$2.50

Color Computer News

September/October 1981
Volume 1 Number 3

REMarks

Have you ever seen a publisher panic? It's not a pretty sight. Shortly after the last issue went out the computer bit the dust, a little later the printer decided to follow the act. All is now well (note crossed fingers) and as I write this we are close to being back on schedule. The postal employees had me worried for a while but we're OK again for a while. (Personal note to Canadian friends! Merry Christmas!). While on the subject of the Post Office, did you know that the 9-digit zip code is now real on a voluntary basis. If I understand the letter they sent me correctly we will have a "Bar-Code" return address on the back of this soon. I hope it does help the Bulk Mail problem.

Starting in this issue we have a new column. The first is Comment Corner. Comment Corner is just what the name implies, every issue from now on we will be giving the comments for the Basic ROM. The column is donated by The Micro Works staff and is terrific. There's a tremendious education in the ROMs with the proper guidance and this is just the guidance we all need. For my personal use I disassembled the ROM with their disassembler, pasted the pages into a notebook and am now adding their comments as they send them. Please notice that they have include the low RAM used by Basic. Thanks Bob, Andy and Ann.

Since last issue I received several new pieces of software. There is too much to talk about all of it but I would like to share a few. Chromasette magazine arrived about two weeks ago. The "cover" is slick. The word Chromasette is written in "long-hand" and scrolls all over the screen in color. In addition to the cover there are 5 game programs. My wife and kids liked Blockade the best. The Micro Works sent two adventure games called Black Sanctum and Calixto Island. If you've never played an adventure game these two are a real treat. If you have played adventures before these two are among the best (Scott Adams beware you have a fierce competitor here). TMW also sent an Asteroids game that was excellent. Computer Ware sent their Invaders game. The program has excellent sound, good graphics and the invaders attack fiercely and, best of all DOESN'T require their Power Pak.

Many of you have sent in software for the Sampler series. Please DO NOT send software that you didn't write. I received a tape copy of Radio Shack personal finance the other day from a very well meaning reader. Please understand that giving or selling copyrighted software is very ILLEGAL. I'm not going to get into a discussion of program swapping or dedicate an entire issue to the subject as some magazines have done but I strongly feel that if you copy software, Software houses will produce "protected" software and you will lose the excellent education that comes from examining other people's code. I think this is the worst possible thing that could happen to Color Computer users at this stage of the game. You have to make the moral judgement for yourself, but try to look at it from the standpoint of the guy who programs for a living.

I promised to announce my decision about going monthly this issue. The response has been about equal on both sides and the reasons are about equally good. So my choice is that we will go monthly when we have at least doubled last months' size without sacrificing quality.

We have, as our feature article, a discussion of the DRAW command by Don Inman. Don is best known for his books for the Model I TRS-80 and is soon to release a book about the Color Computer's Extended Basic. Welcome aboard Don.

Mail Call

Dear Bill,

I really love the latest issue of CCN (July/August). The format looks much better -- it's much easier to read and there are far fewer errors.

Your monitor program is worth the price of the entire subscription. It has cleared up a lot of confusion in my mind. It's a very valuable tool. May I make a suggestion? The BASIC command needs a minor adjustment. When creating the "DATA" statements, it puts a comma after every value, including the last one on the line. When the BASIC program is executed, this "trailing comma" seems to generate an extra data value, 0, and pokes it into memory. The result is garbage. One way to fix this is to replace line 8040 as follows:

```
8040 FOR OF=0 TO 9: A$=A$+STR$(PEEK(ADDR+OF)): IF OF<9 THEN A$=A$+","
8045 NEXT OF
```

For Extended Basic users, lines 7010 and 7020 should be replaced with:

```
7010 H$=H1$: GOSUB 10000
7020 DEF USR0=D
```

Apparently, POKE-ing the MSB, LSB of the entry point into locations 275 and 276 is a no-no for the Ext. Basic machine. (Why doesn't the manual say this?) Also the variable HEX$ must have a different name. I used HE$.

I used the monitor, with the above revisions, to POKE in Tom Rosenbaum's Invader program (CCN July/Aug). It works great! Now if I could just understand it.....

Sincerely,
Kathy Goebel
17211 Glastonbury Rd.
Detroit, MI 48219

* Thanks Kathy, poor proofing on my part.

Dear Sirs,

Believe it or not I've got one subscription already, but I want a second one. I've found CCN to be more than a magazine, it's a tool that I write notes in, and underline key points in. With use like that I need a working copy plus a back-up copy.

I would like to commend Computer Plus; not only did they give me the best price on my 16K CC but the service was very prompt. The computer has worked perfectly from the moment it was plugged in.

Sincerely yours
Bobby Joe Harrison
107 Oakhurst
El Dorado, Arkansas 71730

* Did you know that Computer Plus is the largest Authorized Radio Shack Dealer in the country?

Dear Sir,

I like Robert Huxter's "Appending Programs" in V1 #2 of CCN. I have the Extended Basic and had to make a few changes. Address 25 is NOT a 6 when I power up my CC. It is a 30 and changes with each PCLEAR command. PCLEAR 1=12, 2=18, 3=24, 4=30, 5=36, 6=42, 7=48, 8=54. I must do a PEEK (25) first and use that number in step 6 or I lose both programs!

```
0 PRINT PEEK(25)
1 CLOAD" first program "
2 PRINT PEEK(28)
3 POKE 25, PEEK(27)
4 POKE 26, PEEK(28)-2
5 CLOAD" second program "
```

6POKE 25, value from step 0
7 POKE 26,1
If PEEK(28) yields a value <2 then step 3 & 4 are:
3 POKE 25, PEEK(27)-1
4 POKE 26, PEEK(28)+254
As you can see the only change to Robert's statements are steps 0 and 6.
Michael B. Kromeke
6308 Harper Dr. NE
Albuquerque, NM 87109

Dear Bill,
I have at hand a copy of the May/June CCN and it is my firm belief that it does fill a wide gap for people such as myself. The CCN along with the Color Computer does fill many of the needs of the disabled.
Let me go a little further and explain why. All good feelings and aids for living must be intergrated into a pattern of daily life that makes life "fun again".
It seems to me that the cost/performance goes a very long way toward helping this objective. The 6809 MPU and the surrounding chips, make it best bet for the disabled to keep accurate records. A must to be independant.
I, for my part, have written programs to tell "where the money went", others for medicare and charges. When I aquire a print out devise I will be very glad to share them with anyone interested. My typing them out is so full of errors they are useless.
But PLEASE make it clear (in CCN) the CC is no toy but a good data processing devise. So lets have more on data processing and much less on graphics, that, for serious work is almost, useless. Almost.
Donn B. Jones

Dear CCN;

I have enclosed 2 sketches, Fig.1. shows how to install a Micro Works "CBUG" ROM on a R/S "Diagnostics Pak" allowing switching back and forth by resetting the computer & changing the switch on the PAK. If done as Micro Works shows you lose the use of Diagnostics.

In Figure 2 I show how to add 16K to a 16K computer avoiding cutting up the PC board traces by bending pins up on U29 and U6. Connections to U10 were made by using wire wrap wire (#30 wire) and pushing then ends into the U10 socket pins 12 and 35. The lower RAM pins 4 were already connected to U10 Pin 12 on my PC board. I did not use a 33 ohm resistor to the upper RAM pin 4s, perhaps Bob Lentz knows something that I don't.

Radio Shack has a very good service manual which has schematics and a lot of good dope, and of course a couple of errors. Page 40 figure 15 upper right listing change 4 S21 to S20. On sheet 3 page 73 *W of memory chip U21 should connect to *WE not line *RAS. Pretty obvious.

As an avocation I build electronic I/O equipment for the handicapped (paralyzed people). No money! just fun. I just completed a TRS-80C automatic telephone dialer controlled by one "PUFF" switch. Puff 1 starts scan of coded numbers on CRT. Puff 2 dials the chosen number, Puff 3 picks up the receiver, Puff 4 hangs up. No modification of the computer is necessary. Entered this in the Johns Hopkins University contest.
Your Truly,
Joe Sobieski
2277 Menoher Blvd.
Johnstown, PA 15905

"DIAGNOSTICS" & "C BUG"
SWITCH SELECTABLE.
ON ROM PACK.

3PDT MINI SWITCH
MOUNT ABOVE BOARD
THRU 1/4" HOLE ON JOINT OF CASE.

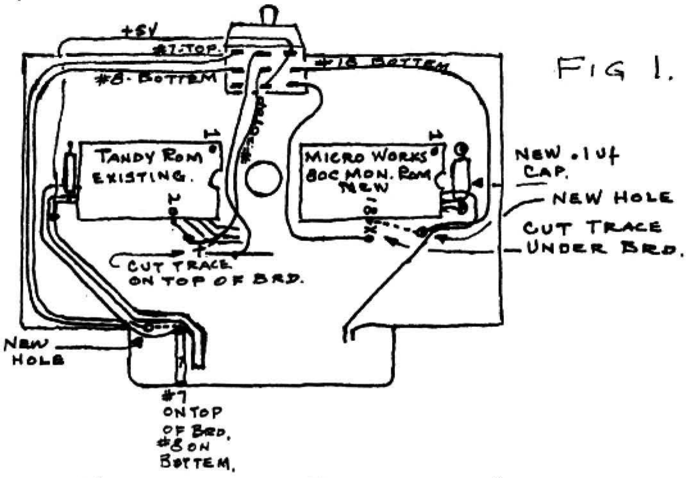
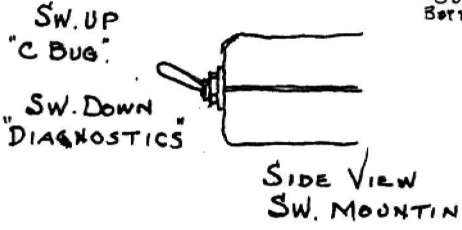


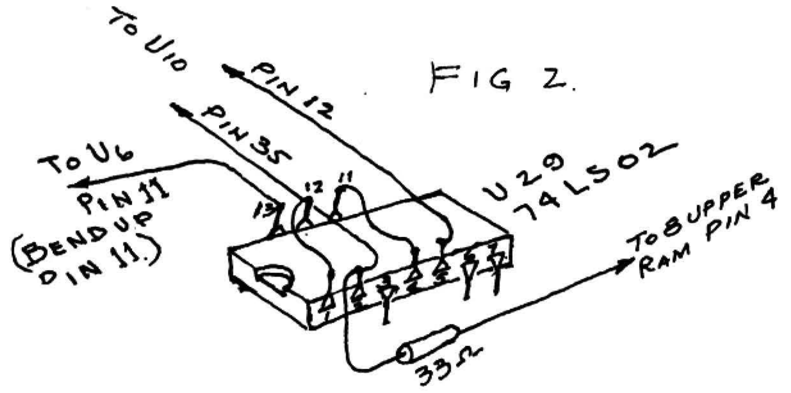
FIG 1.



SIDE VIEW
SW. MOUNTING.

SEE PAGE 7. OF "CBUG"
FOLLOW INSTRUCTIONS EXCEPT
CUT PIN 18 TRACE ON BOTTEM
OF THE BOARD & WIRE IN SW.
AS SHOWN. (# 18 ON "C BUG" SOCKET.)

FIG 2.



```
10 REM HEX/ASCII Dump of memory
20 REM (C) 1981 Gary A. Davis
30 SLOW=65494
40 FAST=SLOW+1
50 CLS
60 INPUT "ENTER START ADDRESS";ST
70 ST=INT(ST/16)*16
80 INPUT "ENTER END ADDRESS  ";EN
90 A$=RIGHT$("000"+HEX$(ST),4)
100 B$=RIGHT$("000"+HEX$(EN),4)
110 PRINT #-2,"Dump starting at";ST;
    "(";A$;")"; and ending at";EN;
    "(";B$;")"
120 PRINT #-2
130 FOR I=ST TO EN STEP 16
140 IF INKEY$<>"" GOTO 50
150 POKE FAST,0
160 AS$=""
170 L$=""
180 FOR J=I TO I+15
190 PJ=PEEK(J)
200 L$=L$+RIGHT$("0"+HEX$(PJ),2)
210 IF J-INT(J/4)*4=3 THEN L$=L$+" "
220 IF PJ<32 OR PJ>127
    THEN PJ=ASC(".")
230 AS$=AS$+CHR$(PJ)
240 NEXT J
250 POKE SLOW,0
260 IF LL$=L$ GOTO 410
270 IF DL=0 GOTO 350
280 IF DL=1 GOTO 320
290 PRINT "---";DL;
    " LINES SAME AS ABOVE ---"
300 PRINT #-2,"        ---";DL;
    "Lines same as above ---"
310 GOTO 350
320 AD$=RIGHT$("000"
    +HEX$(I-16),4)+" - "
330 PRINT AD$;LL$;" *";AS$;"*"
340 PRINT #-2,AD$;LL$;" *";AS$;"*"
350 DL=0
360 LL$=L$
370 AD$=RIGHT$("000"+HEX$(I),4)+" - "
380 PRINT AD$;L$;" *";AS$;"*"
390 PRINT #-2,AD$;L$;" *";AS$;"*"
400 GOTO 420
410 DL=DL+1
420 NEXT I
430 FOR I=1 TO 3
440 PRINT #-2
450 NEXT I
460 END
```

AARDVARK

TRS-80 COLOR   OSI   VIC-64   VIC-20   SINCLAIR   TIMEX

TUBE FRENZY
(by Dave Edson)
This is an almost indescribably fast arcade game. It has fast action, an all concept in play, simple rules, and 63 of difficulty. All machine code, req Joysticks. Another great game by Edson. TRS 80 COLOR ONLY. 16k Joysticks required. $19.95.

QUEST – A NEW IDEA IN ADVENTURE GAMES! Different from all the others. Quest is played on a computer generated map of Alesia. Your job is to gather men and supplies by combat, bargaining, exploration of ruins and temples and outright banditry. When your force is strong enough, you attack the Citadel of Moorlock in a life or death battle to the finish. Playable in 2 to 5 hours, this one is different every time. 16k TRS-80, TRS-80 Color, and Sinclair. 13K VIC-20. $14.95 each.

CATERPILLAR
O.K., the Caterpillar does look a lot like a Centipede. We have spiders, falling fleas, monsters traipsing across the screen, poison mushrooms, and a lot of other familiar stuff. COLOR 80 requires 16k and Joysticks. This is Edson's best game to date. $19.95 for TRS 80 COLOR.

PROGRAMMERS!
SEE YOUR PROGRAM IN THIS SPACE!! Aardvark traditionally pays the highest commissions in the industry and gives programs the widest possible coverage. Quality is the keyword. If your program is good and you want it preserved by the best, send it to Aardvark.

CATCH'EM
(by Dave Edson)
One of our simplest, fastest, funnest machine code arcade games. Raindrops an incredible variety of other things falling down on your head. Use the sticks to Catch'em. It's a BALL! – a flying saucer! – and a Flying YI– an on. TRS 80 COLOR. $19.95.

ESCAPE FROM MARS
(by Rodger Olsen)
This ADVENTURE takes place on the RED PLANET. You'll have to explore a Martian city and deal with possibly hostile aliens to survive this one. A good first adventure.

PYRAMID (by Rodger Olsen)
This is our most challenging ADVENTURE. It is a treasure hunt in a pyramid full of problems. Exciting and tough!

HAUNTED HOUSE (by Bob Anderson)
It's a real adventure – with ghosts and ghouls and goblins and treasures and problems – but it is for kids. Designed for the 8 to 12 year old population and those who haven't tried Adventure before and want to start out real easy.

DERELICT
(by Rodger Olsen & Bob Anderson)
New winner in the toughest adventure from Aardvark sweepstakes. This one takes place on an alien ship that has been deserted for a thousand years – and is still dangerous!

BASIC THAT ZOOOMMS!!
AT LAST AN AFFORDABLE COMPIL
The compiler allows you to write programs in easy BASIC and then matically generates a machine code alent that runs 50 to 150 times faster.

It does have some limitations. It take least 8k of RAM to run the compiler a does only support a subset of BAS about 20 commands including FOR, NE END, GOSUB, GOTO, IF, THEN, RETU END, PRINT, STOP, USR (X), PE POKE, *, /, +, -, >, <, =, VARIA NAMES A-Z, SUBSCRIPTED VARIAB and INTEGER NUMBERS FORM 0-64

TINY COMPILER is written in BASI generates native, relocatable 6502 or 6 code. It comes with a 20-page manual can be modified or augmented by the $24.95 on tape or disk for OSI, TR Color, or VIC.

ADVENTURES!!!
These Adventures are written in BASIC, are full featured, fast action, full plotted adventures that take 30-50 hours to play. (Adventures are interactive fantasies. It's like reading a book except that you are the main character as you give the computer commands like "Look in the Coffin" and "Light the torch.")

Adventures require 16k on TRS80, TRS80 color, and Sinclair. They require 8k on OSI and 13k on Vic-20. Derelict takes 12k on OSI. $14.95 each.

Please specify system on all orders

ALSO FROM AARDVARK – This is only a partial list of what we carry. We have a lot of other games (particularly for the TRS-80 Color and OSI), business programs, blank tapes and disks and hardware. Send $1.00 for our complete catalog.

AARDVARK - 80
2352 S. Commerce, Walled Lake, MI 48088
(313) 669-3110
Phone Orders Accepted 8:00 a.m. to 4:00 p.m. EST. Mon.-Fri.

From Computer Plus to YOU . . .

PLUS *after* **PLUS** *after* **PLUS**

Model 16 128K
1 Drive $4199
2 Drive $4799

Color Computer 16K $305
w/16K Ext. Basic $399
w/32K Ext. Basic $499

Model III 16K $799
Model III 48K
2 Disk & RS232 c $1899

Okidata 82A $425

Color Computer Disk Drive
Drive 0 $470   Drive 1 $315

Smith Corona TPI
Daisy Wheel $599

BUY DIRECT *Here are just a few of our fine offers . . . call TOLL FREE for full information.*

COMPUTERS		**MODEMS**		**DISK DRIVES**	
Model III 4K LEV I	$599	Lynx Direct Connect MI-MIII	235	R.S. Model III 1ST-Drive	67
MODEL III 16K	799	Hayes Smart Modem II	235	Tandon 40 Track MI	28
MODEL III 32K	856.50	Telephone Interface II	169	Color Computer Drive 1	31
*MODEL III 32K	831.50	R.S. Modem I D.C.	130	Color Computer Drive 0	47
MODEL III 48K	914	R.S. Modem II D.C.	210	Primary Hard Disk MI	399
*MODEL III 48K	864	**PRINTERS**		Primary Hard Disk MIII	199
Model III 48K		Daisy Wheel II	1715	**ETC.**	
2 Disk & RS232 c	1899	DWP-410	1335	CCP-81 recorder	5
Color Computer 16K	305	Smith Corona TPI Daisy Wheel	599	C. C Joysticks	2
Color Computer 16K		Epson MX80	599	16K RAM N.E.C. 200 N.S. chips	2
w/extended basic	399	Epson MX80 FT	549	64K Ram Chips	7
Color Computer 32K		Epson MX100	735	Color Computer Flex D.O.S	9
w-extended basic	499	CGP-115	199	Brand Name Software •	
‡Color Computer 32K-64K		DMP-100	318	Send for listing.	
w-extended basic	510	DMP-200	599	R.S. Software 10% off list	
Pocket Computer 2	230	DMP-400	1029		
Model 16 1DR 128K	4199	DMP-500	1559		
Model 16 2DR 128K	4799	Microline 80	325		
DT-1 Data Terminal	599	Microline 82A	425		
		Microline 83A	679		
		Microline 84 Parallel	1029		
		P. C. Plotter Printer	199		

*Computer Plus New Equipment with NEC RAM installed 180 Day Computer Plus Warranty

‡Color Computer 64K requires Disk Band Flex D.O.S.

TOLL FREE
1-800-343-8124

We have the lowest possible Fully Warranted Prices AND a full complement of Radio Shack Software.

Prices subject to change without notice.
Not responsible for typographical errors.
TRS-80 is a registered trademark of Tandy Corp

computer plus

P.O. Box 926
480 King Street
Littleton, MA 01460
617-486-3193

Write for your free catalog

DRAW
by Don Inman

Ideas introduced in this article are expanded in "TRS-80* Color Computer Graphics", a book in preparation for Reston Publishing Company.

One of the most versatile Extended Color BASIC statements used to produce graphics is DRAW. It may be used to draw lines by specifying the starting point, the direction that you want the line to go, and how far you want it to go. This information is all contained in a string that follows the DRAW.

```
DRAW"line-defining string"
      ↑
this string defines the conditions
```

In the Beginning:

In the first part of the string, you move to the origin of the desired line without drawing anything.

```
          DRAW"BM128,96"
                ↑
B for Blank; M for Move to   X,Y coordinates
don't draw   position that   of origin
                    follows
```

If you think of the DRAW statement as commanding the action of an X,Y plotter, the Blank Move (BM) says, "Lift the plotting pen off the paper, move it to the X,Y coordinates that follow, and then lower the pen in preparation for the next command."

Suppose you want to draw upwards from the starting point. You would add to the previous string as follows:

```
      DRAW"128,96;U30
                    ↑
start here  draw up  this many positions
```

The semicolon before the letter U is optional. It helps to visually separate the motion command(s) given. If the above DRAW statement were executed in a program, you would see:

a line going from the center of the screen upward a distance of 30 vertical screen positions

Other Directions:

The computer can DRAW in any of the following directions when the appropriate letter is specified followed by the distance to be drawn.

M for Move to a new position
U for draw Up ↑
L for draw Left ←
D for draw Down ↓
R for draw Right →
E for draw 45 degree angle ↗

DRAW

F	for draw	135 degree angle ↘
G	for draw	225 degree angle ↙
H	for draw	315 degree angle ↖

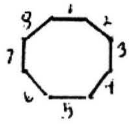
Any of these directions may be combined in a single DRAW statement as demonstrated by the following program.

```
100 'SET UP GRAPHICS SCREEN
110 PMODE 4,1
120 PCLS
130 SCREEN 1,0

200 'DRAW AN OCTAGON
210 DRAW"BM110,60;R40;F40;D40;L40;H40;U40;E40

300 'LOOP HERE TO KEEP PICTURE ON
310 GOTO 310
```

The program draws the sides in the order indicated.

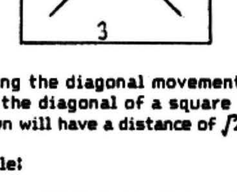


1. R40
2. F40
3. D40
4. G40
5. L40
6. H40
7. U40
8. E40

The Blank Move command may be used at any place within the DRAW statement. Therefore, you can draw a series of unconnected lines also. The statement:

DRAW"BM110,60R20D20L20U20; BM112,62F16; BM112,68E16

should draw the following figure in the order indicated.



1. R20
2. D20
3. L20
4. U20
5. Lift pen and move
6. F16
7. Lift pen and move
8. E16

In using the diagonal movements E, F, G, and H, keep in mind that the distance specified is the diagonal of a square whose sides are the specified distance. The diagonal drawn will have a distance of $\sqrt{2}$ times the specified distance.

Example:

DRAW"BM100,50;E25"

would draw a line from (100,50) to (125,25)

In other words, E25 means draw at 45 degrees to a point 25 units to the right and 25 units up.

Relative Motion

Two types of motion can be created by the motion command (M). We have shown absolute motion in previous examples. This type of command specifies the absolute X,Y coordinates where the drawing is to start.

```
250 DRAW"BM110,60R20D20L20U20"
             ↖
              start here
```

The relative motion command can be used to start a drawing at a specified distance from the last position used in a previous command. Suppose you had just executed line 250 (above).

```
last position ——→(100,60)
drawn
```

You now want to draw another square starting 40 uits to the left and 30 units above the last position plotted by the statement in line 250. The necessary statement would be:

```
      260 DRAW"BM-40,-30;R20D20L20U20"
                 ↑     ↖
      minus sign indicated   minus sign indicated
      start left of last      start up from last
      position               position
```

The picture would look something like this:

40 left, 30 up

original

Other possibilities:

```
BM+40,-30    right 40, up 30
BM-40,30     left 40, down 30
BM+40,30     right 40, down 30
  ↗
```

plus or minus sign is necessary before the X coordinate to indicate to the computer that this is a relative move from the last position.

Stringy DRAW Numbers

The motion commands of a DRAW statement can be readily changed by inserting numbers in the statement in string format.

Example:

DRAW

```
A$ = +30
B$ = -30

DRAW"BM"+A$+;","+B$+"R20E20L20U20
```

The following demonstration program illustrates the changing string values. A square is drawn at clockwise, sequential locations around the edge of the screen.

```
100 'SET SCREEN
110 PMODE 4,1
120 PCLS
130 SCREEN 1,0

200 'DRAW SQUARE AT UPPER LEFT
210 DRAW"BM10,10;R20E20L20U20

300 'DATA FOR DRAW STRING
310 DATA +30,0,7,+0,30,5,-30,0,7,+0,-30,5

400 'MOTION LOOP
410 FOR X = 1 TO 4
420  READ A$,B$,B
430  FOR A = 1 TO B
440   FOR W = 1 TO 200;NEXT W
450   PCLS
460   DRAW"BM"+A$+","+B$+"R20E20L20U20"
470  NEXT A
480 NEXT X

500 'RESTORE DATA AND REPEAT
510 RESTORE: GOTO 410
```

This article merely points out some of the capabilities of the DRAW statement. There are many more. We'll cover them in future issues. Send any questions that you have to the author in care of:

The Dymax Gazettee
P.O. Box 310
Menlo Park, CA 94025

6809 Machine Code
by Bill Sias

Two issues ago we discussed the 6809's opcodes and last issue we wrote a monitor in Basic, so it's time we started programming. For the sake of clarity I will be using SDS80C from the Micro Works. Please do not confuse this as a plug for the product, there are several excellent Assemblers on the market now. I happen to be using this one because it comtains the Editor, Assembler and Monitor all in one package and since it's on a ROM pack it leaves all of my RAM available for Source code. In addition it allows me to Assemble to memory and test before saving the Source code. The first order of business is to explain the commands in SDS80C so that you may compare them with what you are using and note any differences in syntax.

The Editor has several commands that we will not be using. I'll just briefly pass over them and explain the ones will be using.

L insert Lines. Since SDS80C doesn't use line numbers this allows adding lines to the text buffer.
D Delete lines. This allows eliminating lines from the Source buffer.
X eXchange text. The Editor in SDS is a screen Editor and allows several ways of changing the text in the Source buffer.
F Find string. This is an automatic search function.
C Change string. Another form of F that allows automatic search and replace.
A Again. Repeat function, used with F and C.
P Page. Move forward one screen page, -P moves back one page.
T Text. Copy block of text.
M Move. Move block of text.
J Jump. Jump to begin or end of text.
W Write. Write Source buffer to tape.
R Read. Read Source buffer from tape.
@ Assemble. Assemble Source code.
& Recover. Recovers Source code after reset.

The Assembler has the following commands.

L Produce a listing of the assembled Source code.
S Produce a sorted symbol table.
M Assemble to memory.
T Assemble to tape.
! Produce listing in single step format.
3 List to 32 column printer.
4 List to 40 column printer.
8 List to 80 column printer.
= Go to ABUG without assembling.

ABUG allows these commands.

G Execute the Object code.
M Memory examine and modify.
? Evalute expression. (Hex calucator allows using the symbol table and the assembler's expression evaluator)
R Display Registers.
T Transfer block of memory.
J Jump to machine language program.
C Change register values.
S Save machine language program to tape.
L Load machine language program from tape.
U Reset stack.

*   Return to the Editor.

Compare this list with your Assembler and Monitor and make notes where there are differences so that you will be able to make direct conversions in syntax for the utilities that you use.

There are two schools of thought on using ROM calls in machine language programming. Z-80 and 8080 folks will tell you that you shouldn't use ROM calls because it limits the number of machines that your code can be used on. This developed promarily because there is no co-operation between the manufacturers of Intel and Zilog Micros. The folks that produce Motorola based systems haven't had this problem. Because of this there is a lot of compatability between machines, so the practice of using ROM calls is common among 68XX programmers. In our situation we are developing software exclusively for the Color Computer and all of us have the same ROMs so we'll be using ROM calls for things like polling the keyboard and printing characters to the screen. At some point along the way we will be writing software to do exactly those things so that if the ROMs do change our software will still be compatable. One suggestion I will make is that you disassemble the ROMs to a printer and copy the comments from "Comment Corner" to the listing. Not only will this make using the ROM easier but it will help your understanding of how large programs are developed.

To start off let's write a short Basic program and them duplicate the same program in machine code. It will use two ROM calls, Pollcat and Prinit. In Basic it would be:

```
10 A$=INKEY$: IF A$="" THEN 10
20 PRINT A$;: GOTO 10
```

In machine language it could be:

Press L to insert lines.

```
0001 0600 BDA1B1        BEGIN   JSR $A1B1
0002 0603 BDA30A                JSR $A30A
0003 0606 20F8                  BRA BEGIN
```

Press BREAK to leave Insert and press @ M <ENTER> to assemble to memory. When ABUG: appears press G to test the program. Now type anything to see if the code works. Amazing isn't it (well almost)? Let make the program look a little more professional and use some labels. Push RESET to get back to the Editor and lets do it this way.

```
0001 0600               POLCAT  EQU $A1B1
0002 0600               PRINIT  EQU $A30A
0003 0600 BDA1B1        BEGIN   JSR POLCAT
0004 0603 BDA30A                JSR PRINIT
0005 0606 20F8                  BRA BEGIN
```

This works exactly the same but it shows some things that will make programming easier later. The Assembler operation EQU allows you to assign a value to a word so that later in the program you don't have to remember that the ROM routine to read the keyboard is located at $A1B1 just remember that the symbol POLLCAT is the ROM keyboard routine. Another Assembler operation we should look at is RMB. RMB means Reserve Memory Byte(s). Let's change the program to remember how many times a key was pressed. Since we are using a ROM call that waits for a key press we'll just count how many times we return from that routine. Now change your program to read:

```
0001 0600               POLCAT  EQU $A1B1
0002 0600               PRINIT  EQU $A30A
0003 0600               PRESS   RMB 1
0004 0601 BDA1B1        BEGIN   JSR POLCAT
0005 0604 7C0600                INC PRESS
0006 0607 BDA30A                JSR PRINIT
0007 060A 20F5                  BRA BEGIN
```

```
          INC PRESS
          JSR PRINIT
          BRA BEGIN
```

All we have done is to add 1 to memory location PRESS when we return from POLCAT. This won't get us a true count of the number of characters entered so we should make PRESS 0 before we start typing. Add the line:

```
ZERO CLR PRESS
```

between PRESS RMB 1 and BEGIN JSR POLCAT. Now we have a counter for the number of key presses (including spaces). Anyone care to write a Word Processor? Did I see a hand raised in the back? O.K. So far, all we have done is input a character and print it, now we need a way to store the created text so we can print it later and a control code to switch from input to print. First let's store the text in memory. After PRESS RMB 1 let's add TEXT RMB $FF. This will reserve 1K of memory for the text buffer. Add after ZERO CLR PRESS:

```
        LDX #TEXT
        PSHS X
```

And after BEGIN JSR POLCAT add:

```
        PULS X
        STA  ,X+
        PSHS X
```

This will store all the text you enter in the reserve block called TEXT. Now we have to add a control code to inform the "Word Processor" that you are done typing and want to print the text. After BEGIN JSR POLCAT add:

```
PRINT LDX #TEXT
        LDB PRESS
LOOP  LDA ,X+
        PSHS B,X
        JSR $A8BF
        DECB
        BNE LOOP
        LDA #$0D
        JSR $A8BF
        SWI
          .
```

The next thing to do would be to add comments to the code so that it can be refered to later and still understood.This should make the entire program look like:

```
0001 0600                POLCAT EQU $A1B1        BASIC KEYBRD
                         *             SCAN ROUTINE
0002 0600                PRINIT EQU $A30A        PRINT ROUTINE
0003 0600                PRESS  RMB 1            # OF KEYS PRESSED
0004 0601                TEXT   RMB $FF          INPUT BUFFER
0005 0700 7F0600         ZERO   CLR PRESS        START AT 0
0006 0703 8E0601                LDX #TEXT        GET BUFFER ADDR
0007 0706 3410                  PSHS X           SAVE ON STACK
0008 0708 BDA1B1         BEGIN  JSR POLCAT       GET A KEY
0009 070B 8123                  CMPA #'#         CNTRL CODE?
0010 070D 270E                  BEQ PRINT        YES! DO IT.
0011 070F 3510                  PULS X           GET BUFFER ADDR
0012 0711 A780                  STA ,X+          SAVE AND UPDATE
                         *             POINTER
0013 0713 3410                  PSHS X           SAVE NEW POINTER
0014 0715 7C0600                INC PRESS        UPDATE # CHARS
0015 0718 BDA30A                JSR PRINIT       PUT ON SCREEN
0016 071B 20EB                  BRA BEGIN        DO IT AGAIN
0017 071D 8E0601         PRINT  LDX #TEXT        GET BUFFER
0018 0720 F60600                LDB PRESS        THIS MANY
0019 0723 A680           LOOP   LDA ,X+          PUT CHAR IN A
                         *             AND INC. POINTER
```

6809 Machine Code

```
0020 0725 3414              PSHS B,X        SAVE BOTH
                    *            POINTERS
0021 0727 BDA8BF            JSR $A8BF       PRINT#-2
0022 072A 3514              PULS B,X        RESTORE
0023 072C 5A                DECB            CHARS LEFT
0024 072D 26F4              BNE LOOP        IF )0 DO AGAIN
0025 072F 860D              LDA #$0D        GET CR
0026 0731 BDA8BF            JSR $A8BF       PRINT#-2
0027 0734 3F                SWI             GO BACK TO SDS80C
BEGIN  0708  LOOP   0723  POLCAT A1B1  PRESS  0600
PRINIT A30A  PRINT  071D  TEXT   0601  ZERO   0700
```

Type carefully, this is not a word processor actually but more of an expensive electric typewritter that doesn't allow corrections. You can correct the screen but not the text in the TEXT buffer. It would be a simple matter to add it, just add code after CMPA #'# to test for backspace and when it occurs reduce the pointer to the text buffer by one and BRA to BEGIN again. Adding true editing could be done by using any one of a number of techniques, the important thing would be to keep track of where you are in the TEXT buffer.

That's it for this issue and I think it will keep you pretty busy. If you do develop a good word processor from this, please send me a copy and we'll publish it here.

80-U.S.

THE TRS-80 USERS JOURNAL

80-U.S. Journal is a monthly publication for the TRS-80 computer owner. The Journal covers Business, Scientific, Educational, and Recreational areas.

80-U.S. will keep you up to date on new products, software and hardware. Each issue will have listings of programs, reviews, tutorials. 80-U.S. is the complete "How to" Journal for the TRS-80!

If you haven't taken a look at 80-U.S., here is a no-risk opportunity to do it now. Become a trial subscriber now under the protection of a *full* money-back guarantee!

PLEASE enroll me as a trial subscriber to 80-U.S. and bill me just $16. I understand I reserve the right to cancel my subscription any time for any reason and receive a refund for the balance of my subscription. If I should decide to cancel after receiving my first issue I will simply mark my bill cancel and keep the first issue FREE!

Name ______________________
Address ______________________
City ____________ State ______ Zip ____________
☐ I prefer to enclose payment now
Visa MC # ______________________
Exp Date ______
Canada & Mexico $25 per year; other foreign subscriptions $30 surface mail, $52 per year via airmail

80-U.S. Journal
3838 South Warner Street
Tacoma, Washington 98409
(206) 475-2219

Order # CCN

TRS-80 is a Registered Trademark of the Tandy Corp

```
1 REM ROGER LOEWENSTEIN
2 REM 2325 E 29TH ST
3 REM DAVENPORT, IOWA 52803
10 PMODE 4,1:PCLS:SCREEN 1,1
20 A=INT(RND(0)*255)
30 B=INT(RND(0)*191)
40 I=INT(RND(0)*4)+2
50 FOR X=0 TO 253 STEP I
60 FOR S=1 TO 2
70 COLOR S*3,S
80 LINE(X+S,0)-(A,B),PSET
90 LINE(A,B)-(255-X-S,191),PSET
100 NEXT S,X
110 FOR Y=0 TO 190 STEP I
120 FOR S=1 TO 2
130 COLOR 3*S,S
140 LINE(255,Y+S)-(A,B),PSET
150 LINE(A,B)-(0,192-Y-S),PSET
160 NEXT S,Y
170 FOR R=1 TO 1500: NEXT
180 GO TO 10
```

Adventure Game Notes
by Ron Krebs

ADVENTURE GAMES! This is a subject of great interest to many computerists and is reaching cult status with a growing number of regular players. Those of you who have played an adventure game until three in the morning, searching for obscure clues or hidden treasure, know the reasons for their popularity. For you others who have not yet joined this masochistic group, perhaps this article will answer a few questions and enliven your curiosity.

The first thing we should do is define what we are talking about. An adventure game is a story, much like a good novel, arranged in the form of a puzzle. The computer serves as a story teller, basing it's response on input from the operator. Adventure games are intricate and game strategy varies from player to player resulting in a great variety of computer response as each segment of the game is negotiated.

At this point you might ask,"How does the player talk to the computer?" and the answer provides some insight into the mystique of adventure gaming. Each adventure game incorporates a sizeable vocabulary of common english words which the computer will recognize. These words are generally arranged into two groups of verbs and nouns which the player uses to form commands. For example, you might type the verb "open" and the noun "door". If there is a door present the computer will likely respond affirmatively to this command and obligingly respond, "OK, the door is open". Keep in mind that the door in this example may be locked and your computer will admonish you with a response such as "I can't, it's locked." At this time the astute player has gained a clue - there is probably a "key" to be discovered somewhere. A significant new noun should be noted.

So you see, in addition to resolving the puzzle, the player must probe and study to uncover other secrets as well. As each game progresses, the adventurer will discover new locations to visit and objects to investigate, Correct phrasing of commands will result in information and often lengthy responses from the computer. Maybe your flashlight batteries will become exhausted from searching a dark cavern too long. If this occurs at 2:30 in the morning, it will take a lot of willpower to go to bed before you "unlock the door".

From the preceeding it becomes obvious that adventures are pretty sophisticated computer programs and we can reach some initial conclusions. The first conclusion is that adventures take a lot of memory. From this we may correctly assume that any good adventure must be written in machine language and not Basic which would also be too slow for our purposes. After making the above observations we might ask a very important question - "How does one write a machine language adventure?" One appraoch worthy of consideration is the use of a game interpreter.

Let's assume you have just completed several months of work on your new adventure and your friend plays it and enjoys it. You now recognize the need for a second game but cannot bear the thought of starting all over again. This is where the interpreter plays a very important role. The game interpreter is program that does all of the housekeeping involved in game playing. It interprets your commands, searches for play options, formats the computer response and keeps track of player and object lcations. In addition, an interpreter provides the important capability to save the status of a game in progress for resumption later. Because it is written in machine language, the interpreter uses less memory and operates much faster than a Basic program. Now that we have the interpreter at our disposal we need not worry about re-inventing the wheel for each new game. We can describe the interpreter more thoroughly in a future article if reader response indicates an interest in the subject.

The story section of a new game can now be written and this is the hard part. Your first task is to specifically define the object of the game. This is a rule to be observed in all programming and is no less important in games. The story will simply not work until the writer determines what is to be accomplished. The next step is to study all situations, objects and clues which must contribute to the game flow and be interesting. The games should provide balance and challenge but must not present unsolvable problems. A game vocabulary is developed by testing to determine which verbs and nouns are appropriate for a variety of players. As the story portions grows it can be played, de-bugged and modified by use of the interpreter until the final version is resolved.

A few suggestions are in order for those who have not played an adventure. The first suggestion is to maintain a high level of curiosity. Examine all objects and locations thoroughly for clues and information. Secondly, be imaginative in choosing the commands to your computer. If the computer responds,"I don't understand your command", try to come up with a synonymous phrase to achieve the same objective. Taking notes and drawing maps can prove very helpful as you progress through a game. Your enjoyment will grow with each discovery and each obstacle you successfully overcome.

The preceeding comments give you a brief overview of adventure games and their structure but no amount of reading can replace the experience of actual play. The reader of a good novel forms powerful mental images and the same thing occurs playing adventure games. You will experience confusion, frustration, humor, joy and rewarding moments of accomplishment. So, load an adventure game into your computer and see what all the talk has been about. If you happen to choose "Calixto Island Adventure",... don't overlook the bucket!

COMMENT CORNER

The following is a list of comments which could be added to a disassembly listing of the Color Computer ROM. The section given here is called POLCAT, and is the keyboard scanning and debouncing routine. It is called with JSR [$A000] or with JSR $A1C1. It returns with the A register equal to a zero if no key was pressed, or to a key code (ASCII code) if a key has been pressed. All other data registers are saved. It ends with a TST A so that the call to POLCAT can be followed directly by a Branch If Equal instruction to branch if no key was pressed.

There is a bug in POLCAT. There is no hook in POLCAT to the Extended Basic ROM, so the bug is present in Extended Basic also. It is this: If two keys in the same column are pressed simultaneously, they both are entered into the rollover table but only one is processed. For example, press "G" and "O" at the same time, and only "G" will be displayed.

Variables, Areas, and Routines -

Addr	Comments
011A	LOWERCASE FLAG
011B	DEBOUNCE CONSTANT
0152	KEYBOARD ROLLOVER TABLE
A000	ADDRESS OF POLCAT
A1C1	START OF POLCAT
A1C8	GUTS OF POLCAT
A223	HANDLE SHIFT ZERO
A22D	CHECK SHIFT KEY
A238	CHECK KEY COLUMN
A255	DO ASCII $21 THRU $3F
A264	LOOK UP CONTROL KEY
A26E	CONTROL KEY TABLE
A281	LAST BYTE OF POLCAT AREA

Line-by-line Comments -

Addr	Comments
A1C1	SAVE B AND X
A1C3	CALL BULK OF POLCAT
A1C5	SET ZERO FLAG IF NO NEW KEY FOUND
A1C6	RESTORE B AND X, AND RETURN
A1C8	LEAVE 3 BYTES ON STACK
A1CA	KEYBOARD ROLLOVER TABLE
A1CD	INITIALIZE COLUMN COUNTER
A1CF	ZERO BIT IN FIRST COLUMN
A1D1	TO PIA, B SIDE
A1D4	READ COLUMN
A1D6	SAVE DATA
A1D8	FIND KEYS WHICH HAVE MOVED
A1DA	ONLY THOSE WHICH ARE NOW DOWN
A1CD	GET THE NEW KEY PATTERN
A1DE	SAVE FOR NEXT CALL TO POLCAT
A1E0	ANY NEW KEYS
A1E1	GO PROCESS THEM
A1E3	BUMP COLUMN COUNTER
A1E5	SET CARRY BIT
A1E6	NEXT COLUMN
A1E9	LOOP FOR NEXT COLUMN
A1EB	NO NEW KEY; LEAVE WITH A=0
A1ED	GET COLUMN BIT
A1F0	SAVE IT
A1F2	START ROW COUNT AT MINUS 8
A1F4	BUMP BY 8 EACH TIME
A1F6	LOOK FOR THE NEW BIT
A1F7	LOOP TIL THE BIT FOUND
A1F9	ADD ON COLUMN COUNT
A1FB	"@" - GO TO CONTROL
A1FD	BEYOND "Z"?
A1FF	GO TO CONTROL
A201	MAKE ASCII
A203	CHECK SHIFT
A205	MUST BE UPPER CASE
A207	CHECK CASE FLAG
A20A	SKIP IF UPPER CASE ANYWAY
A20C	MAKE LOWERCASE
A20E	SAVE THE CHARACTER
A210	DELAY CONSTANT
A213	DELAY
A216	COLUMN BITS
A218	TO PIA AGAIN
A21B	CHECK COLUMN AGAIN
A21D	SAME AS LAST WE LOOKED?
A21F	GET CHARACTER TO A
A221	IF NOT, FORGET IT !?
A223	IF IT SHIFT ZERO?
A225	IF NOT, RETURN
A227	TOGGLE CASE FLAG
A22A	CLEAR RESULT
A22B	CLEAN STACK & RETURN
A22D	BIT IN SHIFT KEY COLUMN
A22F	TO THE PIA, B SIDE
A232	GET INPUT
A235	GET ONLY THE SHIFT ROW
A237	RETURN
A238	GET PIA INPUT
A23B	MASK JOYSTICK INPUT
A23D	LOOKING AT LAST COLUMN?
A240	SKIP IF NOT
A242	MASK SHIFT KEY
A244	RETURN
A245	FAKE ENTRY FOR "@" SIGN
A247	CONTROL TABLE
A24A	LESS THAN 1/!

A24C	THEN CONTROL	A267	SKIP IF NO SHIFT
A24E	OFFSET TABLE POINTER	A269	PLUS ONE IF SHIFT
A251	BEYOND "?"	A26A	GET ASCII FROM TABLE
A253	THEN CONTROL	A26C	GO DEBOUNCE
A255	CHECK SHIFT	A26E	UP ARROW - UNDERLINE
A257	IF >"+" THEN INVERT SHIFT	A270	DOWN ARROW - "["
A259	SKIP - SHIFT OK	A272	LEFT ARROW
A25B	INVERT SHIFT	A274	RIGHT ARROW - "]"
A25D	TEST SHIFT	A276	SPACE BAR
A25E	OK - GO DEBOUNCE	A278	ZERO
A260	ADD $10 TO MAKE NUMERIC	A27A	ENTER
A262	GO DEBOUNCE	A27C	CLEAR - "\"
A264	TIMES 2 FOR TABLE INDEX	A27E	BREAK
A265	CHECK SHIFT	A280	"@"

QUESTION: I've seen games and other programs which make use of the keyboard in unusual ways. Control keys, typamatic keys, keys which you hold down in order to keep the spacecraft's shields up -- how can these things be done? They can't be done at all on some other computers.

On the Color Computer, the assembly language programmer has direct access to the keys on the keyboard. In only a couple of lines of code a program can tell if any key is down or not.

How does the keyboard work?

There is an output port at location $FF02, and an input port at $FF00. Each key on the keyboard connects one output bit to one input bit. For example, the "H" key connects output bit zero to input bit one. To see if the "H" key is down, write a zero to bit zero of $FF02 and if bit one of $FF00 is a zero, then the key is probably down.

How can I try this out?

With an editor/assembler Rompack such as the SDS80C from The Micro Works, try typing in the "H" code:

```
 LOOP LDA #$FE  ZERO IN BIT 0
      STA $FF02 TO OUTPUT
      LDA $FF00 GET INPUT
      ANDA #2   THAT'S BIT 1
      BNE LOOP  BRANCH IF "H"
      RTS
```

You said before that the key is "probably" down. Why "probably"?

Well, a zero on that input bit could mean a couple of other things. For example, if "I", "P", and "Q" are all down this would provide an alternate (somewhat circuitous) connection between the output bit and the input bit. (Try this on your computer, and see if you get a spurious "H").

What else can cause a false input?

Input bits zero and one are also connected to the right and left joystick buttons. If a joystick button is pressed, then that input bit becomes zero regardless of what is written to $FF02. This is why pressing a joystick button sprays characters onto the screen while in Basic.

What can be done about the joystick buttons?

The simple solution is not to scan the keyboard while the button is down. To see if the button is down, just write all ones to $FF02 and see if all ones come back on $FF00. If not, you might as well wait since you'll just get a spurious reading. If you're calling the ROM routine which scans the keyboard, you can do this check and only call the ROM if no buttons are down.

How can I check the keyboard quickly?

By writing all zeros to $FF02, you can check all the lines at once. If any key is down, then $FF00 will have at least one zero in it. Beware of input bit 7 (the sign bit); it is not connected to the keyboard but to the joystick analog input and depends upon the position of the joysticks.

What is debouncing?

When a key is pressed, it may make and break contact several times before finally closing for good. If a program scans the keyboard fast enough, it may report several keypresses where only one was intended. Owners of the early versions of the TRS80 Model I will be very familiar with this problem. In the Color Computer, after a key press is found, the program waits 10 milliseconds and then looks again to make sure the key is still there. This is called debouncing.

How do I call POLCAT?

POLCAT is the routine in ROM at $A1C1 which scans the keyboard and returns the ASCII code of a key that was pressed. It handles the shift key and uses shift zero to toggle the lowercase flag. When there is no new key, it doesn't wait; it returns a zero in the A register. It does a test on the A register as it leaves, so the call may be followed directly by a Branch If Equal to branch if no key was pressed.

Sounds great. Why not just use POLCAT and never mess with the keyboard input and output ports?

Polcat is fine for many programs, but it doesn't tell you everything. If you're programming a game, for example, you want to keep turning left as long as the key is down, and you have to know whether or not that key is being held down. Or if you want to have typamatic keys (which repeat while held down) in a text editor (as does the Micro Works editor/assembler) you can call POLCAT but then check directly for a key being held down. Also, POLCAT does have bugs.

What bugs are in POLCAT?

First, it doesn't check the joystick buttons. This is normally not a problem, but in some programs which mix joystick and keyboard this could be deadly. Also, it has a real bug in that if it get two keys in one column at the same time, it will ignore one of them. Try typing "G" and "O" at the same time. If this is a problem, then some programming on your own is in order.

Will POLCAT work if Basic isn't running?

Suppose you write an assembly-language program which takes over the machine and generally runs all over Basic's variables. Some routines will then cease working and some won't. POLCAT can be kept alive simply by avoiding locations $011A thru $011C and locations $0152 thru $0159. It uses no other variables; all of its temporaries are put on the stack.

Will the keyboard routines work if I'm using interrupts?

Yes, but it's a little harder. There are two problems. Reading the input port clears the horizontal interrupt, and interrupt routines should restore the output port.

So what's the horizontal interrupt?

This is an interrupt which can be generated each time the TV screen completes a horizontal scan line, which is every 63.5 microseconds. This is very fast even for machine-language programs. If you want to use this interrupt, just beware that a read from $FF00 may clear the interrupt request before it has taken effect, thus causing a cycle to be missed.

And what's this about restoring the output port?

Suppose you are reading the keyboard in an interrupt-driven routine, say every 60th of a second. This will work fine. But what if the program being interrupted is using the same ports? It might be looking at the joystick buttons, for example. But fear not. The output port can be read! The interrupting routine can merely read the output port and save what it found, then do what it wants to (such as call POLCAT), then put back what was there when it came.

All right. So what do I need to know in order to do all of these fun things with the keyboard?

Here it is:

```
I   0      @  A  B  C  D  E  F  G  <--  right joystick button here
N
P   1      H  I  J  K  L  M  N  O  <--  left joystick button here
U
T   2      P  Q  R  S  T  U  V  W

P   3      X  Y  Z  ↑  v  <  >  sp      (↑ v < > are the arrow keys)
O                                       (sp is the space bar)
R   4      0  1  2  3  4  5  6  7
T                                       (en is enter)
    5      8  9  :  ;  ,  -  .  /       (cl is clear)
F                                       (br is break)
F   6      en cl br             sh      (sh is either shift key)
0
0   7  (joystick analog in)

           ↑  ↑  ↑  ↑  ↑  ↑  ↑  ↑
           0  1  2  3  4  5  6  7
             OUTPUT PORT $FF02
```

SPELLIT

by Kathy Goebel

A spelling comprehension aid for kids of all ages.

Especially for Ken

Every week my son, Ken, brings home an assignment consisting of excercises involving 20 or so spelling words. The object of the excercises is to learn the correct spelling, pronunciation and meaning of each word. Since Ken, like many of his peers, is not highly motivated to study by himself, I have often "helped" by drilling him on the weekly word list. After many weeks of this (like, maybe, 2), such devotion to duty tends to become boring, monotonous and repetitive. Boring? Monotonous? Repetitive? This sound like a job for Color Computer!

SPELLIT is a program designed to drill a child in spelling and/or vocabulary. The words and their definitions are input initially via the keyboard. They can then be saved on a cassette for use in a later drill. The computer prompts for each word in the list by printing it's definition. The student then types in the correct word. If the word is misspelled, the computer will give him/her another chance. If, after 3 tries, the word is still not spelled correctly, the computer will print the answer. A final score is calculated based on the percent of answers which were correct on the final try. If the score is under 65%, a sad face is drawn and a sad song is played. The student must then try the whole list again. If the score is 65% or better, a happy face appears and a "happy" song plays.

The critical score, 65, can be modified by changing lines 700, 710 and 730. Currently, up to 30 words per word list are permitted. If more are required, change lines 90 and 140 accordingly.

The songs are, admittedly, not too hot. But I'm sure some of you who are more adept musicians can rectify that little problem.

Our experience has been that this little program is a fun way to study an otherwise un-fun subject. To heighten interest, try changing the pictures and/or songs. And don't tell the kids!

```
10 REM
20 REM    SPELLIT
30 REM
40 REM BY KATHY GOEBEL
50 REM
60 REM    FOR KEN
70 REM
75 CLEAR 500
80 CLS
90 DIM W$(30),D$(30)
100 PRINT @12, "SPELL IT";
110 PRINT:INPUT"NEW WORDS OR OLD";A$
120 IF A$="NEW" THEN GOTO 130 ELSE
IF A$="OLD" THEN GOTO 380
ELSE PRINT "ENTER '
NEW' OR 'OLD'":GOTO 110
130 INPUT "HOW MANY WORDS (<=30)";N
140 IF N>30 THEN PRINT
"TOO MANY WORDS. CHANGE 'DIM'":GOTO 850
150 REM INPUT WORDS & DEFINITIONS
160 CLS
170 FOR I=1 TO N
180 GOSUB 750
190 NEXTI
200 PRINT
:PRINT"WANT TO MAKE ANY CHANGES?"
:PRINT"(ENTER 'NO' OR THE NUMBER OF"
:PRINT"THE WORD TO BE CHANGED)"
210 INPUT A$
220 IF ASC(A$)=78 THEN GOTO 270
230 I=VAL(A$):IF I<1 OR I>N+1
THEN PRINT
"ENTER A NUMBER FROM 1 TO"
;N+1:GOTO 210
240 IF I=N+1 THEN N=N+1:IF N>30
THEN PRINT
"TOO MANY WORDS. CHANGE 'DIM' STMT."
:GOTO 850
250 GOSUB 750
260 GOTO 200
270 PRINT
:INPUT"WANT TO SAVE WORDS ON TAPE"
;A$
280 IF ASC(A$)<>89 THEN 460
290 S$="'RECORD' AND 'PLAY'"
300 GOSUB 790
310 OPEN "O",#-1,NA$
320 PRINT #-1,N
330 FOR I=1 TO N
340 PRINT #-1,W$(I):PRINT #-1,D$(I)
350 NEXT I
360 CLOSE
370 GOTO 460
380 S$="'PLAY'"
390 GOSUB 790
400 OPEN "I",#-1,NA$
```

```
460 REM MAIN SPELL ROUTINE
470 CLS:PRINT @12,"SPELL IT";
480 S=0
490 FOR I=1 TO N
500 F=0
510 PRINT:PRINT "#";I;D$(I);:INPUT W$
520 IF W$()W$(I) THEN F=F+1:IF F(3 THEN
PRINT "WRONG, BUCKO. TRY AGAIN ":GOTO 510
ELSE PRINT "ANSWER IS:";W$(I)
530 IF F=0 THEN S=S+1
540 NEXT I
550 IF F)0 THEN FORJ=1TO400:NEXTJ
560 S=100*S/N
570 FOR I=0 TO 8
580 CLS(I)
590 PRINT @198,"YOUR SCORE IS";
:PRINT USING "###.##";S;:PRINT "%";
600 SOUND 20*(I+1),5
610 NEXT I
620 REM GRAPHICS ROUTINE
630 PMODE 4,1:PCLS
640 SCREEN 1,1
650 CIRCLE (128,96),50
660 CIRCLE (153,77),5
670 CIRCLE (103,77),5
680 CIRCLE (128,96),5
690 IF S)=65 THEN CIRCLE (128,96),
35,,1,.1,.4 ELSE CIRCLE (128,146),
35,,1,.6,.9
700 IF S)=65 THEN A$="T2L402FL8G#G#03L
4C#C#L8FFL4FC#L8C#02L4G#G#L8FF"
:PLAY A$
710 IF S(65 THEN A$="T402L2FL8FL1B-"
:B$="02L3FL4B-03L2D"
:PLAY A$+B$+"P2"+B$+B$+B$
720 CLS
730 IF S(65 THEN PRINT @224,
"I THINK YOU'D BETTER DO IT OVER!";
:FORJ=1TO800:NEXT
J:GOTO 460
740 END
750 REM INPUT SUBROUTINE
760 PRINT "WORD #";I;:INPUT W$(I)
770 PRINT "DEFN #";I;
:LINE INPUT "? ";D$(I)
780 RETURN
790 REM TAPE I/O
800 CLS:INPUT "NAME OF FILE=";NA$
810 PRINT: PRINT "POSITION TAPE AND "
820 PRINT "PRESS "+S$
830 PRINT
:INPUT "HIT 'ENTER' WHEN READY";A$
840 RETURN
850 END
```

```
5 GOTO 50
10 SET(H1,V1,0)
20 SET(H1,V1-1,0)
30 SET(H1-1,V1,0)
40 SET(H1+1,V1,0)
45 RETURN
50 RS=0:LS=0
55 CLS(0)
60 PRINT@480,"          ";LS;"
        ";RS
70 V1=27:H1=15:H2=47
80 GOSUB 10
90 H1=H2
100 GOSUB 10
460 V=RND(20)+3
470 A=RND(2)
480 IF A=1 THEN 500
490 GOTO 730
500 FOR H=1 TO 63 STEP 2
510 SET(H,V,0)
520 GOSUB 560
530 RESET(H,V)
540 NEXT H
545 GOSUB 560
550 GOTO 460
560 S=PEEK(65280)
```

continued on page 33

For Your Color Computer

MASTER CONTROL

Copyright ©1981 Soft Sector Marketing, Inc. – Written by A. Schwartz

Requires 16-32K

1. 50 preprogrammed command keys. Standard and Extended command.
2. Direct control of motor, trace and audio from keyboard.
3. Automatic line numbering.
4. Programmable Custom Key.
5. Direct Run Button.
6. Keyboard overlay for easy program use.
7. Easy entry of entire commands into computer.

Load Master Control into your machine then either type in a BASIC program or load one in from tape to edit. Cuts programming time by 50% or more $24.95

For the Radio Shack Color Computer*

COLOR BONANZA

Some 4K, Some 16K, Some 32K

Written by L.R.T.

50 PROGRAMS In One Package

Distributed by

SSM SOFT SECTOR MARKETING, INCORPORATED

List $49.95

[illegible]

SSM SOFT SECTOR MARKETING, INCORPORATED

6250 Middlebelt • Garden City, Michigan 48135

Order Line 800-521-6504

Michigan Orders & Questions 313-425-4020

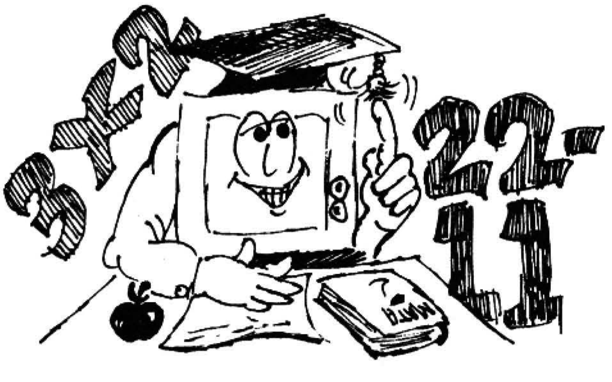
Kid's Page

This issue we have a few donated programs from the Kids. I'm surprised that most of you guys are writing educational programs.

```
1 'KATHLEEN O'BRYAN
2 'AGE 10
3 '2738 N. BENNETT
4 'TACOMA, WA 98407
5 'RANDOM COLORS
10 CLS
20 LET X=0
30 LET Y=0
40 LET Z=RND(9)-1
50 SET(X,Y,Z)
70 X=X+1
80 Y=Y+1
90 IF Y=32 THEN Y=1
100 IF X=62 THEN X=1
110 IF Z=8 THEN Z=0
120 GOTO 40
130 END
```

```
1 'MR. DONALD WHITE
2 '44 DOW COURT
3 'FAIRFIELD, OH 45014
10 X=RND(1000000):Y=RND(1000000)
15 N=0
20 PRINT" "X"+"Y"=":INPUT A
30 IF A=X+Y THEN 10
40 N=N+1
50 IF N=2 THEN 100
60 SOUND 101,10
: PRINT"WRONG TRY AGAIN": GOTO 20
100 SOUND 101,20
: PRINT"WRONG"X"+"Y"="X+Y
110 PRINT"SORRY TRY AGAIN"
: GOTO 10
```

```
5 REM BETH NORMAN AGE 11
10 CLS: CLEAR 500
15 PRINT"WELCOME TO ADDQUIZ";:
INPUT"DO YOU NEED INSTRUCTIONS";
A$: IF LEFT$(A$,1)="N" THEN 20
ELSE 17
17 PRINT"I WILL GIVE YOU A
PROBLEM, TYPE IN THE ANSWER AND
THEN PRESS <ENTER>. I WILL TELL
YOU IF YOU ARE RIGHT OR WRONG.
IF YOU ARE WRONG, I'LL TELL YOU
WHAT THE RIGHT ANSWER IS, IF NOT
I'LL JUST GO ON."
20 INPUT"HOW MANY QUESTIONS DO
YOU WANT;B:C=0
30 X=RND(20):Y=RND(20)
35 PRINT"WHAT IS"X"+"Y"?":
INPUT D: IF D=X+Y THEN 36
ELSE 37
36 PRINT"CORRECT!":C=C+1:
IF C<B THEN 30 ELSE 40
37 PRINT"WRONG. THE CORRECT
ANSWER IS"X+Y".": C=C+1: IF C<B
THEN 30 ELSE 40
40 PRINT"CONGRATULATIONS! YOU
REALLY FINISHED THEM ALL! DO
YOU WANT TO DO IT AGAIN?":
INPUT E$: IF LEFT$(E$,1)="N"
THEN 41 ELSE 10
41 PRINT"SCAREDY-CAT!": END
```



A Basic Bug

by C. J. Roslund

One of the first tasks the users group I am working with undertook after we got our Color Computers was to disassemble and analyze the BASIC interpreter in them. Overall, we are very impressed with the Color Computer and feel it is one of the most powerful and versatile home computers available. This article provides a brief description and should be of use to all Extended Basic Color Computer users. It describes one of the BUGS in Extended Basic and provides a means of working around the BUG.

The PCLEAR command is used to reserve a specified number of graphic (1.5K) pages of RAM. In addition to reserving graphic pages of RAM, PCLEAR does the following tasks:

1. Moves the entire Basic program up or down in memory so that it will begin immediately following the last page of graphic RAM reserved.

2. Does a RESTORE to move the DATA pointer to the beginning of the relocated Basic program.

3. Clears all variable tables (Simple, Array & String) by initializing all variable table pointers with respect to the relocated Basic program.

4. Voids the processor hardware stack pointer (S-Register). This means a PCLEAR cannot be called by a GOSUB.

After the above tasks are complete, execution of the Basic program continues with one major flaw! A pointer located at $A6,$A7 ($ indicates a HEX value) did not get moved with the rest of the Basic program. This pointer is used by the Basic interpreter to locate the next byte in the Basic program to execute. The failure to move this pointer causes the Basic interpreter to continue to execute the original Basic program (or what's left of it), not the relocated Basic program. This will most likely lead to a SYNTAX ERROR when the pointer ($A6,$A7) runs into an area where the relocated Basic program wrote over the original Basic program or the original Basic program area is changed by some other means (such as a PCLS). A sample program that causes this to occur is listed at the end of this article.

Luckily there are a few other Basic commands that will adjust the $A6,$A7 pointer for us: RUN and GOTO.

After the program crashes due to this BUG, entering RUN again will initialize the $A6,$A7 pointer to the beginning of the relocated Basic program and all is well from there on. This is not what I would call an elegant solution (let the program crash and then RUN it again).

GOTO provides us with the best solution. GOTO has two modes of operation. One for going to forward referencing lines and one for going to reverse referencing lines.

For forward referencing lines, the GOTO resets the $A6,$A7 pointer to point to the line called by the GOTO. This doesn't do us any good.

For reverse referencing lines, the GOTO resets the $A6,$A7 pointer to the beginning of the relocated Basic program (SUCCESS!!!) and begins it's search for the called line from there.

This gives us a solution. The PCLEAR must be followed by a reverse referencing GOTO (eg. 20 PCLEAR1:GOTO10).

One more thing to be careful of is not to allow the relocated Basic program to write over the section of your program with the PCLEAR and GOTO in it. A general rule is if you are PCLEAR'ing more graphic pages (eg. PCLEAR8) put the PCLEAR & GOTO at the beginning of your program. If you are PCLEAR'ing fewer graphic pages (eg. PCLEAR1) put the PCLEAR & GOTO at the end.

SAMPLE PROGRAM FOR PCLEAR BUG

A Basic Bug

Step 1: In direct mode enter PCLEAR4

Step 2: Enter following program

```
10 PCLEAR8
20 PMOED3,2
30 SCREEN1,1
40 PCLS6
50 GOTO50
```

Step 3: RUN the program. What you should see is a nice light blue screen, but "SURPRISE!!!" you've got a SYNTAX ERROR.

RUN the program again (without Step 1) and it works. The second RUN initialized the $A6,$A7 pointer to the beginning of the relocated Basic program.

FIX FOR SAMPLE PROGRAM

Add the following lines:

```
 5 GOTO10
 7 GOTO20
15 GOTO7:'REVERSE REFERENCING GOTO
```

Now you may RUN the program with or without step 1 and it works. These extra GOTO's are a bit bothersome but they do provide a fix for this BUG.

continued from page 28

```
570 IF S=126 OR S=254 THEN 580 E
LSE 630
580 FOR M=25 TO V STEP-1
590 SET(15,M,0)
600 IF H=15 AND M=V THEN 790
610 RESET(15,M)
620 NEXT M
630 S1=PEEK(65280)
640 IF S1=125 OR S1=253 THEN 650
 ELSE 700
650 FOR M1=25 TO V STEP-1
660 SET(47,M1,0)
670 IF H=47 AND M1=V THEN 840
680 RESET(47,M1)
690 NEXT M1
700 REM
720 RETURN
730 FOR H=63 TO 1 STEP-2
740 SET(H,V,0)
750 GOSUB 560
760 RESET(H,V)
770 NEXT H
780 GOTO 460
790 LS=LS+1
800 SOUND 220,1:SOUND 100,1
810 IF LS=12 THEN 900
820 RESET(H,V)
830 GOTO 55
840 RS=RS+1
850 SOUND 200,1:SOUND 175,1
860 IF RS=12 THEN 900
870 GOTO 820
900 SOUND 89,3
910 SOUND 108,3
920 SOUND 125,3
930 SOUND 147,3
940 FOR D=1 TO 100:NEXT D
950 SOUND 125,3
960 SOUND 147,3
970 CLS(0)
980 PRINT@480,"        ";LS;"
        ";RS
990 INPUT "TRY AGAIN?";A$
1000 IF A$="Y" THEN 5 ELSE END
```

CONVERGENCE
by Warren White

Now we all know that the color computer is about the neatest little computer to hit the market. High power processor, a good BASIC interpreter, and super graphics. Only one little problem remains (well, probably more than one but...). That !#"$%%$#& TV set just doesn't hold up to the hi-res modes and the text is really cleaner on the old 12" black & white set. You say that the letters and lines smear on the edges of the old TV? Ghost images on the screen? (to the tune of Ghost Riders in the Sky)? Well what can be done about it? I've already diddled the controls on the set to the point that the family can't watch soaps without gagging at the purple faces.

Seriously, one of the weakest links in the computer is the use of a standard TV as the monitor. At NCC this Spring I saw many color displays which were startling for their clarity. The secret was the use of color monitors which are specifically designed to interface to computers. Lacking the $400 to $2000 to purchase one of these beauties, I have been investigating ways of maximizing the performance of a standard color TV.

A little theory may help before we proceed. A color TV generates the picture we see on the screen by sweeping three streams of electrons across the face of the screen. These electrons pass through holes in a metal mask just behind the face of the screen and due to their slightly different angles, strike phosphor dots that glow in red, blue, or green. Each beam is modulated as it sweeps across the screen to vary the intensity of the glow of its color. Since the dots are very small, your eye merges the glow of the dots into a single color at each point on the screen.

Problems occur when we try to feed the TV with a computer signal because most TVs are not aligned to respond to abrupt changes in color. Most changes in color are more gradual in pictures or, when they are not, the total gestalt of the picture draws your attention away from the blurred details. Two places where this does not occur are when text or fine line graphics are on screen. These are common occurances in computer use but not in TV watching.

To minimize the distortions on the TV set manufacturers have included controls on the back of the set and inside which will optimize the tracking of the electron beams across the screen. Most sets are fairly well aligned at the time they are sold, but, they are not aligned to the standards required for computer use. Some improvement is almost always possible.

WARNING: THE FOLLOWING ADJUSTMENTS TO YOUR TV REQUIRE OPENING THE SET. UNLIKE MOST OF THE STICKERS WARNING OF HIGH VOLTAGE THE ONES ON YOUR TV ARE FOR REAL. COLOR TV SETS HAVE VOLTAGES OF 20,000 TO 35,000 V. PRESENT WHICH CAN REALLY HURT YOU.

Before we continue I suggest that you obtain a TV signal generator or enter the program CONVERGE that follows (after all, what do we have in the CC but a TV signal generator). The program uses Extended BASIC, I don't know how you could duplicate it in Color BASIC. My other suggestion is that you obtain the manufacturers' service literature or a copy of SAMS PHOTOFACTS for your set. Unless you are used to working on TVs, the descriptions of convergence proceedures will be necessary.

All set? OK, now carefully remove the back of the set making sure that it is unplugged first. On most sets the line cord is clipped into an interlock that unplugs the AC cord from the set back. Either use a jumper cord or take off the metal clip and use the set cord to plug in the set. Keep your fingers out of the set for now and let it warm up.

NOTE: When I say SMALL adjustments I mean it, if the TV is giving good pictures on broadcast signals, the adjustments required will be very small. Move the controls very small amounts at each try.

We will first set the grey scale. Tune the set to a good strong station. Using the controls on the front of the set, turn the color off. Using the brightness and contrast controls, blacken the screen until only the highlights are still grey or white (black background). On the back of the set find the color gun controls marked red, blue and green drive. Very very slowly adjust these one at a time until the set shows no color cast in the white part of the screen. When this is done leave the controls set as they are now.

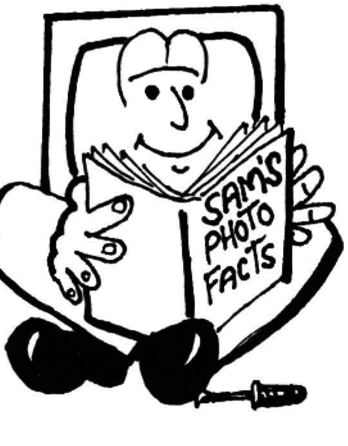
Now hook up your computer to the set and load in the CONVERGE program. Select the single dot pattern from the menu. Refer to the set instructions looking for a section called Convergence proceedure. The first step is called static convergence. This consists of removing as much color fringing as possible from the center. On the back of the picture tube are a set of magnets. These move the individual beams around on the screen. Refer to your literature to find which controls which. Very slowly move the beams around to the point where they overlap as much as possible (you probably will not get them perfect). It helps to move only one at a time as the do interact quite a bit.

When you have obtained the best possible dot, white with little or no fringe of other color, change the program to the dot filled screen. On most TVs there is another board or group of controls called a dynamic convergence panel. This adjusts the combined tracking of the three beams across the screen. Generally there are controls which separatly adjust the top and bottom, sometimes other controls affect the left and right sides of the screen. Referring to your set literature, adjust these controls to reduce color fringing on screen one section at a time. Again, these controls may interact somewhat so work slowly in small increments.

When you have achieved the best compromise, switch to the crosshatch section of the program and tweak the adjustments a bit if needed.

Put the set back together. You are done. Without modifying the set you have done almost everything possible to improve the performance of the TV. At this time I am attempting to bypass the RF modulator in my set and the TVs tuner section for composit video input. This is a much more involved operation which I hope will increase resolution a bit more. If successful, I will report the results in a later issue.

```
10 CLS:PRINT"ENTER THE NUMBER FOR"
20 PRINT"1] CROSSHATCH"
30 PRINT"2] DOT IN CENTER"
40 PRINT"3] DOT SCREEN"
50 GOSUB100
60 PCLS:PMODE4,1:SCREEN1,1
70 ON A GOSUB 120,200,220
80 GOTO10
90 END
100 A$=INKEY$:IFA$="" THEN 100
110 A=VAL(A$):RETURN
120 FORX=1TO192STEP20
130 LINE (0,X)-(255,X),PSET
140 NEXT
150 FORY=0TO255STEP20
160 LINE(Y,0)-(Y,191),PSET
170 NEXT
180 GOSUB100
190 RETURN
200 CIRCLE(128,95),1
210 GOSUB100:RETURN
220 FORX=5TO250STEP20
230 FORY=5TO190STEP20
240 CIRCLE(X,Y),1
250 NEXTY,X
260 GOSUB100:RETURN
```



U-Boat and Orion War
By James Guilford

The relative famine of prewritten software for our favorite computer is showing some early signs of relief. Computer Simulations Company seems to be making efforts to reach the untapped Color Computer software market with more cassette-based programs available from them than from any other single source I know of.

Two of their early efforts are U-Boat and Orion War both designed by Stan Schriefer. I'm not sure if designed by means that he wrote the program or just proposed the idea for it but his is the only name credited.

The two games are very similar in their basic layout with a gun or U-Boat controlled by the user located at the bottom of the screen and with targets moving left to right across the screen. In the case of Orion War the gun fires (using the up key) rockets at a probe which descends after each pass across the screen. Left/right control of the gun position is so sluggish that the user might as well leave the firing position in one place. You only get one chance per pass of the probe to fire your weapon. If you miss, the probe will descend one more level until finally it crashes into a mountain on the right-hand side of the screen and blows up--everything--game over.

Sound effects include a constant beeping tone which increases in pitch until the crash occurs with a shower of colored screens and random tones and a "boop" tone each time one of your rockets detonates (whether or not it hits the probe).

Orion War is okay for the first play but the user quickly learns the proper timing and can soon learn to hit the target every time.

Although similar in general design, U-Boat is much more of a challenge and quite absorbing even though both games are in low-resolution mode and fit inside a 4K memory.

In U-Boat the up arrow fires torpedos at targets which, again, move left to right. But the ships move at different speeds from one another, you can keep up with the progress of the ships by moving your U-Boat with left/right arrow controllers. Mines drifting between the U-Boat and the target ships often get in the way of an otherwise clean shot.

The player plays against a timer which is affected by what happens on the screen. If a torpedo hits a mine, time is added to play time. The further away from the U-Boat a target is when it is hit, the more points are awarded.

Again, movement of the U-Boat is sluggish and only one torpedo is allowed on screen at one time but these problems seem to add to the challenge. You'll find yourself cursing the mines and the near-misses and jamming the fire key harder than necsessary to hit the low-resolution targets.

Sound effects are short sequences of musical tones and have little bearing upon what is happening in the game.

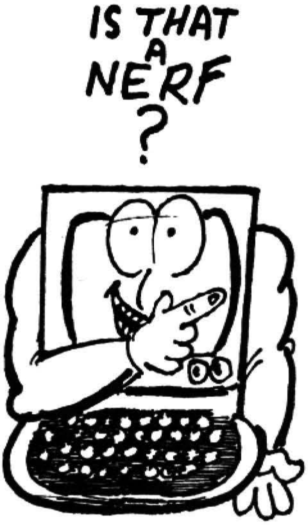
Neither of these games is sophisticated, or what one might call "thrilling" or "dazzling". Neither game comes close to the potential that a CC with Extended Basic and 16K (or better) memory can offer. But that doesn't mean simplicity can't be fun.

The greater play potential is definitely with U-Boat which I highly recommend. At only $5.95 I have already got my money's worth in entertainment. Orion War is too simple, too easy and will quickly bore the user and is the same price as the better offering.

Computer Simulations
305 Hammes Ave.
Joliet, IL 60436

```
                   * HERE IS A PROGRAM WHICH
                   * GENERATES PRIME NUMBERS.
                   * IT WAS WRITTEN ON THE COLOR
                   * COMPUTER USING THE MICRO
                   * WORKS SDS80C - THE POWERFUL
                   * EDITOR/ASSEMBLER ROMPACK.
                   *
                   * THIS PROGRAM USES AN ALGORITHM
                   * WHICH INVOLVES ONLY INTEGERS.
                   * THERE IS NO DIVISION, AND NO
                   * EXPLICIT MULTIPLICATION.  IT
                   * IS FAST - IT GENERATES THE
                   * FIRST 500 PRIME NUMBERS IN
                   * 5 SECONDS (8 SECONDS IF YOU
                   * DISPLAY THEM AS YOU GO).
                   * THE PROGRAM COULD BE MADE EVEN
                   * FASTER, BUT AT THE EXPENSE OF
                   * CLARITY.
                   *
                   * THIS IS THE ALGORITHM USED:
                   *
                   *  PROGRAM PRIMES;
                   *  CONST NUMPRM = 500;
                   *  VAR PRIMES,CCNT =
                   *        ARRAY [1..NUMPRM]
                   *        OF INTEGER;
                   *      I,MP,TEST = INTEGER;
                   *
                   *  REPEAT
                   *    TEST := TEST + 1;
                   *    PRIMES[MP] := TEST;
                   *    CCNT[MP] := TEST;
                   *    I := 0;
                   *    REPEAT
                   *      I := I + 1;
                   *      WHILE CCNT[I] < TEST DO
                   *        CCNT[I] := CCNT[I] +
                   *        PRIMES[I];
                   *    UNTIL CCNT[I] = TEST;
                   *    IF I=MP THEN
                   *      BEGIN
                   *      WRITELN (TEST);
                   *      MP := MP + 1;
                   *      END;
                   *  UNTIL MP > NUMPRM;
                   *  END.
                   *
                   *
                   *
0001 136D                NAM PRIMES
                   *
                   * WRITTEN FOR THE MICRO
                   * WORKS BY ANDREW E. PHELPS
                   * C. 1981 THE MICRO WORKS
                   *
0002 136D          NUMPRM EQU 500        NUMBER OF PRIMES
                   *           TO CALCULATE

0003 136D          CCNT    RMB 2*NUMPRM    CURRENT
```



```
                         *          MULTIPLE OF EACH
                         *          PRIME

1755                     PRIMES RMB 2*NUMPRM    PRIMES WHICH
                         *          HAVE BEEN DISCOVERED
                         *          SO FAR

1B3D                     TEST   RMB 2           NUMBER BEING TESTED
                         *          TO SEE IF IT IS PRIME

1B3F                     MP     RMB 2           NUMBER OF PRIMES
                         *          FOUND SO FAR

                         ******************************
                         *
                         *  START HERE
                         *
1B41 CC0001              START  LDD #1          INITIALIZE LOOP
1B44 FD1B3D                     STD TEST
1B47 CC0000                     LDD #0          ZERO FOUND SO FAR
1B4A FD1B3F                     STD MP

                         *
                         *  OUTER LOOP - TRY NEXT NUMBER
                         *  TO SEE IF IT IS PRIME
                         *
1B4D FC1B3D              NEXPRM LDD TEST
1B50 C30001                     ADDD #1         NEXT CANDIDATE
1B53 FD1B3D                     STD TEST

1B56 BE1B3F                     LDX MP          INDEX INTO TABLE
1B59 ED891755                   STD PRIMES.X    := TEST
1B5D ED89136D                   STD CCNT.X      TIMES ONE

1B61 8EFFFE                     LDX #-2         TO START AT ZERO
                         *
                         *  INNER LOOP - TRY FACTORS
                         *
1B64 3002                NEXTES LEAX 2.X        COUNT THRU TABLE

1B66 10A389136D          BUMPQ  CMPD CCNT.X     MULTIPLY OK?
1B6B 2311                       BLS NOBUMP      IF NOT NEEDED

1B6D EC89136D                   LDD CCNT.X
1B71 E3891755                   ADDD PRIMES.X   ADD 1 MORE
1B75 ED89136D                   STD CCNT.X
1B79 FC1B3D                     LDD TEST        RESTORE D
1B7C 20E8                       BRA BUMPQ       TRY AGAIN

1B7E 26E4                NOBUMP BNE NEXTES      NOT PRIME IF 0
                         *
                         *  END INNER LOOP: WAS IT PRIME?
                         *  IF THE FACTOR WAS ITSELF.
                         *  THEN IT IS A PRIME.
                         *
1B80 BC1B3F                     CMPX MP         MADE IT TO END?
1B83 26C8                       BNE NEXPRM      NO. LOOP
1B85 3002                       LEAX 2.X        INC. # OF PRIMES
1B87 BF1B3F                     STX MP
                         *
                         *  IF THE NUMBERS ARE TO BE
```

```
                          *  PRINTED AS THEY ARE CALCU-
                          *  LATED, THEN THE FOLLOWING
                          *  STATEMENT PRINTS THE NUMBER
                          *  IN D (WHILE SAVING X).
                          *  IF JUST A TABLE IS NEEDED,
                          *  THE CALL IS OMITTED.
1B8A 8D07                        BSR OUTPUT       PRINT NUMBER

1B8C 8C03E8                      CMPX #2*NUMPRM   DONE?
1B8F 25BC                        BLO NEXPRM       IF NOT, LOOP

                          *
                          *  THE SWI STATEMENT CALLS THE
                          *  ABUG MONITOR BEFORE THE
                          *  RETURN TO THE EDITOR, SO
                          *  THE TABLE OF PRIMES MAY BE
                          *  EXAMINED OR SAVED.
1B91 3F                          SWI

1B92 39                          RTS              RETURN TO EDITOR

                          ********************************
                          *
                          *  BASE 10 OUTPUT ROUTINE
                          *
                          *  SAVES D,X.
                          *  PRINTS NUMBER IN D IN BASE 10
                          *  (UNSIGNED) WITH CARRIAGE RET.
                          *  USES NO GLOBAL STORAGE.
                          *  CALLS $A30A FOR SCREEN PRINT.
                          *
1B93 3416                 OUTPUT PSHS D,X         SAVE REGISTERS
1B95 308D0024                    LEAX TAB10,PCR
1B99 327F                        LEAS -1,S        ROOM FOR COUNT

1B9B 6FE4                 A@     CLR 0,S          CLEAR COUNT
1B9D 6CE4                 B@     INC 0,S          COUNT SUBTRACTS
1B9F A384                        SUBD 0,X         TRY SUBTRACT
1BA1 24FA                        BHS B@           BRANCH IF IT FIT
1BA3 E381                        ADDD ,X++        ADD IT BACK ON
1BA5 3406                        PSHS D
1BA7 A662                        LDA 2,S          GET COUNT
1BA9 8B2F                        ADDA #'0-1       MAKE ASCII
1BAB BDA30A                      JSR $A30A        OUTPUT CHAR
1BAE 3506                        PULS D
1BB0 6D01                        TST 1,X          END OF TABLE?
1BB2 26E7                        BNE A@           LOOP IF NOT END

1BB4 860D                        LDA #$0D
1BB6 BDA30A                      JSR $A30A        CARRIAGE RETURN
1BB9 3261                        LEAS 1,S         CLEAN UP STACK
1BBB 3596                        PULS D,X,PC

1BBD 271003E800           TAB10  FDB 10000,1000,100,10,1,0

1BC9                             END START

BUMPQ  1B66  CCNT   136D  MP     1B3F  NEXPRM 1B4D
NEXTES 1B64  NOBUMP 1B7E  NUMPRM 01F4  OUTPUT 1B93
PRIMES 1755  START  1B41  TAB10  1BBD  TEST   1B3D
```