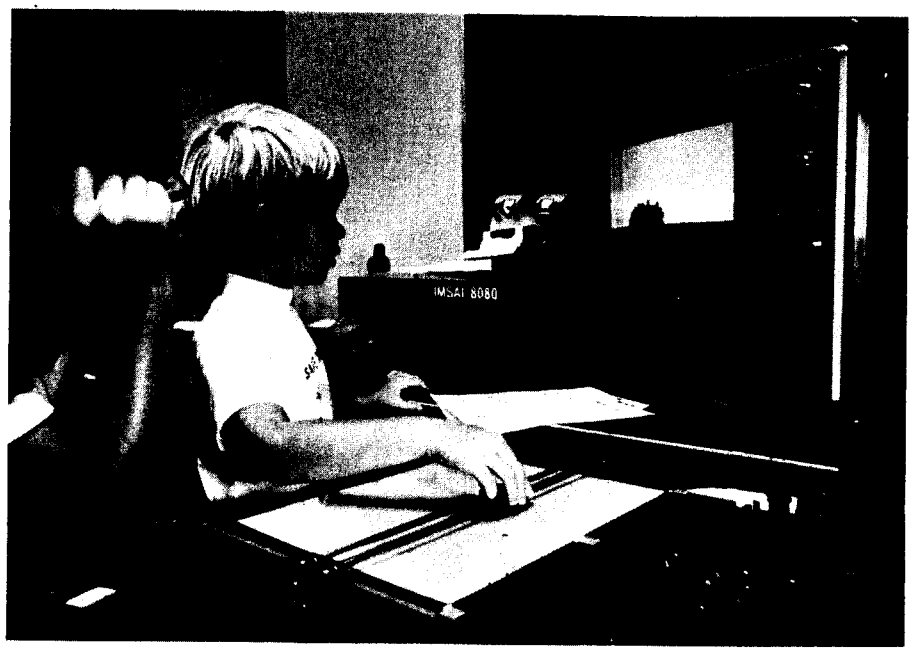
The Cybernetic Crayon



A Low Cost Approach to Human Interaction with Color Graphics

Thomas A Dwyer
Leon Sweer
Soloworks Lab
University of Pittsburgh
Pittsburgh PA 15260

The Cromemco TV Dazzler (described in BYTE No. 10, June 1976, page 6) is one of the most interesting (as well as economical) peripherals available for displaying computer output. It literally puts a picture of what's in your computer's memory on a home color TV set. The simplicity of this idea cuts through all the complexities that expensive color graphics systems (some costing over $100,000) have presented to "ordinary" computer users in the past. The potential applications of low cost color graphics, especially in learning environments of the type we have been developing at Soloworks *[The Soloworks lab is concerned with using computers in education as tools for supporting student creativity. A newsletter describing the project is available from author Dwyer.]*, are almost endless.

At the present time there are two obstacles to using the Dazzler to its full potential. The first is difficulty in programming. Most users find it inhibiting to work at the machine language (or even assembly language) level. There isn't any doubt that color graphics will really take off in educational and home computing when simple user oriented graphic instructions become available in higher level languages like BASIC.

The second problem that needs to be attacked is the lack of human-oriented input devices that allow one to interactively "play" with color graphics. It is of course impressive to see what a clever programmer can do by loading in carefully written machine language graphics demonstrations. But the real future is in making computers responsive to control actions that mirror the "macro" ideas of human imagination and even fantasy. It's the difference between sitting in the back of an airplane admiring how clever your captain is, and moving into the pilot's seat with a chance to do a few lazy eights around the sky yourself.

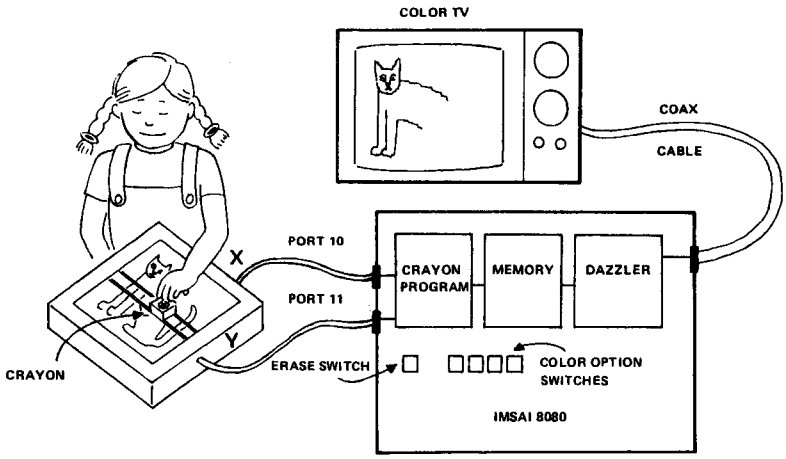


Figure 1: The Cybernetic Crayon System. The XY digitizer position is set by the young artist, and interpreted by the 8080 processor running a program shown in Listing 1. This program uses the position information along with the settings of front panel switches to determine the color value of each position in the picture as it is drawn.

Graphics Software

Some of the people at Dartmouth (Arthur Luehrmann in particular) are working at defining a set of graphic extensions for the language BASIC. This is a good idea, but it's slow work. Getting different groups with different interests to agree on anything is pretty difficult. We'll be following this activity at Soloworks, and may try extending one of the microcomputer versions of BASIC in this direction. In the meantime, we think much attention also needs to be given to *what* people may want to do with graphics, especially the low cost type.

One way to do this is to "imagine" something you'd like to do, and also "imagine" a language for instructing the computer to do this. You can then try to write subroutines in assembly language to implement these macro instructions. True, that's hard work, but eventually the detailed code could be hidden from the user (possibly in BASIC, or possibly in ROM). Then programmers (including young children) could do most of their thinking at the higher level.

Let's illustrate this idea by looking at a first attempt we made along these lines in defining what we call our "cybernetic crayon box." We had lots of ambitious ideas for using the Dazzler, but decided to start very simply. Our thinking was that new features could then be added one at a time in the form of additional subroutines. In other words, the approach we took was to build a total system from what are usually called program "modules." (It's worth noting that this is a good idea for most large programs where clarity is essential. In fact it's the basic idea behind the new rage for what is called "structured programming.")

The Cybernetic Crayon Idea

Let's imagine that we want to make the system of figure 1 possible. The idea is that it would be neat if a child could move some kind of electronic "crayon" around and experiment with drawing colored pictures on a TV screen. In the back of our heads was the thought that it would be even neater if a "big" child (guess who) could drive a space ship around a full color galaxy in some futuristic Star Trek type game.

Let's imagine a computer program to do this using an imaginary high level language. It might look like the following:

1. Turn on the Dazzler and start displaying memory.
2. If desired, erase memory (to get a blank screen).
3. Look at where the crayon is pointed.
4. See what color it is.
5. Decode this information into proper machine language.
6. Now put information into the computer's memory for display.
7. Go back to step 3.

The "hardest" parts are steps 2, 3, 5, and 6. We decided to make step 3 "easy" by using a special piece of hardware, an $80 surplus XY digitizer which was sold by Delta Electronics Co (their ad appeared in the May 1976 issue of BYTE). Steps 2, 5, and 6 were handled by software subroutines that can be thought of as simulating macro instructions. Let's look at each of these four steps in further detail.

Using an XY Digitizer as the Crayon

There are several options for the "crayon." One would be a light pen. Another would be a two axis joystick. The

third possibility is to use what's called an XY digitizer. All of these devices can be expensive, since they usually require special interfacing electronics. The exception to this rule is when the devices produce digital data directly, either through brush type contacts or optical disks that control the light falling on photo electric cells. We chose to use an XY digitizer with brush contacts, partly because it was available as surplus, and partly because the XY frame of reference looked like a good way to help even very young students learn about Cartesian co-ordinate systems (more about this later).

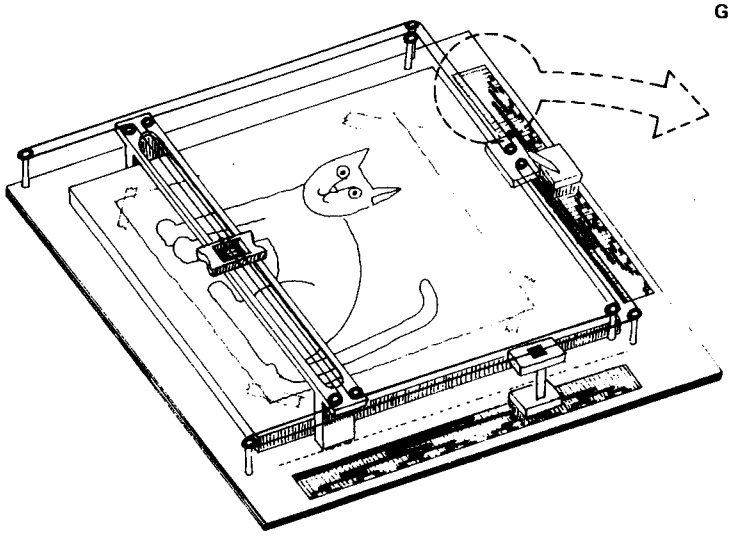
How the Digitizer Works: the Gray Code

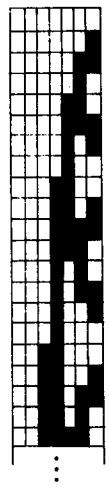
The digitizer is a mechanical device which works something like a plotter in reverse. When the user moves the pointer (what we call the crayon), this moves two sets of contacts to positions corresponding to the X and Y coordinates of the crayon. These brushes slide across metal templates that look something like the pattern in figure 2.

The output for each coordinate is a 7 bit binary number. This means that 128 values (2**7) for each coordinate are possible. Each output can be connected directly to one 8 bit parallel input port of your micro-computer. The way the digitizer is wired, each bit that is enabled by the digitizer (that is, contacted by a brush) is grounded. Therefore it is necessary to complement the number read from the port before further processing. Thus the input pattern (1, 1, 0, 1, 0, 0, 1) becomes (0, 0, 1, 0, 1, 1, 0).

The second trick to using this particular digitizer involves decoding the patterns used on the templates for representing X (and Y) positions. Instead of using a standard incrementing binary code to represent values for X and Y, the digitizer templates use what's called a Gray code. The way that this code represents the X and Y positions between 0 and 127 is shown in figure 2. One may ask, why not use a standard binary code instead of this "strange" version? The reason becomes apparent when one examines each successive number representation. Note that only one bit ever changes between two consecutive positions (or numbers). On the other hand, if the conventional form of binary code were used, many instances would occur in which several bits would have to change at once. (eg: 0111 to 1000 for 7 to 8). This must be avoided because it would be impossible for a low cost mechanical device to succeed in changing all the bits at exactly the same time. Instead, the computer (being as obedient as it is) would read incorrect values as the bits changed. Use of the Gray code solves this problem, but requires that some means be used to translate back into the standard binary code expected by your computer.

The following algorithm will translate the Gray coded numbers into standard binary codes. An example helps to illustrate.





GRAY CODE	BINARY CODE	DECIMAL
0000000	0000000	0
0000001	0000001	1
0000011	0000010	2
0000010	0000011	3
0000110	0000100	4
0000111	0000101	5
0000101	0000110	6
0000100	0000111	7
0001100	0001000	8
0001101	0001001	9
0001111	0001010	10
0001110	0001011	11
0001010	0001100	12
0001011	0001101	13
0001001	0001110	14
0001000	0001111	15
0011000	0010000	16
0011001	0010001	17
0011011	0010010	18
0011010	0010011	19
0011110	0010100	20
.	.	.
.	.	.
.	.	.

Figure 2: Detail of the Cybernetic Crayon's Surplus Digitizer. The rough artwork of an image may be drawn on paper, or the image can be created without such a layout. The Gray code pattern of the inputs can be seen in diagram form and equivalent binary form at the right.

Problem: Change Gray code 0110100 into a standard 7 bit binary code.

Step	Example (in the example, x's indicate bits yet to be determined, initially zero.)
1. Look at the high order (left most) bit first.	Bit is 0
2. Is the exclusive or (XOR) of this bit and all bits to its left 1?	No
3. Put a 0 in the high order bit of a "running total" RT.	RT = 0xxxxxx
4. Look at next bit.	1
5. Is the XOR of this bit and all bits to its left 1?	Yes
6. Then put a 1 bit in RT on right.	RT = 01xxxxx
7. Look at next bit.	1
8. Is the XOR of this bit and all bits to its left 1?	No
9. Put a 0 bit in RT on right.	RT = 010xxxx
10. Look at next bit.	0
11. Continue until finished.	etc.

Using this algorithm, you can see that 0110100 decodes at the test with exclusive OR to (NO, YES, NO, NO, YES, YES, YES), that is, to 0100111 (which is decimal 39). It is fairly easy to write a program for an 8080 or any other processor which does this, and it is shown in listing 1, relative addresses 004D to 0066. It will be part of our final crayon program. The variable C is where RT is kept in binary form, using the trick of putting higher order bits in at the right, and then shifting them (rotating) to the left until 7 bits have been accumulated. The Gray code is in A at the start.

```
; ROUTINE TO DECODE GRAY CODE
DECOD:  MVI     C,0     ;CLEAR REG. C
        MOV     D,C
        MVI     E,7     ;INITIALIZE LOOP
        MOV     B,A     ;PUT GRAY CODE IN B
LOOP:   MOV     A,C
        RLC             ;ROTATE C LEFT
        MOV     C,A
        MOV     A,B
        RLC             ;ROTATE B LEFT
        MOV     B,A
        ANI     80H     ;MASK ALL BUT MSB
        RLC             ;PUT MSB IN LSB
        XRA     D       ;XOR IT WITH ALL HIGHER ORDER BITS
        MOV     D,A     ;REPLACE RESULT
        ADD     C       ;PLACE IN LSB OF C
        MOV     C,A
        DCR     E       ;COUNT DOWN LOOP
        MOV     A,E
        JNZ     LOOP
        MOV     A,C     ;PUT BINARY CODE IN A
        RET
```

A Subroutine for Erasing Memory

If we want to draw a picture in the "memory space" of our computer, the first thing we may want to do is erase the space. This is accomplished simply by writing zeros in each location. The routine in listing 1, relative addresses 009B to 0048, erases memory, beginning at the location specified in the 8080's HL register pair, and erases a number of locations equal to 256 times the number in the A register. (It's a good idea to keep subroutines such as this as general as possible when developing flexible software.)

```
;ROUTINE TO ERASE 256*A BYTES STARTING AT H,L
ERASE:  MVI     D,0     ;CLEAR D,E
        MOV     E,D
        MVI     M,0
NEXT:   INX     H       ;ADVANCE POINTER
        MVI     M,0     ;CLEAR THAT BYTE
        INX     D       ;INCREMENT COUNTER
        CMP     D       ;SEE IF A BYTES WRITTEN
        JNZ     NEXT    ;IF NOT, GO BACK
        RET
        END
```

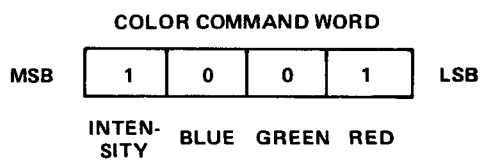
It will be seen later (in the main program) that the erase routine is called by flipping the left most switch of the "programmed input" register on the IMSAI front panel.

A Subroutine for Mapping XY Coordinates Into Memory

Before showing how to translate (or "map") the decoded X and Y values into a memory location to be used as part of our TV picture, it is useful to understand how the Dazzler works. In particular, we want to know something about how it interprets a block of memory and translates it into a color TV picture element.

The Dazzler uses two output ports of the microcomputer. Through these two ports, the computer tells the Dazzler whether to turn itself on, where in memory the picture begins, how many bytes the picture comprises (the choices are 512 or 2048), and what type of picture (color and resolution) should be displayed on the TV. The output to the TV comes directly from the Dazzler.

Figure 3: Color Command Word of the TV Dazzler. This is the layout of each 4 bit nybble in the color display memory region of 2048 bytes.



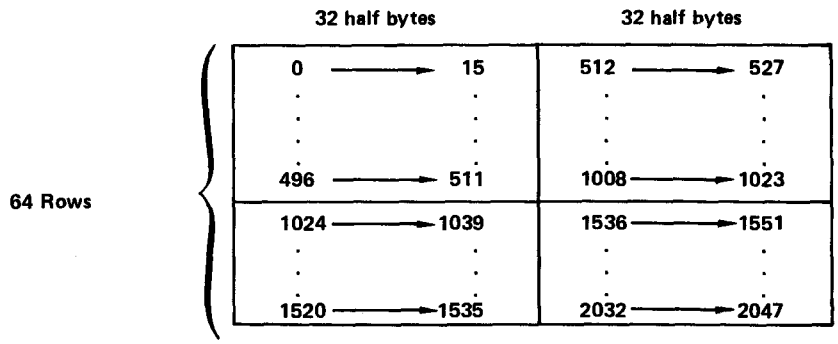


Figure 4: Memory Map of the Dazzler Peripheral. This map correlates the displayed picture to the array of memory bytes. The addresses are indicated here in decimal.

Listing 1: The Cybernetic Crayon Program. This shows the complete assembly listing (relative to address 0). The major subroutines discussed separately in text are the gray code conversion routine DECOD, the memory clearing ERASE, and the address calculation routine TRANS.

```
0000                ;MAIN PROGRAM                                        16
0000                        ORG     0
0000 3E 88                  MVI     A,88H   ;TURN ON DAZZLER
0002 D3 16                  OUT     16H     ; "
0004 3E 30                  MVI     A,30H   ; "
0006 D3 17                  OUT     17H     ; "
0008 31 EF 00               LXI     SP,0EFH ; SET STACK POINTER
000B DB FF          INPUT:  IN      0FFH    ;GET SWITCHES
000D 17                     RAL             ;CHECK FOR MSB
000E D2 19 00               JNC     NOERS   ;IF NOT SET, GO GET POINTS
0011 21 00 10               LXI     H,1000H ;IF SET, LOAD H,L AND A
0014 3E 08                  MVI     A,08H   ;AND CALL ERASE
0016 CD 9B 00               CALL    ERASE
0019 DB 10          NOERS:  IN      10H     ;GET X DATA FROM CODER
001B 2F                     CMA             ;COMPLEMENT X
001C CD 4D 00               CALL    DECOD   ;TRANSLATE X INTO BINARY
001F 67                     MOV     H,A     ;SAVE X
0020 DB 11                  IN      11H     ;GET Y DATA FROM CODER
0022 2F                     CMA             ;COMPLEMENT Y
0023 CD 4D 00               CALL    DECOD   ;TRANSLATE Y INTO BINARY
0026 4F                     MOV     C,A     ;SAVE Y
0027 44                     MOV     B,H     ;PUT X IN B
0028 1E 10                  MVI     E,10H   ;LOAD HIGH ORDER STARTING ADD OF PIC
002A CD 67 00               CALL    TRANS   ;TRANSLATE INTO MEMORY ADDRESS
002D DB FF                  IN      0FFH    ;READ SWITCH REGISTER
002F E6 0F                  ANI     0FH     ;SCREEN OUT OTHER SWITCHES
0031 4F                     MOV     C,A     ;SAVE SWITCH REGISTER
0032 3E 00                  MVI     A,0     ;IF REGISTER D=0
0034 BA                     CMP     D       ; "
0035 CA 45 00               JZ      SNOT    ;THEN GOTO SNOT
0038 79                     MOV     A,C     ;GET SWITCH REGISTER
0039 07                     RLC             ;SHIFT LEFT 4 BITS
003A 07                     RLC
003B 07                     RLC
003C 07                     RLC
003D 4F                     MOV     C,A     ;RETURN TO C
003E 7E                     MOV     A,M     ;GET OLD WORD IN DESIRED LOC
003F E6 0F                  ANI     0FH     ;GET RID OF THIS HALF
0041 B1                     ORA     C       ;PUT IN NEW GROUP
0042 C3 49 00               JMP     STUFF
0045 7E             SNOT:   MOV     A,M     ;GET OLD WD IN DESIRED LOC
0046 E6 F0                  ANI     0F0H    ;GET RID OF HALF
0048 B1                     ORA     C       ;PUT IN NEW GROUP
0049 77             STUFF:  MOV     M,A     ;WRITE NEW WORD IN M
004A C3 0B 00               JMP     INPUT   ;GET ANOTHER POINT
004D                ;
004D                ; ROUTINE TO DECODE GRAY CODE
004D 0E 00          DECOD:  MVI     C,0     ;CLEAR REG. C
004F 51                     MOV     D,C
0050 1E 07                  MVI     E,7     ;INITIALIZE LOOP
0052 47                     MOV     B,A     ;PUT GRAY CODE IN B
0053 79             LOOP:   MOV     A,C
0054 07                     RLC             ;ROTATE C LEFT
0055 4F                     MOV     C,A
0056 78                     MOV     A,B
0057 07                     RLC             ;ROTATE B LEFT
0058 47                     MOV     B,A
```

All these functions are represented on the output ports of the IMSAI 8080 we used, as follows:

Port 16: Bits 0 to 6 contain the most significant 7 bits of the starting picture address. Note that only multiples of 512 are possible.

Port 17: Bit 6 is a resolution multiple of 4 if on, normal if off.

Bit 5 is a picture in 2 K bytes of memory if on, 512 bytes if off.

Bit 4 is a color picture if on, black and white picture if off.

Bits 0 to 3 are intensity and color bits used only in high resolution mode. We will not be concerned with these 4 bits.

It is not necessary to understand all of these options and modes to use the Cybernetic Crayon. We will therefore concentrate on the mode in which 2 K of memory is used, and in which each byte represents two picture elements (small rectangles). In this mode, the picture is composed of 4098 (64 by 64) such elements, the color and intensity of each element being specified by one half of a byte (called a color command word). Each half byte has the meaning to the Dazzler shown in figure 3.

The command word shown in our example is for "high intensity red." Another important thing to understand about the Dazzler is that it is able to read memory on its own, just like the computer, and at the same time that the computer is running its own program. This is called Direct Memory Access (DMA).

Once the computer tells the Dazzler (through ports 16 and 17) where the picture starts, the Dazzler simply takes over and puts the picture right on the screen. Every 1/30th of a second it reads through the entire 2 K of memory and displays it. This arrangement means that the computer can be changing the picture at the same time it is being displayed.

The way in which the Dazzler reads memory can be seen with the illustration of figure 4.

We see that the picture is divided into four quadrants. As the Dazzler reads across a sequence of locations, beginning with the starting location, it displays the least significant half byte first. It is important to remember this when figuring out exactly where in memory a particular color command word should go.

Let's now go back and see if we can piece this information together to enable us to translate our XY coordinates from the digitizer into a memory location that will cor-

respond to the same place in the Dazzler picture. A logical way to go about this is to load a register pair with the beginning address of the picture, and then add to this address an amount derived from the XY coordinates. We could envision the following sequence of events:

1. Load register pair with picture starting location.
2. Ask: "What quadrant are we in?" That is, "are X or Y or both ⩾ 64?" If X ⩾ 64, we are in the right half of the screen, and must add 512 to the starting location. If Y ⩾ 64, we are in the bottom and must add 1024 to the starting location. (If both are true we add both.)
3. Subtract 64 from X or Y if they are greater than 64. This way, every point would be translated into the first quadrant, with X and Y values ranging from 0 to 63.
4. Add the final displacement to the register pair as follows:
   A. Multiply Y by 8; this translates the range from 0-63 to 0-504.
   B. Mask out last four bits to have this address at far left of quadrant (X=0).
   C. Divide X(0-63) by 4 to get 0-15, remembering the right hand carry bit to determine which half word to write (done by shifting right twice).
   D. Add this to the address calculated from Y. Now add total result to the previously calculated quadrant address.

Example: X=47, Y=71. Picture begins at memory location 4096.

1. Load H,L registers with 4096.
2. X < 64 while Y > 64. So add 1024 to H,L. (We are in lower, left quadrant).
3. Subtract 64 from Y, so translated point is X=47, Y=9.
4. A. Multiply Y by 8. 9X8=72 (hexadecimal 48).
   B. Mask out lower 4 bits to get 64 (hexadecimal 48 AND FO gives 40).
   C. Divide X by 4. 47/4 = 11. Carry bit = 1.
   D. Relative location is 64 + 11 = 75. Add this to H,L. 5120 + 75 = 5195

So, the digitizer is pointing at location 5195 in memory. All this calculation would be done in 8080 machine code, resulting in the hexadecimal address value of 144B.

The 8080 subroutine TRANS (see listing 1 addresses 0067 to 009A) translates the XY coordinates stored in the B and C register

Listing 1, continued:

```
0059 E6 80              ANI     80H     ;MASK ALL BUT MSB
005B 07                 RLC             ;PUT MSB IN LSB
005C AA                 XRA     D       ;XOR IT WITH ALL HIGHER ORDER BITS
005D 57                 MOV     D,A     ;REPLACE RESULT
005E 81                 ADD     C       ;PLACE IN LSB OF C
005F 4F                 MOV     C,A
0060 1D                 DCR     E       ;COUNT DOWN LOOP
0061 7B                 MOV     A,E
0062 C2 53 00           JNZ     LOOP
0065 79                 MOV     A,C     ;PUT BINARY CODE IN A
0066 C9                 RET
0067            ;
0067            ;ROUTINE WHICH TRANSLATES B,C INTO ADDRESS IN H,L
0067            ;E MUST CONTAIN PICTURE STARTING ADDRESS (MOST SIG HALF)
0067 78         TRANS:  MOV     A,B     ;PUT X IN AC
0068 E6 40              ANI     40H     ;GET RID OF ALL BUT 64 BIT
006A CA 73 00           JZ      QUADL   ;IF ZERO (X<64) GO AROUND
006D 1C                 INR     E       ;ADD 2 TO E
006E 1C                 INR     E
006F 78                 MOV A,B
0070 DE 40              SBI     40H     ;SUB 64 FROM B
0072 47                 MOV     B,A
0073 79         QUADL:  MOV     A,C     ;GET Y COORD
0074 E6 40              ANI     40H     ;GET RID OF ALL BUT BIT 64
0076 CA 81 00           JZ      QUADU   ;IF ZERO (Y<64) GO AROUND
0079 7B                 MOV     A,E
007A C6 04              ADI     04H     ;INCREMENT THIRD BIT OF E
007C 5F                 MOV     E,A
007D 79                 MOV     A,C     ;SUB 64 FROM C
007E DE 40              SBI     40H
0080 4F                 MOV     C,A
0081 78         QUADU:  MOV     A,B     ;GET X AGAIN
0082 0F                 RRC             ;DIVIDE BY 2
0083 1F                 RAR             ;DIVIDE BY 2 AGAIN
0084 47                 MOV     B,A     ;STORE IN B
0085 3E 00              MVI     A,0     ;CLEAR A
0087 8F                 ADC     A       ;PUT CARRY IN A
0088 57                 MOV     D,A     ;STORE CARRY IN D
0089 78                 MOV     A,B     ;GET X/4 AGAIN
008A E6 0F              ANI     0FH     ;ZERO HIGHER HALF-WORD
008C 47                 MOV     B,A     ;SAVE IN B
008D 79                 MOV     A,C     ;GET Y AGAIN
008E 07                 RLC             ;MULTIPLY BY 8
008F 07                 RLC
0090 17                 RAL
0091 D2 95 00           JNC     LOWER   ;IF NO CARRY, DONT INCREMENT E
0094 1C                 INR     E
0095 E6 F0      LOWER:  ANI     0F0H    ;DUMP LOWER HALF-WORD
0097 80                 ADD     B       ;ADD X DISPLACEMENT
0098 6F                 MOV     L,A     ;LOAD L
0099 63                 MOV     H,E     ;LOAD H
009A C9                 RET
009B            ;
009B            ;ROUTINE TO ERASE 256*A BYTES STARTING AT H,L
009B 16 00      ERASE:  MVI     D,0     ;CLEAR D,E
009D 5A                 MOV     E,D
009E 36 00              MVI     M,0
00A0 23         NEXT:   INX     H       ;ADVANCE POINTER
00A1 36 00              MVI     M,0     ;CLEAR THAT BYTE
00A3 13                 INX     D       ;INCREMENT COUNTER
00A4 BA                 CMP     D       ;SEE IF A BYTES WRITTEN
00A5 C2 A0 00           JNZ     NEXT    ;IF NOT, GO BACK
00A8 C9                 RET
00A9                    END
```

into a Dazzler byte location, and places it in the HL register pair. At the time it is called, register E must contain the most significant half of the starting address of the picture. At the end of the routine, register D is equal to 1 if the most significant half of the byte is to be used, and equal to 0 if the least significant is to be used.

```
;ROUTINE WHICH TRANSLATES B,C INTO ADDRESS IN H,L
;E MUST CONTAIN PICTURE STARTING ADDRESS (MOST SIG HALF)
TRANS:  MOV     A,B     ;PUT X IN AC
        ANI     40H     ;GET RID OF ALL BUT 64 BIT
        JZ      QUADL   ;IF ZERO (X<64) GO AROUND
        INR     E       ;ADD 2 TO E
        INR     E
        MOV A,B
        SBI     40H     ;SUB 64 FROM B
        MOV     B,A
QUADL:  MOV     A,C     ;GET Y COORD
        ANI     40H     ;GET RID OF ALL BUT BIT 64
        JZ      QUADU   ;IF ZERO (Y<64) GO AROUND
```

Continued on page 138

Continued from page 29

```
        MOV     A,E
        ADI     04H     ;INCREMENT THIRD BIT OF E
        MOV     E,A
        MOV     A,C     ;SUB 64 FROM C
        SBI     40H
        MOV     C,A
QUADU:  MOV     A,B     ;GET X AGAIN
        RRC             ;DIVIDE BY 2
        RAR             ;DIVIDE BY 2 AGAIN
        MOV     B,A     ;STORE IN B
        MVI     A,0     ;CLEAR A
        ADC     A       ;PUT CARRY IN A
        MOV     D,A     ;STORE CARRY IN D
        MOV     A,B     ;GET X/4 AGAIN
        ANI     0FH     ;LOOSE HIGHER HALF-WORD
        MOV     B,A     ;SAVE IN B
        MOV     A,C     ;GET Y AGAIN
        RLC             ;MULTIPLY BY 8
        RLC
        RAL
        JNC     LOWER   ;IF NO CARRY, DONT INCREMENT E
        INR     E
LOWER:  ANI     0F0H    ;DUMP LOWER HALF-WORD
        ADD     B       ;ADD X DISPLACEMENT
        MOV     L,A     ;LOAD L
        MOV     H,E     ;LOAD H
        RET
```

Selecting the Color of the Crayon

We decided to let the user select the desired color and intensity by moving the switches for the lower four bits of the "programmed input" register on the IMSAI 8080 front panel. (If the switches are all off, zeros will be written into memory, which gives the "black" color.) We must also know into which half of Dazzler byte location we should put this color information. This is determined by the state of the carry bit which was saved in register D, when X was divided by 4 in the algorithm for address calculation. For example, X = 40 and X = 42 will translate into the same address. But X = 42 will set the carry bit, causing the most significant half of the word to be used, while X = 40 won't, causing the least significant half to be used. A program sequence to accomplish this would be:

1. Read switch registers.
2. Mask out the most significant half word.
3. Store in C.
   If D = 0, get word in location referenced by H,L. Then zero least significant half and "OR" the result with C.
4. If D = 1, rotate C 4 times, get word referenced by H,L. Then zero most significant half, and "OR" the result with C.
5. Move result back to memory.

The above manipulations are taken care of in the main program which we show assembled in listing 1 along with all the subroutines needed.

The Final Program

The final Cybernetic Crayon program consists of a "main program" (which is actually not very long), and three subroutines. The main program handles a few minor tasks (like turning the Dazzler on, and selecting a color), and also calls the subroutines as needed.

For example, in order to get X and Y from the digitizer, it must "read" each value separately from the appropriate input ports (in our program X is read from port 10, and Y from 11). These values must be decoded, and stored in two of the registers of the microcomputer. Thus, once the digitizer is connected to the two parallel ports as described, all that is needed to read and decode each coordinate is an input statement followed by a call to the "DECODING" routine. We can accomplish this task as follows:

1. Input from X port to accumulator and complement.
2. Call decode routine.
3. Move result to B register.
4. Input from Y port to accumulator and complement.
5. Call decode routine.
6. Move result to C register.

This segment is found at addresses 0019 to 0027, with the H register used as a temporary copy of the X value. All our programs were written in 8080 assembly language. For those readers who don't have an assembler, a machine language translation is also shown. This was produced by a cross-assembler written in BASIC-PLUS by Don Simon of Soloworks.

Some Ideas for Extending and Applying the Cybernetic Crayon

Extensions that we are working on include a blinking cursor, superimposing pictures from two digitizers, subpictures, moving pictures, and games that allow human interaction. Some of these may exceed the capabilities of a single processor, which suggests that several processors with shared memory is an idea worth exploring.

The principal application we have in mind is to education, but not in the sense of what is called "CAI" (computer assisted instruction). CAI says that computers should be used to "teach" children. We think that anyone who has used computers knows that it should be the other way around. One of the best ways to learn something is for the student to try to "teach" it to a critical audience. What more critical (but fair) audience can you find than a computer?

Another deep idea about human learning that comes out of letting people play with

computers (as opposed to using computers as Skinnerian teaching machines) is that real computing helps build a rich background of experiences. This is educationally valuable because people with lots of experiences are *much* better audiences for lectures and books. For example, a young child who has played with the Cybernetic Crayon will surely get a lot more out of a math book that explains Cartesian coordinate systems than one who reads the same book cold.

Computers are revolutionary for education, not because they can "automate" teaching, but because they make it possible to undo a serious mistake. Present educational practice is basically upside down. It says to young children "listen to, and memorize all this stuff because some day you'll do great things with it." How much better it would be if we could let kids *do* great things first, and *then* explain how it all worked. The followup would be to show how even better things could be done with new information. The power of computers is that they make such a strategy not only possible, but workable in a way that makes learning the adventure it ought to be. This is why the personal computing movement has much to contribute to the future of education.■

The use of this system can lead to quite practical results for the artist. What called the Crayon System to our attention and resulted in this article was Margot Critchfield's first entry into the BYTE Computer Art Contest, the pastoral scene, **photo 7.** *Here are several of the Art Contest entries which Margot Critchfield has created using the Cybernetic Crayon system described by this article. The comments are based on Margot's notes with direct quotes as indicated.*

Using Computer Graphics as a Medium for Artistic Expression: A Portfolio of Explorations

By

Margot Critchfield
Project Solo
311 Alumni Hall
University of Pittsburgh
Pittsburgh PA 15260

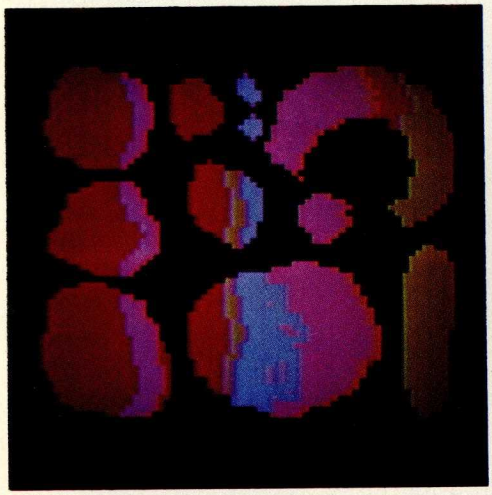


Photo 1: **Fruit Salad.** *The background of warm colors in vertical stripes was created first. Then the shapes were created in Erase mode, after which the stripes were modified to echo the shapes.*

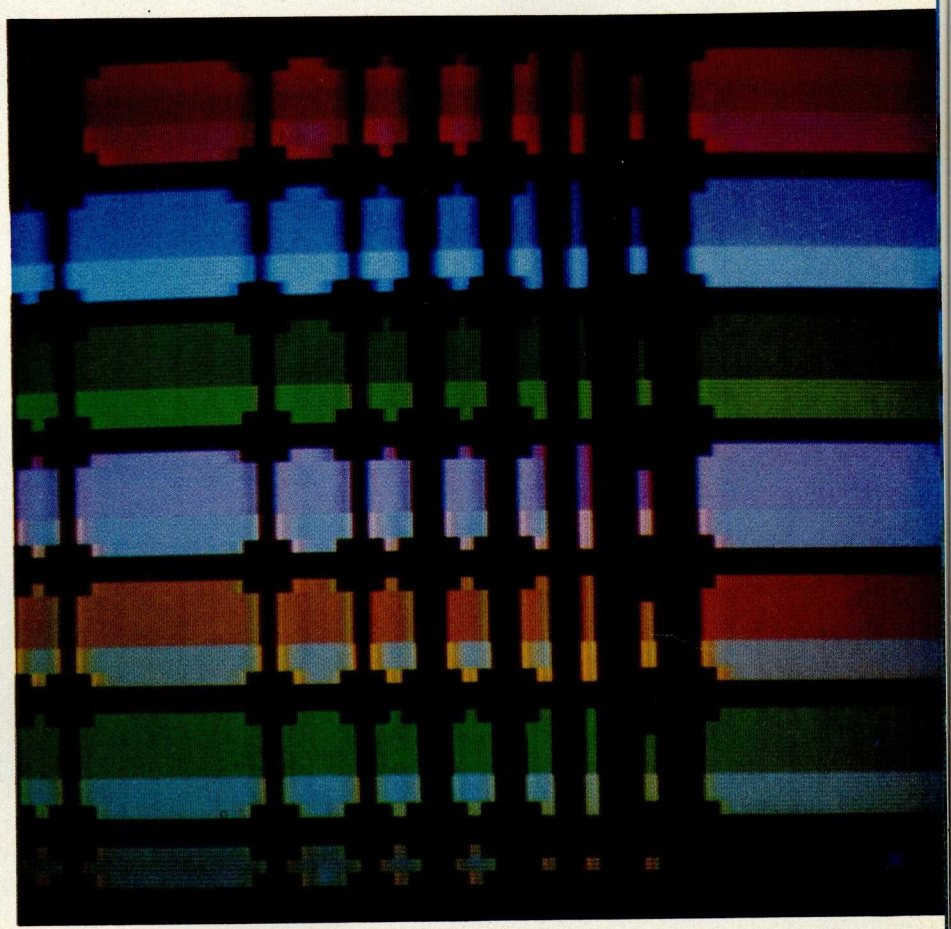


Photo 2: **Framework.** *The background of all possible colors was done first in horizontal stripes. Then the framework and "joints" were done in Erase mode.*

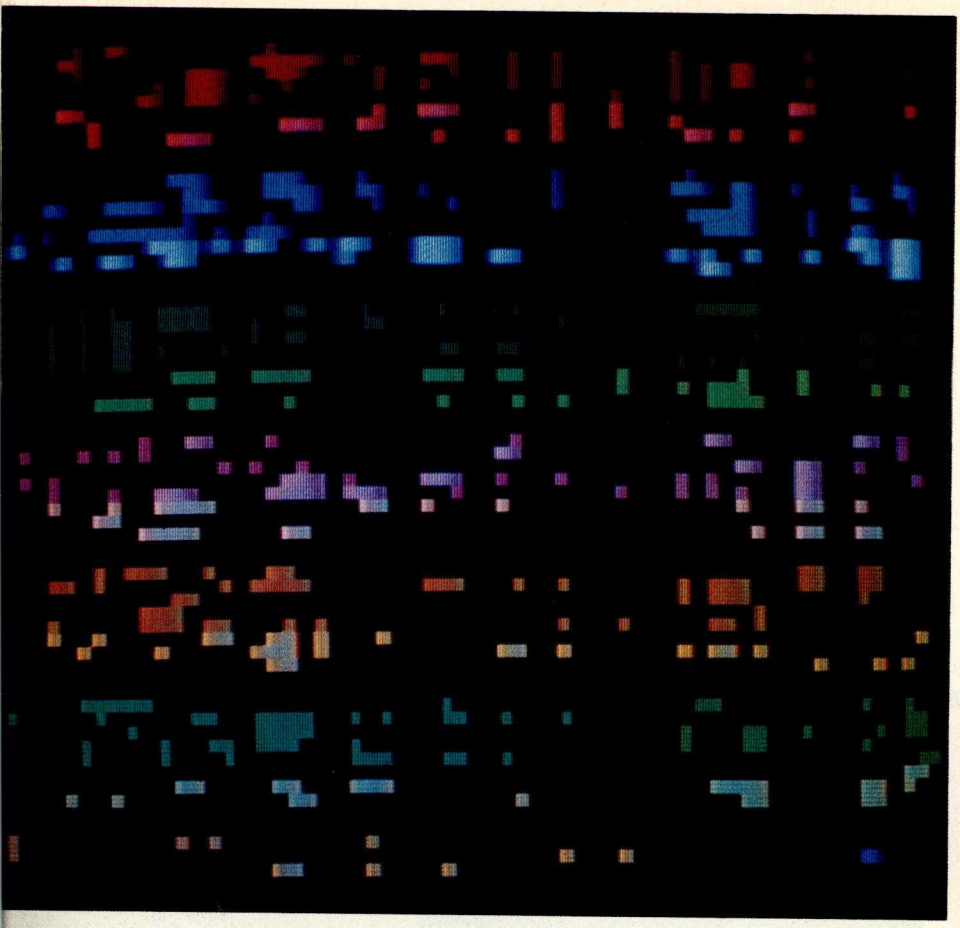
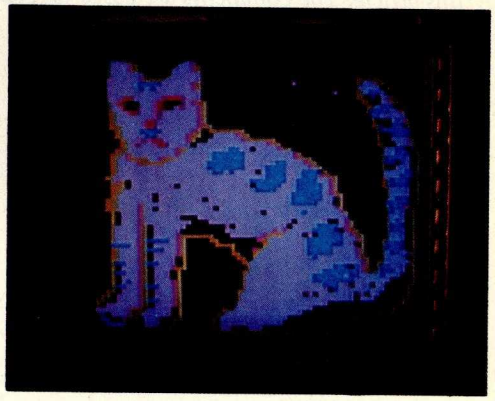


Photo 3: **Modern Stained Glass.** *The framework of photo 2, modified by further erasures.*

Photo 4: **Psychedelic Cat.** *This picture is an attempt to photograph a frame attached to the front of the television set. Margot writes "This is an attempt to work in a more traditional or painterly way with the digitzer. It involves a good deal of patience and much switching back and forth between colors. By this time I had more or less memorized the switches."*



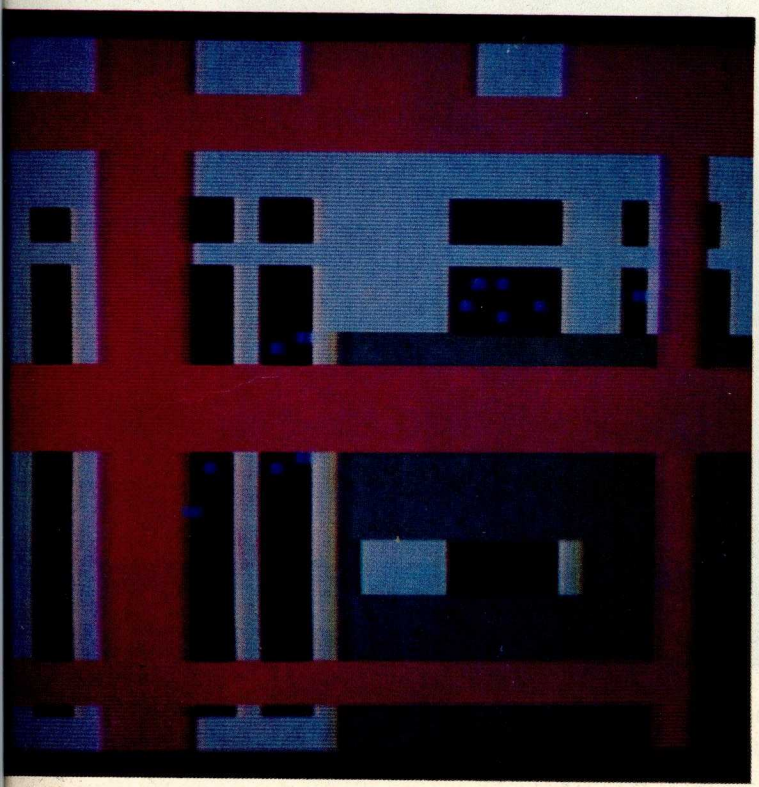


Photo 5: **Windows and Spaces.** *This picture is restricted to horizontal and vertical contours. It achieves a dreamlike quality, with an illusion of overlapping forms.*

Photo 6: **Patriotic Motif.** *"A predrawn map of the US was traced with the digitizer on a blue background then filled in. Initials were done in Erase mode."*

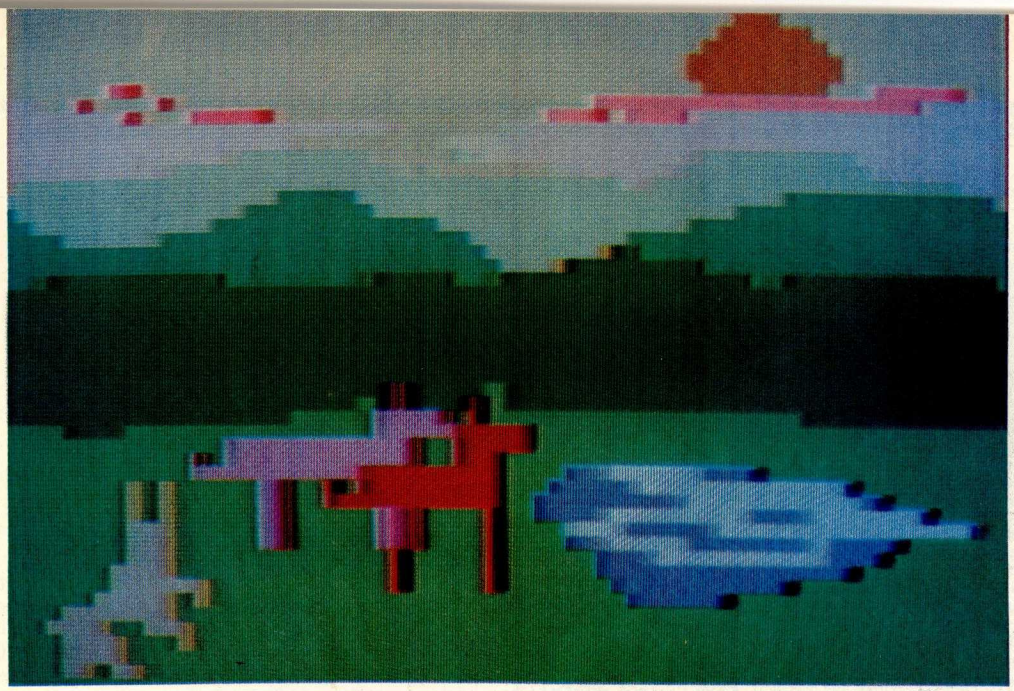


Photo 7: **Pastoral Scene or Ferocious Rabbit Attacking Two Horses at a Pond While the Sun Sinks Slowly Behind the Hills.** *The background of this image was drawn first, then the animals were added. There is a childlike quality (naturally) since a visiting 5 year old drew the horse.*

Photo 8: **Lily Pond.** *"With apologies to Monet. The attempt here is to approximate soft contours, impressionist type color mixtures. Looks good through an out of focus projector lens."*