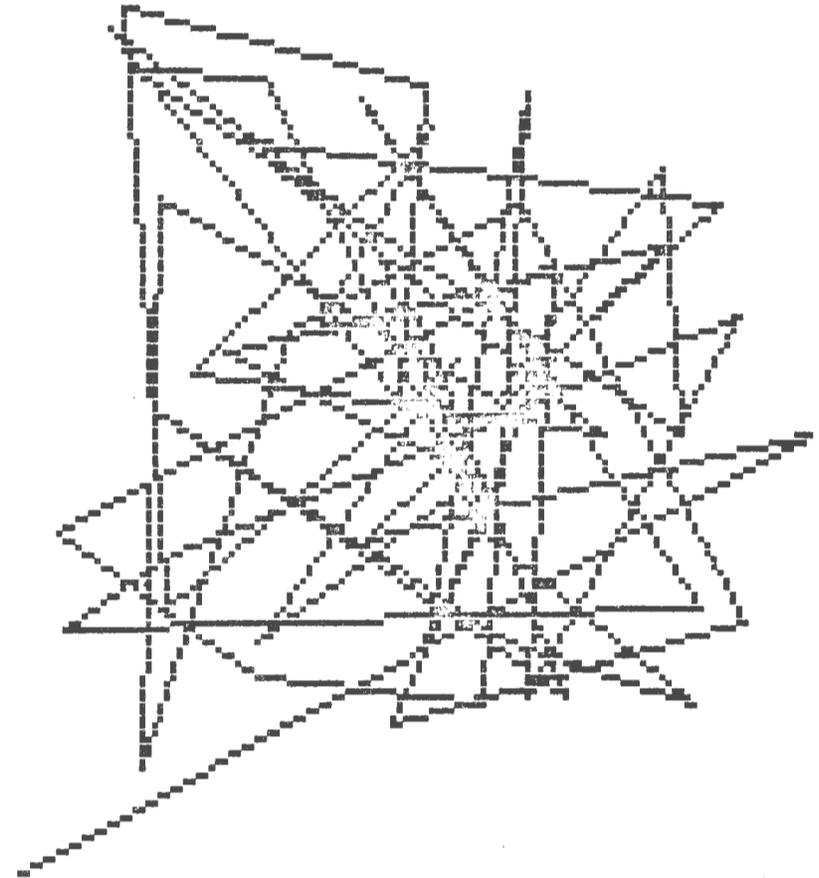
CI:    PUSH   H
      PUSH   D
      PUSH   B      ;SAVE REGS
CII:   XRA    A      ;DO A LOOP SO THERE
CII2:  DCR    A      ;WILL NOT BE A KEY BOUNCE
      JNZ    CII2   ;SPIN FOR 255 COUNTS
      CALL  KEYBD  ;THEN CALL KEYBD SCAN
      POP    B      ;RESTORE THINGS
      POP    D
      POP    H
      RET      ;RETURN

;*** A TABLE OF A PLOT SQUARE ***
SQUTAB: DB    1
        DB    2
        DB    4
        DB    8
        DB   16
        DB   32
        DB   64
        DB  128

;*** PROGRAM VARIABLES ***
VY:    DS    1      ;VARIABLE FOR ACROSS THE SCREEN LOOP CTL
VX:    DS    1      ;VARIABLE FOR DOWN THE SCREEN LOOP CTL
VI:    DS    1      ;VAR FOR VERTICAL SPOTS PER EPSON PLOT
        ;        BLOCK. 3 FOR ALL BUT LAST PASS, WHEN 1.
VJ:    DS    1      ;LOOP VARIABLE FOR HORIZ SPOTS PER BLOCK
VM:    DS    1      ;EPSON PWR OF 2 SPOT CODES. REL. TO 160
VIM:   DS    1      ;VERTICAL SPOTS LOOP LIMIT. WOULD BE 3
        ;        EXCEPT LAST PASS REQUIRES BUT 1
VDA:   DS    1      ;DATA CHAR FROM SCREEN
VCH:   DS    1      ;EPSON PLOT CHARACTER
VSY:   DS    1      ;CCII Y PLOT POS. (0 TO 127)
VSX:   DS    1      ;CCII X PLOT POS (0TO127)
VPT:   DS    1      ;RETURN VARIABLE FROM "IS DOT ON?" SUBR.
        ;        -1 IF NOT ON,
        ;        >0 IF DOT PRESENT

END      START

```



Cueties

```

PLOT 12,2:F=.3:X=64:Y=X:FOR J=0 TO 200:
A=F*J:X=X+SIN(J)*A:Y=Y+COS(J)*A:PLOTX,Y:
NEXT:PLOT 255

```

CALL Subroutine Linkage

by Ben Barlow

Volume 1, Number 1 of **Colorcue** contained the first assembly language CALLable subroutine (the famous "Scrolling Patch"), and many have appeared since. Recently we have published CALLable subroutines to sort strings and handle protected fields on the screen. It's about time to explain what they are, why they are, and how they work.

First, what are they and why are they used? A CALLable subroutine, or assembly language patch, is a piece of code similar to a GOSUB subroutine, but coded in 8080 assembly (i.e., machine) language. It is connected to a BASIC program, and the BASIC program CALLs it much like it GOSUBs to BASIC subroutines. Figure 1 illustrates this type of linkage. (Some computer languages offer the ability to "drop into" assembly language from a high level language, which would bring the assembly language routine "inline", but BASIC is not one of these.)

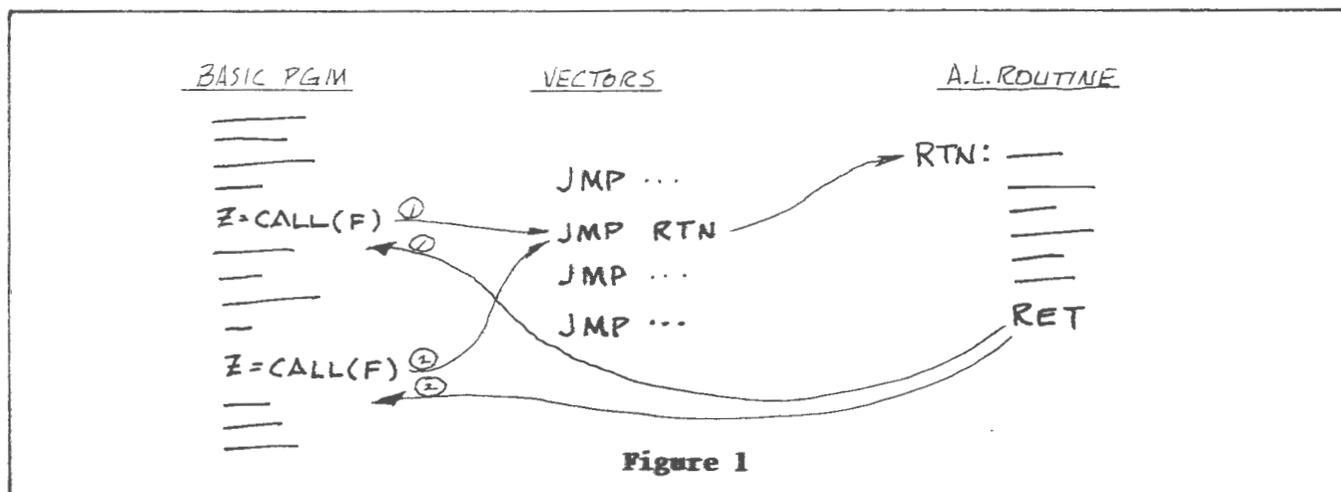
Such routines are most frequently used when speed is required or desired. The scrolling patch used in the Star Trek program, for example, had a need to manipulate a small portion of the screen as if it were a scrolling CRT terminal screen. BASIC could have done the job,

PEEKing and POKEing, but would have been just too slow to be effective. The sort routine (Oct/Nov **Colorcue**, p. 21) has been written in BASIC, but was speeded up by a factor of 50 when recoded into assembly language. The Soundware routines, on the other hand, wouldn't work if they were coded in BASIC; fast execution is needed to control the sound.

Figure 2 shows how CALLable subroutines are most often positioned in memory, stuffed away almost at the top. (Although the routines can be placed in other locations, the top of memory is the easiest and safest place. Other locations are left as an "exercise for the reader".) One obvious question is how to get the subroutine up there. The most common methods are:

(1) A series of DATA statements, with READs and POKEs from BASIC, suitable for short routines, but tedious, very tedious, for something the size of the sort routine. This type of placement is illustrated by the scrolling patch, but is useful only for small routines.

(2) Using FCS to LOAD a previously assembled routine and establishing the linkages through BASIC POKEs. This method saves most of the trouble of converting



statements, but it is still a non-optimum mix of function between the BASIC and the assembly language program.

(3) Using FCS to RUN a previously assembled program which then proceeds to set up its own linkages. This is the method used in recent **Colorcue** articles; it keeps the BASIC program small, reduces the chance for error (and the need to do hex to decimal conversions), and is simple to program.

Linkage to the subroutine is established by putting a JMP (jump) instruction into the CALL(x) jump vector location (33282 or 8202H for the hex fanatics) to

cause a jump to the beginning of the assembly language routine. Normally, this location contains a JMP to an error routine in BASIC. When BASIC interprets a CALL, it sets up registers as directed, and then JMPs to the CALL vector location. If the vector contains a JMP to your patch, it's entered. If not, the CALL immediately returns to the interpreter, and acts as a NOP (No Operation) statement. There are two ways to get the JMP to your routine into the CALL vector:

(1) POKE it in from BASIC. Assuming your routine begins at location OFE0DH in memory, the following code in Listing 1 will set up the vector.

(2) Store it from the assembly language routine. Given the same assumptions as above, the code would be as given in Listing 2.

Loading the routine into memory and establishing linkage to it is not the end of the initialization phase. Without taking precautions to protect the routine, BASIC would overwrite (read destroy) it at the first opportunity. By putting the routine at the top of memory, however, it is a simple matter to "fake out" BASIC and protect our routine. This is done by changing the system's pointer to the top of usable memory, making it point just in front of the routine we wish to protect. When BASIC subsequently allocates string space, it will check the "top of memory" location (32940, or 80ACH) and not go beyond. Figure 3 shows what this looks like. There are two ways to set this

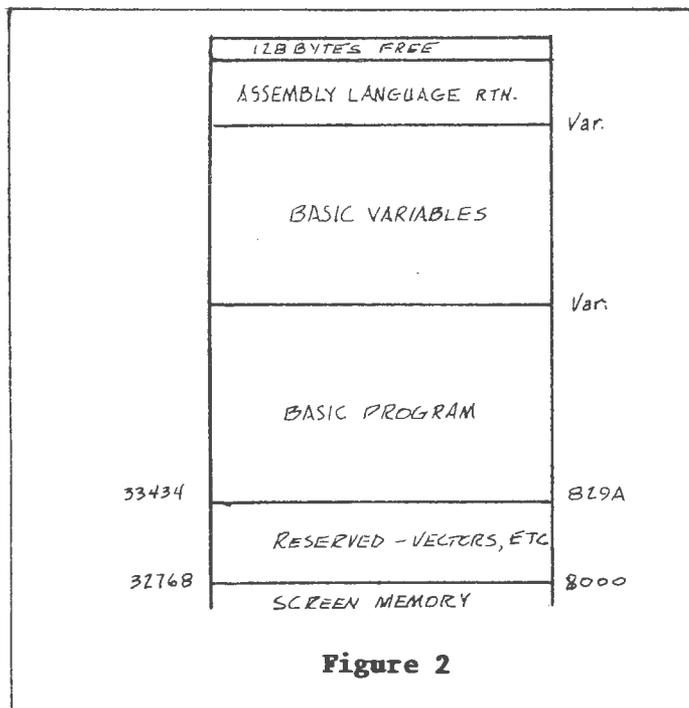


Figure 2

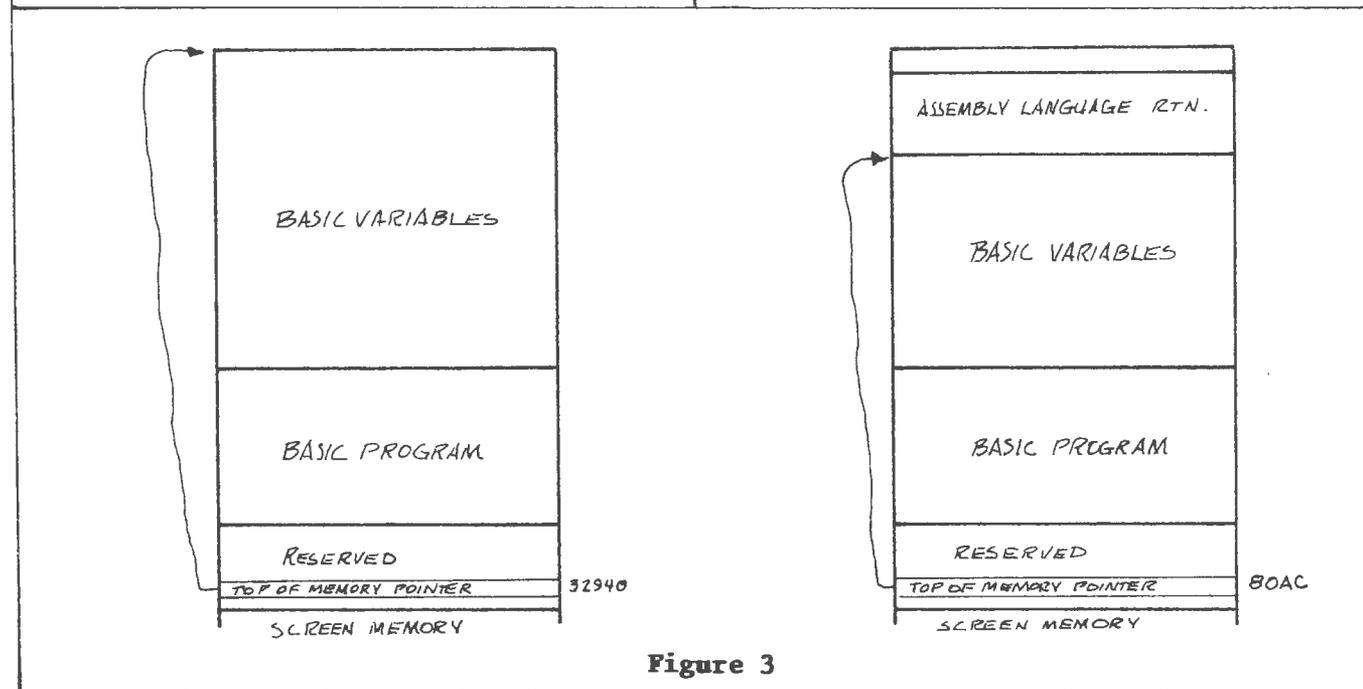


Figure 3

pointer (wouldn't you guess?):

(1) By executing BASIC POKES, as in Listing 3.

(2) By executing 8080 instructions in the setup section of the routine (Listing 4).

Once the routine is loaded into memory and linked to the BASIC program, it is used by executing the CALL statement in BASIC, e.g.:

```
60 Z = CALL (F)
```

As one might expect from an understanding of BASIC, the variable F is given to the subroutine, and the value returned by the subroutine is assigned to the variable Z. There are restrictions on F and Z; F will be converted to a two byte integer value (giving it a range from 0 to 65535), and Z will be similarly limited.

On entry to the subroutine, BASIC will have placed the value of F (after converting to an integer) into the 8080's D and E registers (E is the low order byte). To return a value, which the interpreter will

place into Z, the subroutine must exit (RET) with the value in the D and E registers. During the routine, H and L should be preserved.

Now that we understand the concepts, let's look at a small assembly language routine (Listing 5) that illustrates them. There are several interesting topics related to the ones discussed in this article:

Implied parameter passing, i.e. using BASIC variable values directly in an assembly language routine.

Linking to an assembly language routine from things other than CALLs, e.g. interrupt routines, such as User Timer no. 2 or the keyboard, the serial port, or BASIC output.

Implementing multiple functions in a single subroutine without POKING new addresses.

These are topics for a future article, though you can find examples of some of them in past **Colorcue** pieces. Write your editors and let them know what you'd like to see. **■**

```
15 RT = 65037:REM FE0DH
16 POKE 33282,195:REM The JMP op code
17 POKE 33283,RT AND 255:REM The low order byte of the address
18 POKE 33284,INT(RT/256):REM The high order byte of the address
```

Listing 1.

```
MVI A,195 ;The JMP op-code. Can be (JMP) if Macro-Assembler.
STA 33282 ;or 8202H
LXI H,CALL ;CALL is the entry point of the routine.
```

Listing 2.

```
19 TM = 61439:REM 0FDFH, one location in front of routine
20 POKE 32940,TM AND 255:REM The low order byte
21 POKE 32941, INT(TM/256):REM The high order byte
```

Listing 3.

```
LXI H,START-1 ;One byte in front of code
SHLD 80ACH ;Stuff into top of mem ptr (32940)
```

Listing 4.

```

;      SHORT ASSEMBLY LANGUAGE PROGRAM TO DEMONSTRATE
;      CALL SUBROUTINE LINKAGE FROM BASIC.
;
;      THE PROGRAM CHANGES SCREEN COLOR TO THE CCI CODE
;      PASSED TO IT BY ITS CALLER.
;
;      THIS PROGRAM CONSISTS OF TWO PARTS:
;      1. THE "RUN" PART, WHICH IS RUN THRU FCS FROM
;         BASIC, AND WHICH ESTABLISHES LINKAGES.
;      2. THE "CALL" PART, WHICH IS CALLED FROM A BASIC
;         PROGRAM, AND ACTUALLY DOES THE WORK.
;SYSTEM EQUATES - ADDRESSES OF THINGS WE'LL NEED

CALLVEC EQU 33282 ;ADDRESS OF VECTOR FOR CALL STATEMENT
TOPMEM EQU 32940 ;ADDRESS OF TOP OF MEMORY POINTER
SCREEN EQU 7000H ;ADDRESS OF SCREEN MEMORY

;      ORG 0F00H ;UNCOMMENT THIS (OR ANOTHER) ORG
;      STATEMENT FOR REGULAR ASSEMBLER

START:  PUSH  H
        PUSH  PSW ;SAVE REGISTERS WE'LL USE
;      WE'LL USE THE FCS STACK; 2 PUSHES WON'T HURT

        MVI  A,(JMP) ;GET JUMP OP CODE (CAN ALSO USE 195
;      FOR REGULAR ASSEMBLER)
        STA  CALLVEC ;PUT IT INTO CALL VECTOR
        LXI  H,CALL ;GET ADDRESS OF CALLABLE SUBROUTINE
        SHLD CALLVEC+1 ;AND PUT IT INTO VECTOR, GIVING:
;      JMP CALL

        LXI  H,START-1 ;GET ADDRESS OF LAST BYTE OF "SAFE"
;      ;MEMORY,
        SHLD TOPMEM ;AND PUT INTO BASIC'S POINTER.

        POP  PSW
        POP  H ;RESTORE REGISTERS THAT WERE USED,
        MVI  B,0 ;SET B=0 TO INDICATE NO ERRORS,
        RET ;AND RETURN TO FCS.

;      CALLED SUBROUTINE. ENTERED FROM BASIC WHEN
;      Z = CALL(F)
;      EXECUTED. REGISTER DE (REALLY JUST E) CONTAINS CCI CODE

CALL:   PUSH  H
        PUSH  PSW ;SAVE REGISTERS WE'LL USE

        LXI  H,SCREEN+1 ;POINT TO FIRST CCI IN SCREEN MEM
LOOP:   MOV  M,E ;PUT NEW CCI CODE DOWN

        INX  H
        INX  H ;STEP TO NEXT CODE; IT'S TWO BYTES AWAY.

        MOV  A,H ;CHECK TO SEE IF WE'VE GONE PAST 7FFF
        ANI  0F0H ;ZAP OUT LOW ORDER 4 BITS
        CPI  070H ;IF IT GETS TO 80, IT'S TOO FAR.
        JZ   LOOP ;STILL IN RANGE.

        POP  PSW ;DONE. RESTORE SAVED REGISTERS AND
        POP  H
        RET ;LEAVE

        END  START

```

Listing 5.

```

5 REM THE FIRST THING TO DO IS LOAD AND LINK TO SUBROUTINE
10 PLOT 27,4:PRINT "RUN QALP":PLOT 27,27:REM LINK UP TO AL SUBR
OUTINE
15 CLEAR 100:REM MUST BE HERE, RIGHT AFTER THE RUN CMD.
25 PLOT 27,24,6,2,15
30 FOR Y= 0 TO 31
35 PLOT 3,0,Y
40 FOR X= 0 TO RND (2)* 63
45 PLOT 32+ X
50 NEXT :NEXT
55 PRINT
60 PLOT 3,0,0,6,2,11:INPUT "ENTER CCI CODE: ";CCI
65 Z= CALL (CCI)
70 IF CCI< 256THEN 60
80 FOR CCI= 0 TO 255:Z= CALL (CCI):NEXT :END

```

Listing 6.

Keyboard Reading In BASIC

by Steve Perrigo
c/o Harding Lawson Associates
P.O. Box 3885
Bellevue, WA 98009

Several months ago, an article in **FORUM** (1) by B.F. Muldowney of Australia caught my eye. The article was titled "Advanced Keyboard Reading" and outlined a method to interrogate the keyboard for single or multiple key closures. At the same time I was looking for a simple joystick modification that could provide more than up, down, right and left. The solution was in the use of the Atari joystick and a keyboard scanning technique. An article describing keyboard scanning for the joystick appeared in the March/April issue of **FORUM**. An update was published in the May/June issue.

Since the publication of articles regarding joystick modification, software using joysticks has begun to appear on the market. Most notable (as of this writing) is **CHOMP** (2), a version of Pac-Man. As a result of the article explaining a method to install and use the joystick and keyboard scanning technique, I received several requests to explain a simplified version of how to read the keyboard in BASIC. It should be noted that most of this material is a representation of material from Muldowney's article.

To understand keyboard reading you must look at the keyboard from the computer's point of view. Figure 1 is a schematic drawing of just that. To understand this article you should rely heavily on this Figure. In addition, it is very important to note that this method only determines whether or not a particular key is physically depressed at any one time. It does not matter whether that key is in a shift, control or command status. That is why on the schematic you may not see some keyboard characters that appear on your key caps. In all cases I have listed the non-shifted, default characters represented by a key closure.

As depicted in Figure 1, the keyboard is accessed by 16 lines in, any of which may be interrogated individually, and one line out which will tell you the status of the keys along the interrogated line. The keyboard is interrogated through port 7 using the `OUT 7,"line number"` statement. For example, the statement `OUT 7,10` will send a signal through line 10 to interrogate the closure/non-closure status of the 5, E, U, F5, AUTO and MAGENTA keys. Note that only one of the 16 lines may be interrogated at any one time.

After the keyboard is interrogated with `OUT 7,"line number"`, you must then check for an answer from the keyboard. This is done by checking the value in port number 1 by means of the `INP(1)` statement. The value is retrieved using the form `X=INP(1)`, and X will be in the range 128-255. Note that in Figure 1 each of the horizontal grid lines is assigned a value in the box on the far right. These values are 1, 2, 4, 8, 16, 32 and 64. (Notice any relationship among those numbers?) When a key is closed along an interrogated vertical line (i.e., `OUT 7,"line number"`), the ultimate result is that the value 255 is decremented by the value assigned to the corresponding horizontal grid line. Now an example. Suppose you interrogate line 10 as described above with the `OUT 7,10` statement. If, for example, the E and F5 keys are both closed, then the corresponding value of X that you retrieve with the `X=INP(1)` statement will be 245. That is, 2 for the E closure and 8 for the F5 closure for a sum of 10, which is subtracted from 255, resulting in 245. I'll talk more about simultaneous closures later.

Now for another example. Suppose you wanted to detect the closure of any of the keys 2, 4, 6 or 8. How would you go about it? Answer: scan for each closure

independently. First scan line 13 for the closure of 2, then line 11 for the closure of 4, then line 9 for the closure of 6, and finally line 7 for the closure of 8. Immediately after executing each OUT 7, "line number" statement, check the value of INP(1) for a value of 254 (=255-1) indicating that the key is closed.

Prior to scanning the keyboard, you will want to lock out the keyboard (and some other built in routines) by an OUT 8,0 statement in your program. In this mode all the keys, including AUTO, ERASE PAGE, ATTN/BREAK, etc. may be depressed and their intended functions will be ignored by the computer. With this lock out command your program will run, but no characters from the keyboard will be echoed to the screen, thus keeping a "clean screen" during program execution. At the end of program or function execution you will want to reenable the keyboard with OUT 8,255.

Try the routine in Listing 1 to check the values of the keys depressed. Make sure you understand why it operates the way it does before you proceed.

After using this short routine you should have a feel for how this system monitors the keyboard for single key closures. Now try the routine with multiple closures and examine the results. Depress the 4 key and keep it depressed. Now add the D key, then the T key and so forth, pressing all of the keys along one of the lines into the keyboard. You will see that the same line is being interrogated and that the decremented value changes as the sum of the multiple closures changes.

Of course a slick method like this can't exist without a hitch. And here it is. There are times when the keyboard can signal the closure of a key that is not really depressed. Refer to the schematic. Suppose that you simultaneously close the H, G and W keys while you are monitoring along lines 7 and 8 for key closures. As usual, while interrogating along line 8 for closures, the value of INP(1) will be 249. (If you don't understand this yet, return to go and start over.) However, when you interrogate along line 7, the value of INP(1) will also be 249, indicat-

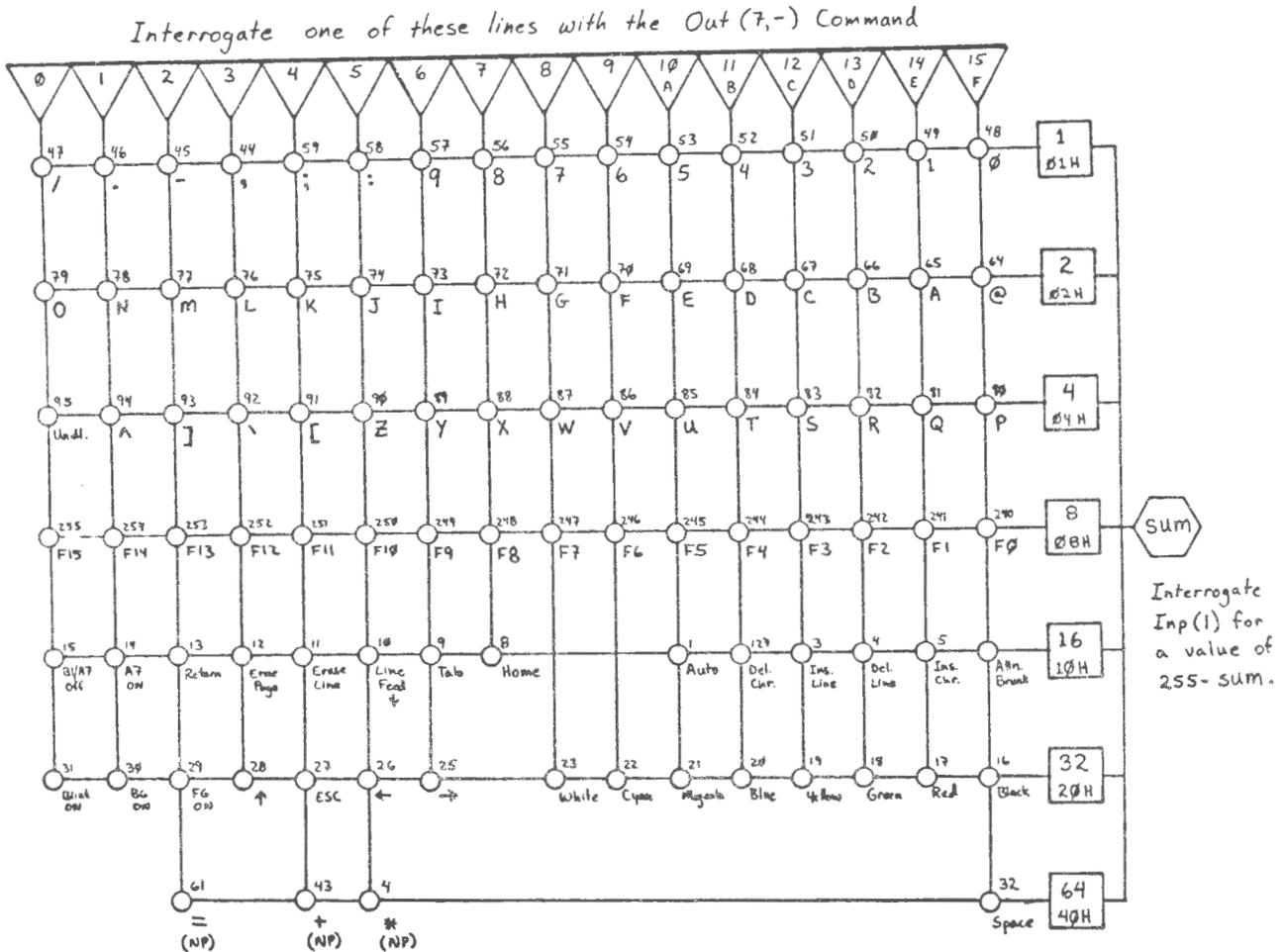


Figure 1. Keyboard scanning lines and decrement values.

ing that both the H and X keys are depressed. Why? Because when you interrogate the keyboard with OUT 7,7, the signal travels down line 7 and then, because the H key is depressed, the signal travels along the horizontal line which decrements the value by 2. But when it reaches the closed G key, the signal also goes down line 8 to the closed W and out along the next horizontal line, which decrements the value by 4. Simple? Not really; but it does make sense. Needless to say, when designing software using this technique, be sure to avoid this problem.

Now you should have a general idea of how the technique works. As you can see, it is a cumbersome method of getting a character from the keyboard. The real value of keyboard scanning is when you are monitoring for multiple closures, especially in game situations.

I will make a quick digression into why this system works so well for the Atari joystick modification. Basically, the Atari joystick has five switches in it. For ease of programming, the primary joystick is wired into the four main directions of the number keypad, namely, the 2, 4, 6 and 8 keys. The joystick fire button is wired to the 5 key. The joystick is designed so that an angle closure will close two keys simultaneously. For example, the diagonal vector of up and to the right corresponds to the closure of the 6 and 8 keys. With the keyboard scanning technique we can detect the double closure and with a short routine interpret that

intended direction. Listing 2 is a short routine to do that. It returns a value of 0 (no key closure) to 9, corresponding to the positions of the numbers on the number key pad. If you come up with a more efficient routine in BASIC to do this, let me know.

By now you have probably noticed that the system as described will not detect the closure of certain keys, namely, the SHIFT, CONTROL, COMMAND and REPEAT keys. (The only key closure that cannot be detected is the CPU RESET because it has a direct line to the processor.) Those key closures and several others are detectable using the following similar method. Use OUT 7,128 and interrogate INP(1) for a value of 127-SUM (instead of 255-SUM as before). The decrement values are given in the Table. You will notice that the COMMAND key is not listed. Remember that it is actually the simultaneous closure of the SHIFT and CONTROL keys. ■

<u>Key</u>	<u>Decrement</u>
/ (divide)	1
0 (letter O)	2
_ (underline)	4
F15 (Function key)	8
Control	16
Shift	32
Repeat	64

Table of Keys and Decrement Values
Used with the OUT 7,128 Command

```

10 REM KEYBOARD SCANNING PROGRAM
20 PLOT 29,27,11,6,3,12
30 OUT 8,0: REM DISABLE KEYBOARD
40 FOR I=0 TO 15
50 OUT 7,I: REM SCAN EACH LINE INTO KEYBOARD
60 X=INP(1): REM GET A VALUE FROM THE KEYBOARD
70 IF X>255 GOTO 100: REM CHECK FOR ANY KEY CLOSURE
80 NEXT I
90 GOTO 40
100 IF X=X1 AND I=I1 GOTO 80
110 X1=X: I1=I
120 PRINT I,255-X,X: REM PRINT CLOSURE VALUES
130 IF X=251 AND I=15 GOTO 150: REM QUIT IF "Q"
140 GOTO 80
150 OUT 8,255: END: REM REENABLE INTERRUPTS & END

```

Listing 1. Keyboard Scanning Routine.

```

10 OUT 8,0
20 T=0
30 OUT 7,13: IF INP(1)=254 THEN T=T-3: REM DOWN (2)
40 OUT 7,11: IF INP(1)=254 THEN T=T+5: REM LEFT (4)
50 OUT 7,9: IF INP(1)=254 THEN T=T+7: REM RIGHT (6)
60 OUT 7,7: IF INP(1)=254 THEN T=T+9: REM UP (8)
70 IF T>9 THEN T=T-6
80 T=ABS(T)
90 IF T=0 THEN T=1
100 OUT 7,10: IF INP(1)=254 THEN T=6: REM FIRE (5)
110 T=T-1
120 OUT 8,255
130 RETURN

```

Listing 2. Scanning the Number Keypad.

REFERENCES

(1) B.F. Muldowney, "Advanced Keyboard Reading", **FORUM International**, Vol. I, No. 5 (Nov/Dec, 1981), pp. 64-65. And see S. Perrigo, "Joysticks Standard for the CompuColor", **FORUM**, Vol. II, No. 2 (May/June, 1982). (**FORUM** is the publication of the Canadian CompuColor Users' Group. c/o Editor, 21 Dersingham Cres., Thornhill, Ontario, Canada L3T 4P5. I highly recommend that any serious user join this group.)

(2) "CHOMP", a highly recommended program for the joystick, available from ICS, 12117 Comanche Trail, Huntsville, Alabama 35803, and other CCII software dealers for \$29.95.

Assembly Language Programming

by

David B. Suits

PART VI: The Input Routine

In this installment we'll finish up the input routine begun last time, and we'll be taking another look at using the assembler.

The program in the listing is my version of the input routine, fancied up a bit just for purposes of demonstration. It not only accepts input (both upper and lower case, contrary to my proposal in the last issue), but, just for fun, reprints the input in a different color. This verifies that the input string did indeed get stored in memory. In case you still have doubts, after you've assembled and run the program, you can use the Machine Language Debug Package to look at what's in the input buffer (starting at 822EH, if you use my program). After the input is reprinted, the program loops back and asks for another line of input. It will loop forever this way, and you'll have to push CPU RESET to get out of it.

The two columns of numbers at the left of the listing are the hexadecimal numbers which are the machine language code which the assembler generated from my assembly language text. (The rest of the material is the source code which I typed in using the Screen Editor.) The column at the far left is the address (the program starts at 8200H), and the next group of numbers indicate the contents of the memory locations starting at that address. For example, at 8202H the 3EC3H represent two bytes. The first byte, 3EH, is the code for the MVI A instruction, and 0C3H is the byte of immediate data to be moved into A. That is, it is a translation of the assembly language line MVI A,0C3H.

(Remember that the assembler takes all numbers as decimal unless followed by H for hexadecimal or B for binary or O or Q for (ugh!) octal. Moreover, all numbers must begin with a digit; that's why 0C3H must be written the way it is.)

Somewhere before the first statement in the program, I must tell the assembler where the program is to reside in memory. The ORG 8200H tells it that.

Notice that I've probably gone overboard in my use of EQUate statements. For example, instead of using the number 13 throughout as the code for a carriage return, I've used CR and I've told the assembler that CR is to be EQUated with 0DH (=13 decimal). I've used hexadecimal everywhere, merely for the sake of adventure.

After the EQUates comes the label INBUFF. You can tell it's a label of an address because it has a colon after it. The DS means Define Storage, and the number to the right of that indicates how many bytes, namely, LIMIT+1. Since LIMIT is EQUated to 40H (=65 decimal), the assembler knows to reserve, or skip over, 41H bytes and to assign the label INBUFF to the location of the first byte (822EH). When characters are read from the keyboard, they will go here. Next come two bytes reserved for holding the address of (i.e. pointing to) the next available location for a new character in INBUFF. The program really doesn't make any use of that pointer (although it looks like it is going to at 828DH-8290H), but if you decide to add a lot of bells and whistles to the program, you might have need for such a thing. Anyway, I put it in just in case. (Or was it to confuse you? I can't remember now.) As it stands,

the instruction at 8290H is not necessary; the contents of HL always keep track of the next spot in the buffer, and so HL is used as the buffer pointer.

Next come the labels SETUP, PROMPT and REPRNT. These define addresses at which a string of Defined Bytes (sort of like BASIC's DATA) can be found for use with the OSTR routine. If you translate the hex numbers on the left into ASCII, you'll see that the assembler has translated YELLOW into 13H (= 19 decimal), the T in TYPE SOMETHING into 54H (= 84 decimal), and so on. In BASIC if you PLOTTed these numbers (in decimal) you would get the words "TYPE SOMETHING:" in yellow. But you knew all that from our previous work, right?

The very last line of my program is END TEST (followed by a carriage return). The END signifies the end. (Natch!) The TEST tells the assembler that the starting address for the program is at the label TEST. Didn't it know that already? Well, yes; that is, it assumed it. But while the ORG statement told the assembler where the program is to be **loaded**, the TEST at the end tells it where to begin execution. It is assumed that the two are the same unless told otherwise, so technically the TEST is not necessary. But if I had written END INPUT, then the program would load at 8200H, but execution would start at 828DH. But I (almost) always make the start address the same as the load address; you will too. For now, anyway.

Dealing with the Assembler

It would be a tax on anyone's patience to have to plunk in the TEST program one byte at a time. Even using the MLDP would not lessen the burden very much. So it's time to use the assembler.

I use ISC's assembler (**not** the Macro Assembler, please), but if you're using someone else's assembler, the same, or similar, procedures will probably apply. First, write your program using a suitable text editor. (The Screen Editor is perfect. If you have the old Text Editor, throw it away and buy the Screen Editor.) Just type it right in and save it on disk. Now run the assembler. It will sit there blinking stupidly at you. You have several choices. You can quit and go watch Rockford Files re-runs. Or you can put your text disk in the drive and tell the assembler to assemble your program:

>ASM TEST

Or you can have the assembler assemble the program and save the result on disk:

>ASM TEST TO CDO:

(Depending on your disk drive type, you will use CDO: or MDO: or whatever you call your drives. Actually, just the drive number and a colon should be necessary.) If you have two drives, you can take the text off one and save the assembled program onto another. For example:

>ASM 1:TEST TO 0:

When the assembler is assembling your program, the assembled listing (as in Listing 1) will flash by on the screen. Be prepared to hit the ATTN/BREAK key to have it pause, especially if you see a line in red flash by, because that will mean that the assembler has picked out an error and you will want to re-edit the source file accordingly and then give it to the assembler for another try. If you want only the errors, if any, to be displayed, then add /E when you tell the assembler what to do. For example:

>ASM TEST/E

Usually I have the assembler just assemble the program without saving the result, so that I can be assured that the program is error free. When I am satisfied, I run the assembler again and have the object file (i.e., the assembled code) written to disk (and printed on the printer at the same time).

.LDA and .PRG Files

The result which is saved on disk is called a load file (.LDA). It is not quite the final, machine language program which you can run. The .LDA file can be loaded into memory with

FCS>LOA TEST

(The .LDA extension is not necessary, since it is the default type for the LOAD command.) Now you can run the MLDP and test the program. When you're confident that it is the way you want it, leave the MLDP, return to FCS and type:

FCS>SAV TEST.PRG <;version, if you wish> startaddr-endaddr

task. Or how about a machine language patch to a BASIC program? (See Ben's article in this issue.)

For my program, I would type:

FCS>SAV TEST.PRG 8200-82FE

The result is a copy of the memory contents from 8200 to 82FE and is a machine language program which can be run just like any other .PRG file.

Try my input routine. Then try adding variations. (Have the input string reprinted backwards, for example.) Fiddle around. We've come a long way since last year. There are many tricks still to be learned, but you ought by now to have a fundamental grasp of 8080 assembly language programming. What you need now is practice. Why not try converting some simple BASIC programs into A.L.? That's often an easy

In the issues ahead we'll explore tables and vectors, animation, and simple number crunching. And we'll start looking at some of the more esoteric features of the routines in your machine's ROM: disk file handling, talking to your printer, and so on. In the meantime, if you've specific areas of interest you'd like to see covered, please drop me a note.

See you next issue! (Unless your subscription expires.)

```

;TEST -- An almost do-nothing program to get a string of
; characters from the keyboard and then reprint
; that string.

8200 (8200)      ORG 8200H

;First set up jump to CHRINT.

8200 3EC3      TEST:  MVI A,0C3H      ;='JMP'
8202 32C581    STA NOECHO
8205 21F782    LXI H,CHRINT
8208 22C681    SHLD NOECHO+1
820B 3E1F      MVI A,1FH      ;Jump vector.
820D 32DF81    STA KBJVEC

8210 217182    LXI H,SETUP    ;Clear screen, etc.
8213 CDF433    CALL OSTR

;*****
;
;Here is the main program loop.
;
;*****

8216 217382    MAIN:  LXI H,PROMPT
8219 CDF433    CALL OSTR
821C CD8D82    CALL INPUT    ;Input is echoed in green.

;Now reprint the input string in cyan.

821F 218A82    LXI H,REPRNT
8222 CDF433    CALL OSTR
8225 212E82    LXI H,INBUFF
8228 CDF433    CALL OSTR

822B C31682    JMP MAIN      ;Back for another round.

;*****
;
;Equates and storage.
;
;*****

(000D) CR      EQU 0DH      ;Carriage return.
(000A) LF      EQU 0AH      ;Line feed.
(001A) LFTARO  EQU 1AH      ;Left arrow for backspace.
(000B) ERSLIN  EQU 0BH      ;ERASE LINE key.
```

```

(0020) SPACE EQU 20H ;Just a regular space.
(00EF) EOLINE EQU 0EFH ;I'm choosing 239 as end of line byte
; so that the input can be reprinted
; with the OSTR routine.

(00EF) EOM EQU 0EFH ;End of string byte for OSTR.
(0012) GREEN EQU 12H
(0013) YELLOW EQU 13H
(0016) CYAN EQU 16H
(000C) PAGE EQU 0CH ;Erase page.

(007E) MAXKEY EQU 7EH ;Key codes above this are ignored.
(0040) LIMIT EQU 40H ;Max number of characters allowed.

(81FE) KBCCHAR EQU 81FEH ;Holds code of most recent key press.
(81FF) KBFLAG EQU 81FFH ;Keyboard character ready flag.
(81C5) NOECHO EQU 81C5H ;Holds JMP to CHRINT routine.
(81DF) KRJVEC EQU 81DFH ;Holds inout jump vector.
(3392) LO EQU 3392H ;Sends char in A to screen. (V6.78)
(33F4) OSTR EQU 33F4H ;Prints string ending in 239. (V6.78)

;Note: Set LO and OSTR as appropriate for your system. If your
; machine is V8.79 or V9.80, you will use:
;
; LO EQU 17C8H
; OSTR EQU 182AH

822E (0041) INBUFF: DS LIMIT+1 ;Buffer for maximum number of chars
; plus the end of line byte.
826F (0002) INBFPR: DS 2 ;Pointer to next spot in INBUFF.

8271 0CEF SETUP: DB PAGE,EOM ;Very simple.

8273 0D0A0A PROMPT: DB CR,LF,LF
8276 13545950 DB YELLOW,'TYPE SOMETHING:'
827A 4520534F
827E 4D455448
8282 494E473A
8286 0D0A12EF DB CR,LF,GREEN,EOM

828A 0D16EF REPRNT: DB CR,CYAN,EOM

;*****
;
;Here are the subroutines.
;
;*****

;INPUT -- Subroutine to get a string of characters from the
; keyboard until carriage return.
;
; Upper and lower case ASCII characters are allowed,
; as well as the ERASE LINE, CARRIAGE RETURN, and
; BACKSPACE (LEFT ARROW) keys. All other characters
; are ignored.
;
; The input prompt is assumed to have been printed.
;
; CHRINT must be set up.
;
; ENTRY: No register values expected.
;
; EXIT: (C) = count of characters in buffer,
; excluding the end of line byte.
; D,E unchanged.
; (HL) -> end of input buffer.

;Initialize.

828D 212E82 INPUT: LXI H,INBUFF ;Buffer pointer points to first spot
8290 226F82 SHLD INBFPR ; in the input buffer.
8293 0E00 MVI C,0 ;Use register C for count of characters.

8295 CDE882 INPUT1: CALL GTCHA ;Get a character from keyboard.
8298 FE0D CPI CR ;End of inout?
829A CAD182 JZ INPUT4 ;Yes. Go finish up.
829D FE0B CPI ERLIN ;No. ERASE LINE key?
829F CAC882 JZ INPUT3 ;Yes. Go erase the line.
82A2 FE1A CPI LFTAR ;No. Left arrow?
82A4 CAC282 JZ INPUT2 ;Yes. Go backspace.

;At this point we're expecting a normal printable
;character. Anything out of that range must be
;ignored.

82A7 FE20 CPI SPACE ;Is SPACE ) character?
82A9 DA9582 JC INPUT1 ;Yes, so ignore it.
82AC FE7F CPI MAXKEY+1 ;No. Is MAXKEY+1 ) character?

```

```

82AE D29582      JNC INPUT1      ;No. Ignore the character.
                 ;The input character (still in A) is valid. But
                 ;see if there is still room in the buffer for it.

82B1 47          MOV B,A          ;Save character temporarily.
82B2 79          MOV A,C          ;Get count of characters.
82B3 FE40        CPI LIMIT        ;Has it reached its limit?
82B5 CA9582      JZ INPUT1        ;Yes, so ignore the character.

82B8 70          MOV M,B          ;Put character into buffer.
82B9 23          INX H            ;Bump buffer pointer.
82BA 0C          INR C            ;Increment count of characters.
82BB 78          MOV A,B          ;Get character again for echoing.
82BC CD9233      CALL LO          ;Echo character on screen.
82BF C39582      JMP INPUT1        ;Back for more.

                 ;Come here upon finding a left arrow.

82C2 CDD482      INPUT2: CALL BCKSP   ;Delete previous character.
82C5 C39582      JMP INPUT1        ;That was easy!

                 ;Come here with ERASE LINE.

82C8 CDD482      INPUT3: CALL BCKSP   ;Delete a character.
82C9 C2C882      JNZ INPUT3        ;Continue until count of characters = 0.
82CE C39582      JMP INPUT1        ;Then back for new input.

                 ;Come here with carriage return.

82D1 36EF        INPUT4: MVI M,EOLINE ;Put end of line byte into buffer.
82D3 C9          RET              ;Return to calling routine.

;BCKSP -- Subroutine to delete the previous character (if
;
;
;      ENTRY:  (C) = present count of characters.
;              (HL) -> spot in buffer of previous char + 1.
;
;
;      EXIT:   (C) = new count of characters.
;              (HL) adjusted accordingly.
;              (Z) if buffer empty.
;              (NZ) if buffer not empty.
;

82D4 79          BCKSP: MOV A,C          ;Get count of characters.
82D5 B7          ORA A            ;Is it zero?
82D6 CAEA82      JZ BCKSP1        ;Yes, so don't backspace.

                 ;Get rid of the character on the screen.

82D9 3E1A        MVI A,LFTARD
82DB CD9233      CALL LO
82DE 3E20        MVI A,SPACE
82E0 CD9233      CALL LO
82E3 3E1A        MVI A,LFTARD
82E5 CD9233      CALL LO
82E8 2B          DCX H            ;Adjust buffer pointer.
82E9 0D          DCR C            ;Decrement count of characters.
82EA C9          BCKSP1: RET       ;Return with (Z) or (NZ) as appropriate.

;GTCHA -- Simple subroutine to wait until a key is pressed.
;
;
;      ENTRY:  No register values expected.
;
;
;      EXIT:   Character in A.
;              All else unchanged.
;

82EB AF          GTCHA: XRA A
82EC 32FE81      STA KECHAR
82EF 3AFE81      GTCHA1: LDA KECHAR
82F2 B7          ORA A
82F3 CAEF82      JZ GTCHA1
82F6 C9          RET

                 ;Keyboard character interrupt routine.

82F7 F5          CHRINT: PUSH PSW
82F8 AF          XRA A
82F9 32FF81      STA KBFLAG
82FC F1          POP PSW
82FD C9          RET

82FE (8200)      END TEST

```

0 ERRORS

Classified Colorcue Index

by James A. Kavanagh
Gnostech, Inc.
222 S. Easton Road, Suite 15
Glenside, PA 19038

There have been two **Colorcue** indices published: the first appeared in the January, 1980 issue, and the second in the June, 1980 issue. They are arranged by article names only and not by topic or subject.

This index includes all issues of **Colorcue**, including the present issue. It is arranged by subject, although author names are included if the name appeared in two or more articles.

I compiled this index for my own convenience, and so I cannot guarantee its accuracy or completeness. It is intended more for the intermediate programmer and less for the beginner and the non-programmer. There is little reference to elementary topics or to application programs such as games.

The format used is **V.I-P**, where **V** is the volume number, **I** is the issue number, and **P** is the page number. The December 1980/January 1981 **Colorcue** appeared without Volume/Issue numbers. For the purposes of this index, that issue will bear the designation III.7.

Appending (programs) II.3-2, II.5-6
Apple II.5-13, II.5-14
Arrays II.6-6, III.4-7
ASCII codes III.4-14
Assembly Language
 Add A to HL (ADHLA) III.6-12
 AND DE to HL (ANDHD) III.6-12
 ASCII to binary III.3-16
 Binary to ASCII III.3-16
 Binary to 1 hex nibble (B2HEX) III.4-20
 Binary to 2 hex chars (LBYT) III.4-19
 Block move (MOVVDH) III.4-20
 Carriage return/line feed (CRLF) III.4-19
 Communication III.6-4
 Compare, dble precision (CMPHD) III.4-19
 Compatibility 6.78 vs 8.79 III.1-22
 Cursor movement III.2-19
 Debug IV.2-3, IV.2-6
 Divide DE by HL (DIVVDH) III.3-16, III.6-12
 Error codes II.4-2

Exclusive OR DE with HL (XORHD) III.6-12
FCS from assembler III.5-13
Graphics (cursor movement) III.2-19
IBM I/O terminal (USC 1035) III.5-15
Interfacing with Teletype III.1-7
I/O II.7-14, II.8-9, III.1-18 III.2-19,
 III.3-16, III.3-18, III.3-26, III.4-18,
 III.4-26, III.5-13, III.7-3, IV.3-5,
 IV.5-19, IV.6-00
Keyboard input III.7-3, IV.5-19, IV.6-00, IV.6-20
Multiply DE by HL (MULHD) III.6-12
NOR (1's complement) HL (NOTH) III.6-12
OR DE with HL (ORHD) III.6-12
Printer program III.3-18, III.4-26
Programming II.7-14, IV.1-17, IV.2-6, IV.3-19,
 IV.4-19, IV.5-19, IV.6-20
Protected fields (input) IV.3-5
Shift DE left HL times (SHLHD) III.6-12
Shift DE right HL times (SHRHD) III.6-12
Sort IV.2-21
Subtraction, dble precision (SUBHD) III.4-20
2's complement HL (NEGH) III.6-12
Utility routines III.4-18, III.6-12
Wait routines
 20 milliseconds (WATL) III.4-19
 0.5 milliseconds (WATS) III.4-19
Barlow, Ben IV.2-4, IV.3-13, IV.4-19
BASIC
 Chaining (Menu) II.1-6
 Changing directory name II.5-3
 Code line format III.1-16
 Editing (FREDI) II.5-8, II.8-3, II.8-4
 Editor ('THE') IV.3-25, IV.6-3
 Fixing sequence numbers III.1-16
 Format of code line III.1-16
 Formatting numeric fields I.7-18, II.8-6,
 III.4-8
 Input flag, input table I.3-2
 Keyboard input II.2-3, IV.6-17
 Random files (see Random files)
 ReORGing FREDI IV.5-11
 Sequence numbers III.1-16
 String manipulation I.3-4
 Structure of code line III.1-16
 Tokens II.4-4, III.1-16, III.4-16
 Variable listings III.3-24
Bell III.7-16
Binary to ASCII III.3-16
Book reviews III.7-18
Break (generation) III.3-12, III.6-21
CALL function III.7-9
Chaining (Menu) II.1-6
Characters
 Large special II.8-7
 String manipulation I.3-4

Clarke, de France III.1-27, III.2-9, III.2-24
 Clock, real time I.2-2
 Color
 Codes II.3-11, III.2-16
 Misc. II.3-3, III.2-18, III.3-25, III.6-19
 Printer, ink jet III.7-16
 'Comments & Corrections', etc. I.3-6, II.2-6,
 II.3-3, II.4-9, II.5-6, II.7-17
 II.7-19, II.8-15, III.3-26, III.6-22,
 III.7-13, IV.2-3, IV.2-20, IV.4-3
 Communication III.1-14, III.1-18, III.3-26, III.6-4
 Community access bulletin board III.5-22
 Compatibility: 6.78 vs 8.79 III.1-22
 Comp-U-Writer III.7-15, IV.4-15
 COPY, FCS command III.5-21
 COPY (screen display) II.6-13
 Cross reference listing, V6.78 III.7-6
 Curnin, Peter III.3-3, III.4-3
 Cursor movement III.2-19, III.5-4
 Custom character sets III.4-12
 Date routine II.7-17
 Debug IV.2-3, IV.2-6, IV.4-23
 Devlin, Tom IV.1-13, IV.5-5
 Directory
 Changing name II.5-3
 Format II.5-11, III.5-9
 Printout I.8-15
 Supplemental (non-FCS) III.2-4
 Disk dup II.1-4
 Disk recovery III.6-20
 Dotted lines II.1-2
 Dup II.1-4
 Dust cover II.4-2
 Editing
 Basic, FREDI II.5-8, I.8-3 II.8-4, IV.5-11
 Basic, 'THE' IV.3-25, IV.5-9, IV.6-3
 Screen, text III.1-13
 Epson Printer IV.1-14, IV.2-4, IV.2-15, IV.3-13
 Factoring numbers II.5-5
 FCC (on CompuColor) IV.1-3
 FCS
 Copy III.5-21
 From assembler III.5-13
 Input flag, input table I.3-2
 Printing directory II.8-15
 (See also Directory)
 Files
 Creating data files II.2-4
 Random (see Random files)
 Recover from errors III.6-20
 Transferring from other computers III.6-4
 LBA and PRG between disks III.6-21
 Formatting (numeric fields) II.7-18, II.8-6,
 III.4-8
 Fortran III.7-17
 FREDI (BASIC Editor) II.5-8, II.8-3, II.8-4,
 IV.5-11
 Gline II.7-3
 Graphics
 Bar graphs III.7-13
 Circular plots I.2-3
 CRT Mode plotting IV.4-17
 Cursor movement III.2-19, III.5-4
 Dotted lines II.1-2
 Re-entrant plot submodes III.2-11
 Scaling III.7-13
 Sphere IV.2-15
 3-D III.2-6, IV.2-15, IV.4-7
 Green, Bill II.4-2, II.8-3, III.1-4, III.1-6,
 III.4-23, III.5-15, III.6-9, III.6-22
 Handshake III.2-26, III.3-26, III.5-21, IV.1-10
 Hardware mods III.2-26, III.3-12, III.3-26,
 III.4-21, III.5-21, III.7-16,
 IV.1-5, IV.1-13, IV.3-13, IV.5-5,
 IV.5-13
 Hogan, Brian III.1-26, III.2-7
 Hudson, Tom II.5-14, III.4-7, III.5-3
 IBM I/O terminal (USC 1035) III.5-15
 Index (CompuColor) II.3-12
 Input flag, input table I.3-2
 I/O I.2-6, I.2-7, I.3-2, II.2-3, II.7-14, II.8-9,
 III.1-6, III.1-18, III.2-19, III.2-24,
 III.2-25, III.3-12, III.3-16, III.3-18,
 III.3-26, III.4-18, III.4-26, III.5-21,
 III.6-4, IV.1-5, IV.1-14, IV.3-5
 I/O controller (TMS 5501) III.1-6
 Interface, serial to parallel IV.3-16, IV.4.3
 Interfacing with Teletype III.1-6
 Interrupts III.2-25
 Keyboard
 Input I.2-7, III.2-24, III.4-11, III.7-3,
 IV.5-19, IV.6-17, IV.6-20
 Lockout II.2-3, III.2-25
 Keywords (see Tokens)
 Light pen III.3-14
 Line length II.5-15
 Linked lists II.3-5, II.4-5
 Lissajous figures IV.2-18
 Literature II.3-10, III.2-11, III.4-13,
 III.6-7
 Manazir, Richard III.2-19, III.4-17, III.5-21
 Map, system memory III.1-21
 Martin, Dennis II.3-3, III.1-16, III.4-15
 III.5-9
 Matzger, Alan IV.2-21, IV.5-17
 Memory map III.1-21, III.7-6, III.7-10
 Menu (chaining) II.1-6
 MX-80 (see Epson)
 Networking III.1-14
 Newcombe, F. Lee II.2-4, II.8-6
 Noise, power line III.5-16
 Numeric base conversion III.5-11
 Numeric field formatting II.8-6, III.4-8
 OP code table III.6-9
 Pascal triangle III.2-7
 Personal software II.4-7, II.7-12
 Photographing screen II.6-2
 Plot table IV.2-17
 Power line noise III.5-16
 Printer (color ink jet) III.7-16
 Printer (Epson) (see Epson)
 Printer interfacing III.2-25, IV.1-5
 Printer program III.3-18, III.4-26
 Printer, screen to MX80 IV.1-14
 Printing directory II.8-15
 Publishing/selling programs I.2-5, II.6-5
 Raffae, Bernie IV.3-5
 Ram, add on II.4-9, IV.5-5
 Random files II.6-9, II.7-12, IV.5-17
 Random number III.1-26
 Real time clock I.2-2
 Rose, Howard IV.1-27, IV.2-27, IV.4-15
 RS232 I.2-6, II.5-7, III.1-6, III.1-18,
 III.3-26, III.5-15
 Screen character position III.2-13
 Screen display copy II.5-13
 Screen editor III.1-13
 Screen save III.5-3
 Screen to MX80 IV.1-14, IV.2-15, IV.6-9
 Scrolling patch I.1-2
 Selling/publishing programs I.2-5, II.6-5, III.7-3
 Serial port (see RS232)
 Shank, Bill II.7-19, III.1-9
 Smith, Bob V. IV.2-17, IV.4-17
 Software (for CompuColor) I.8-14, II.4-7, II.5-7,
 (see also II.6-4, II.7-6, III.1-13,
 Personal software) III.1-27, III.2-14,
 III.5-18, IV.4-3, IV.4-18
 Sort routine (CALLable) IV.2-21
 Sound board II.5-7
 Source network III.1-14
 Space saving in arrays III.4-7
 Steffy, Myron II.5-13, III.3-24, III.7-9
 String manipulation I.3-4

Stroop phenomenon II.2-2
Suits, David II.2-6, III.2-11, III.3-25, III.6-19
IV.1-3, IV.1-19, IV.2-3, IV.2-6,
IV.3-19, IV.4-19, IV.5-19, IV.6-20
Taylor, Denise III.2-25, III.3-25
Taylor, Trevor III.2-25, III.3-12, III.3-25,
III.4-21, III.6-4, IV.2-18
Teletype (interfacing) III.1-6
Text editor III.1-13
'THE' Editor (see Editing, BASIC)
TMS 5501 (see I/O controller)
Tokens (BASIC) II.4-4, III.1-16, III.4-15
TRS-80 II.5-13, II.5-14, III.2-23, III.6-4
Ungerman, Mike III.2-23, III.6-21

User groups II.2-6, II.5-3, II.7-5, III.1-9,
III.1-10, III.3-15, III.4-13,
III.5-8, III.6-13, III.7-19,
IV.1-17, IV.4-3, IV.5-3
Van Putte, Doug IV.4-7
Variable listings (BASIC) III.3-24
Williams, A.E. II.3-5, II.4-5
Word processing III.7-15, IV.4-15
Ys, lower case IV.1-13, IV.2-3

Cueties

```
PLOT 12:FOR Z=0 TO 1:FOR X=0 TO 255:A=28672+X*4+
128*INT(X/32)+Z*2048:POKE A,X:POKE A+1,128*Z+2:
NEXT:NEXT
```

** About Your Subscription **

Most of our subscribers' subscriptions will come up for renewal after July. (Check your mailing label: the number indicates your last issue number. The June/July issue is issue number 6.)

We need your subscriptions to continue publication.

Since its beginnings in 1978, **Colorcue** has been financed by Intelligent Systems Corporation. That financial assistance will terminate with the June/July issue. Whether we will be able to continue publishing **Colorcue** will depend on whether you renew your subscription. There is a critical number of subscribers, below which **Colorcue** will not have the funds to continue. Let us know of your desire to see **Colorcue** flourish. Send us your renewal now so that we may know in advance where we stand and so that you can help guarantee the continued publication of what we believe to be an outstanding magazine for Intecolor/Compucolor users.

Subscription for one year (six issues) is \$12 in U.S., Canada and Mexico; \$24 elsewhere. Please make check or money order in U.S. funds payable to "Colorcue".

At the same time, why not take this opportunity to let us know what kinds of information you would like to see in **Colorcue** during the coming year? Would you prefer more hardware oriented articles? Tutorials? Programs? Applications? Games? Perhaps you have something specific in mind.

BULK RATE
U.S. POSTAGE
PAID

Rochester, N. Y.
Permit No. 415

Colorcue
Editorial Offices
161 Brookside Dr.
Rochester, NY 14618

Address Correction Requested

An  Publication