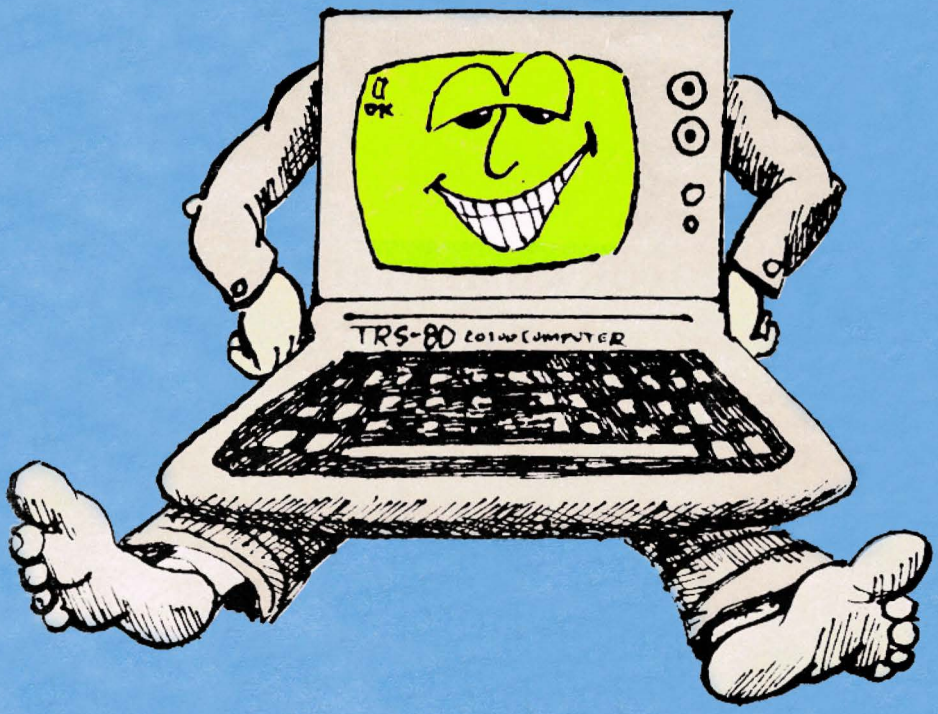
Color Computer News

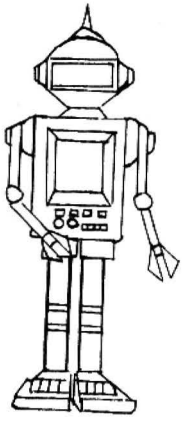
The Best of 1981



$2.50

Color Computer News

May / June 1981



Color Computer News
Volume 1, Number 1
REMarks

I wanted to call this section REMark but the Heath Company took that years ago, so I guess we'll go with REMarks unless someone can come up with something better.

Since this issue is the premier I thought I'd tell you all about why this newsletter is being written, myself, and REMarkable Software. Let's go in reverse order.

REMarkable Software started long ago as a custom programming company primarily aimed at business. The problem that hardware manufacturers seem to produce less than exceptional software and don't usually have the facilities to customize their software for their customers. About spring of 1980 I was forced to change attitudes about a machine I had been opposed to since it's inception. I discovered that the TRS-80 Model I wasn't all that bad a computer, so I bought one and REMarkable Software entered the world of personal computing.

Recently Tandy introduced 2 new computers, the Model III and the Color Computer, and we decided to support the users of these new machines to help the new computer owners over the difficult times. The Color Computer was chosen for two basic reasons, first because I am particularly excited about the 6809 MPU and second because the Model III is an updated version of the Model I and as such is well supported by magazines such as The Alternate Source, 80 Microcomputing, 80-US and Chicatrug News. Don't get me wrong, we are not just a "beginner's" newsletter, as far as the Color Computer is concerned we are all beginners.

As far as plans for this newsletter we are at this point very open to suggestions. We want your articles and comments, you have to support CCN in order for CCN to support you. Please forgive any rough spots. Any new venture starts out slow. I personally wrote much of this issue so send in your articles and programs to share with other CC owners. It took me a long time before I ever wrote an article for a computer magazine, I've published other magazines (Ham Radio related), but never felt that I had anything unusual enough for a computer magazine. At this point in the CC's life we all know something that another needs to know. Software reviews, unusual uses and explanations of the things you have learned. Don't worry about grammar or being perfect, that's what we have editors for.

We have also implemented another pet project of mine. That is a real user's group for the Color Computer. Here's how it works. We are collecting original programs form CC owners and putting several of them on one cassette for distribution to other CC owners on a cost basis. There will be anywhere from 1 to 100 programs on each cassette and CCN subscribers may order them for $7.95, at that price only one really good program makes the whole cassette worth while. Anyone contributing a program will receive a copy of the cassette where it appears. All programs appearing in CCN will be available by this method unless the contributing author requests otherwise. The other part of the User's group is that we can supply you with names of other people in your area so that you may form a local user's group. Just send a self addressed stamped envelope and a list of zip codes in your area. If you do not wish to be included in this project, send us a letter stating so. Also we occasionally make our subscriber list available to responsible Color Computer vendors if you don't wish to be included so state so in a letter to us.

There are three software reviews in this issue. A good one, one is fair, and the last is poor. We certainly hope that you have seen so that we may publish your review and warn the other readers about the bad ones and support the companies producing the good stuff. There is a lot of trash out there right now, let us learn from your mistakes.

I read a lot of magazines during the course of a month and you probably share my feeling about the large volume of ads in the mags. I read them first but some of the mags seem to have more ads than articles. We are in a good position. Because we are a dedicated newsletter we will attract fewer ads but all of them will relate directly to you, the Color Computer owners. Remember that your $9.00 a year just covers the cost of printing and postage, the advertisers pay the authors of articles and pay the salaries of our people. We will try to keep the ratio of articles to ads to about 4-1 (4 pages of article to 1 page of ad copy). We have informed potential advertisers that we reseve the right to refuse ad copy from companies that do not fulfill their claims or that we receive complaints about. Let us know how our advertisers deal with you and Color Computer companies advertising in other magazines.

Beginning next month we will have another regular feature. We have recruited a technical advisor who will answer your questions. To use this service send your questions to

the attention of Tech Advisor. The most interesting questions will be printed in CCN. If you wish a personal reply, include an SASE and we'll get back to you as soon as possible.

I guess all that's left to talk about is myself. My name is Bill Sias and I'm the principal owner of REMarkable Software. I intend to write the Editorial and Assembly Language Programming sections of this newsletter. My primary function at REMarkable Software is writing business software, so there are a lot of times when I'm not here. If you would like to talk please make your calls after 5:00 EST and you can be fairly sure that I will answer the phone or at lest be available. If you order anything from us by phone please call at the same times so that I can speak to you. That's enough for now. Enjoy this first issue and please return the survey so that we can provide you with what you want.
Bill

LETTERS

Here is $9.00 for the first year of Color Computer News. What are you looking for in the way of articles?
Mel Evans
Ann Arbor, MI
* Mel, I think the answer to your question is best answered by Richard Batt.

Dear Sirs:

Enclosed is a check for $9.00 for a subscription to "The Color Computer News" as mentioned in the New Products section of 80 Microcomputing, April '81 issue. I have just purchased a Color Computer and want to find out as much as possible about it - both programs to run and internal construction (hardware and software).
Richard Batt
Buffalo, NY
* In addition to what Richard mentioned, I would like to add software and add-on hardware reviews, unusual aplications and most anything else. How about writing an article about what you wanted to know before you got your CC and have found out since.

Dear Sirs,

Enclosed please find a check for $9.00 for a one year subscription to Color Computer News. If you have other material related to the CC please let me know.
David Bodnar
Pittsburgh, PA
* A smile began to grow across the merchant's face as he rubbed his pudgy palms together. "Well", he said, "let me tell you about it."

I've sent a copy of our catalog to David. At this point we are developing and accepting Color Computer software for possible publication and resale. Like many retail operations, we also sell products produced by other companies. We have, however, made it a policy not to mention this end of the business in the magazine other than our advertising and addressing specific questions. We refuse to become what Radio Shack's Newsletter is.

To whom it may concern,

I'm very gald to see that someone else besides myself thinks of the Color Computer as something other than a toy. It is clear to me that your company will pioneer the software market for the Color Computer. (Just from reading the short letter sent into 80 Microcomputing.) Therefore, I wish to order a subscription to your Color Computer News. Enclosed you will find a money order for $9.00. I am looking forward to my first copy.

Thank you very much for your time.

Cordially Yours
Calvin E. Cock III
* Calvin, you're right, we don't think of the Color Computer as a toy and anyone that does had better take a closer look. For one thing, the 6809 is a much more powerful chip than any other 8 bit chip on the market and benchtests have shown it to be more powerful than many of the 16 bit chips. Tandy stated from the start that it wasn't a business machine, they had better look

at our catalog. Thank you very much for the great complement about being pioneers, we're trying.

Dear Sirs,

I saw the article in the "New Products" section of April 1981 edition of 80 Microcomputing announcing your inauguration of the "Color Computer News". Enclosed is a check for $9.00. Please initiate my subscription for one year. I have had a Color Computer for 4 months, and have added Enhanced color Basic and expanded the memory to 32K bytes. The only two problems I have encountered to date with my Color Computer are the lack of a disk operating system and a paucity of available software. I hope that Color Computer News will help alleviate these problems by unifying color computer users, and making the vendors aware of the new market.

Thank you very much for filling a much needed void.

Sincerely,
David Dacus
New Mexico

* I think David has stated our purpose more clearly than I could have. I have been thinking about offering a Users group service. We are in a prime position to let you know about other Color Computer owners in your area. The way it could work would be to send us a self addressed stamped envelope and a list of zip codes in your area and we will send back a list of names and addresses that fall in those zip codes. If you don't want to be included in this send us a letter and inform us of your desire not to be included. I also agree with the complaint about disks.

Dear Sirs

Please send me a one year subscription to "Color Computer News" for the Radio Shack Color Computer.

I read about you in the April edition of 80 Microcomputing. I have a 4K Color Computer and am using it as a terminal and as entertainment for my family. I hope to be able to transfer files to and from a time sharing computer in the future.

Sincerely,
William Miller
Boise, ID

* How many others of you have an application other than programming and game playing?

Dear Sirs,

Enclosed find $2.00 to try out your Color Computer News. I don't want to get it if it isn't what I want. I saw your ad in 80 Microcomputing. I was disappointed in that mag (80 Microcomputing). Nothing on Color Computers and we got a one year subscription. I wish I'd only got one issue now.

Randy Martin

* I don't think that you can really blame 80 Microcomputing for their lack of coverage of the Color Computer. A magazine can only publish articles sent in by readers and the majority of their readers have Model 1s. I think you have to look at it from the other side of the coin also, you as readers have to accept some responsiblity to educate other readers and to share your discoveries. Color Computer News will only be as good as you make it. If we aren't what you are looking for you have the ability and responsibility to make it what you want. I firmly believe in this fact! Readers totally control magazines, editorial policy, articles published. If you don't like what's happening then WRITE. Don't give the excuse that you haven't learned enough about it yet, if you've purchased a program lately then you have a review, if you've written a program that's an article in itself.

FAST BASIC GRAPHICS

When I first brought my Color Computer home I was disappointed in the graphics functions. It would seem that a computer whose major selling point is high resolution color graphics would support high resolution color graphics. It has since turned out that it will, see the article by Tom Rosenbaum in this issue, but Tandy certainly hasn't helped much. In fact they claim that it won't (I love it when an expert says that something is definitely not true and turns out that it is). Be that as it may let's look at low resolution high speed color graphics.

I should clarify that I am referring to the 8K or Level I ROM and not to Enhanced Basic. If you have been writing many programs using SET and RESET for graphics you must be aware by now that these commands do not allow enough speed for real-time graphics. The alternative is to use the graphic characters built into the machine. Fire up your computer and type in the following one liner:

```
10 FOR X=128 TO 225: PRINT CHR48X9;: NEXT
```

What you see are the building blocks for fast graphics. What you see are the building blocks for fast graphics. Refter to table 1 and we'll continue. Using Table 1 we can construct anything that SET and RESET can. To do so pick a color, determine which pixels are to be SET and add the color's number to the numbers referring to the pixels you need to be set. The resulting number can be printed as a CHR$ (pronounced character string). Try this:

```
10 CH$=CHR$(248)
20 BL$=CHR(32): REM A SPACE
30 FOR X=0 TO 500 STEP 33
40 PRINT@X,ch4;: PRINT@X,BL$;
50 NEXT X
```

Fast? O.K. next, using a copy of the video work sheet in the back of your manual construct a table showing all of the building blocks for future us. Done? Good. Now we can build entire figures using the same techinque described earlier. For example, GR$=CHR$(248)+CHR$(240)..... For characters that are more than one block high add CHR$(11) in the proper places and don't forget to build some blank characters to erase your graphics as you move your characters around the screen and to CLEAR enough string space to hold the figures you are building.

If your program uses more than a few of these characters you will notice that it takes some time to build all of these figures. We can resolve this also by making the strings a part of the program they are used in.

Model I owners will recognize that we are about to do some string packing. In order to wrk this bit of magic we need to locate and understand two pointers in Basic' scatchpad and to understand how strings are stored in memory. The pointers are: Start of variable storage and End of variable storage. The pointer for start of variable storage is located at 27 and 28 decimal and the pointer for End of variable storage is at 31 and 32. To calculate either of thee use the formula:

```
LOCATION=PEEK(ADDR1)*256+PEEK(ADDR2)
```

Therefore Start of variable storage is at PEEK(27) * 256 + PEEK(28) and End of variable storage is at PEEK(31) * 256 + PEEK(32). The location that these formulas return will depend on the size of the program in memory and the amount of string spaced CLEARed.

Strings are stored in memory as: the ASCII value of the first letter then the ASCII value of the second letter plus 128 then the length of the string and the next two bytes are the actual address where the string is contained in memory. Therefore to locate AA$ we need to serch for three consecutive memory locations that contain: 65, 193 and since the example will be fifteen characters long, 15. 65 is the ASCII value of A, 193 is 65 plus 128 and 15 is the length of our string. Type in these lines:

```
10 AA$="..............."
1000 LS=PEEK(27)*256+PEEK(28)
1010 HS=PEEK(31)*256+PEEK(32)
```

This creates the dummy string we will use and locates the low and high addresses of variable storage. Run it and type PRINT LS,HS to see where variable are stored now. These numbers will change as we add more to our program. Now add:

```
1020 FOR SL=LS TO HS
1030 IF PEEK(SL)=65 AND PEEK(SL+1)=193 AND PEEK(SL+2)=15 THE VL=SL+4 ELSE NEXT SL
1040 LC=PEEK(VL)*256+PEEK(VL+1)
```

Run it again and type PRINT LC. LC is the location of AA$. Again this will change as we add more lines, type PRINT LS,HS to verify that these did change as a result of adding more corde. Add these lines:

```
1050 FOR MS=LC TO LC+15
1060 POKE MS,RND(127)+128
1070 NEXT MS
1080 PRINT AA$: LIST 10
```

Now run it again and you should see a line half filled with graphics and line 10 should be 10 AA$="the same graphic characters". If you now were to CSAVE the program AA$ would be the same every time you CLOADed it. The graphic characters that you build become a part of the program that you are writing, thus eliminating the long pause at the beginning of the program used to build the strings.

You could add more strings and use data statements to hold the values for the block that you need for your figures. Be sure to change line 1030 to find the string that you are looking for.

When you have packed all of the strings that you need be sure to eliminate line 1000-1080 as they are no longer needed. This technique isn't as nice looking, as far as the graphics created, as the technique for high resolution that Tom's article shows but if low resolution is what you need them this is the way to go.

Table 1

GREEN	128
YELLOW	144
BLUE	160
RED	176
BUFF	192
CYAN	208
MAGENTA	224
ORANGE	240

8 / 4

2 / 1

Editor's Note: A rough draft of this article was sent to 80 Microcomputing by accident. The finished version is printed here due to the fact that the copy in 80 Microcomputing was a rough draft.

VIDEO DISPLAY MODE	VDG SIGNALS					SAM SIGNALS		
	A/N	INT	INV	CHR EXT	CSS	V2	V1	V0
INTERNAL ALPHANUMERIC	0	X	X	0	X	0	0	0
EXTERNAL ALPHANUMERIC	0	X	X	1	X	0	0	0
SEMIGRAPHIC-4	0	X	X	0	X	0	0	0
SEMIGRAPHIC-6	0	X	X	1	X	0	0	0
SEMIGRAPHIC-8	0	X	X	0	X	0	1	0
SEMIGRAPHIC-12	0	X	X	0	X	1	0	0
SEMIGRAPHIC-24	0	X	X	0	X	1	1	0
FULL GRAPHIC 1-C	1	0	0	0	X	0	0	1
FULL GRAPHIC 1-R	1	0	0	1	X	0	0	1
FULL GRAPHIC 2-C	1	0	1	0	X	0	1	0
FULL GRAPHIC 2-R	1	0	1	1	X	0	1	1
FULL GRAPHIC 3-C	1	1	0	0	X	1	0	0
FULL GRAPHIC 3-R	1	1	0	1	X	1	0	1
FULL GRAPHIC 6-C	1	1	1	0	X	1	1	0
FULL GRAPHIC 6-R	1	1	1	1	X	1	1	0

TABLE 2

HIGH RESOLUTION GRAPHICS

by Tom Rosembaum of
Spectral Associates

The TRS-80 Color Computer is one of the best values on the personal computer market today. However, it is also the least understood and supported computer at the present time. A considerable amount of this mystery is undoubtedly caused by he fact that the Color Computer is built around the 6809E Microprocessor (MPU), the 74LS783 Synchronous Address Multiplexer (SAM) and the 6847 Video Display Generator (VDG). It is still difficult to get accurate data sheets from Motorola on these parts.

It has been my experience that most people are interested in the Color graphics capabilities of the Color Compter. The purpose of this article is to explain these capablities. It is assumed that the reader has a fair knowledge of machine language programming as only machine language programs will be able to effectively utilize the graphics capabilities of the machine. (Editor's Note: While this statement is true, you can use much of this information fom Basic, although the time used by Basic is a serious problem with any good game. Bill)

The CC display as seen on your TV set is stored in the computer's memory. The VDG fetches the data and formats it properly to be output to the display. However, it is the SAM which provides the data to the VDG and the SAM must know where the data is stored. In the TRS-80 Models I/III the video display data is stored in a specific location in RAM. The Color Computer may have it's video display stored anwhere in memory and the SAM must be programmed with the starting address of the display. How? Well, all Gaul was divided into three parts but the Color Computer's memory is divided into 128 parts. The video display address of the left hand display element is stored in locations FFC6-FFD3 (Decimal 65478-65491). Writing to these addresses will set or clear the appropriate bit in the SAM's internal display offset register ($F0-$F6). The data is not relevant since the data buss is not connected to the SAM. Each register bit has two locations, one even and one odd. Writing to the even bit will clear the register and writing to the odd bit will set it. The address stored in the display offset register coresponds to the upper left display element. By programming different data into the display offset register the display can be moved to any 1/2 K page boundary in RAM. This does not mean that any data is moved around in RAM; only that the address where the SAM looks to find video display data is changed. This feature may be used to create animated graphic displays by filling several graphic pages with pictures and paging back and forth between the pages.

Once the display offset has been set, it is necessary to choose the display mode. The VDG processes data in order to display it; the SAM provides the data to the VDG. Both the SAM and VDG must be programmed with the display mode information. The SAM is programmed by setting or clearing bit in the 3-bit VDG mode register (V2,V1,V0) which is accessed by addresses $FFC0-$FFC5 (see figrue 1). The VDG is programmed through the five high order bits of PIA #1, Port B. Table 1 shows which VDG pins are connected to PIA #1.

Port B of PIA #1 occupies address $FF22. Storing data in the 5 high order bits of address $FF22 will program the VDG mode pins.

The modes of the SAM VDG mode register and the data on the VDG mode pins must match or you will be in a "garbage" display mode which will not properly display your data. Table 2 (Page 26) shows the proper mode correspondance between the SAM and VDG. A detailed description of the VDG modes and their operation is beyond the scope of this article. Most of that information may be obtained from the Motorola spec sheet on the MC6847.

Anyone thinking about using custom high resolution graphics for a program should get acquainted with machine language programming. The highest resolution graphics mode requires 6K of RAM for the video display. Any attempt to make a fast moving program in Basic will be doomed to failure because of the massive amount of data to be moved.

In the next issue, I will describe the VDG display modes in detail and demonstrate how to build a routine which will draw a figure on the screen and move it around under joystick control, it will all be in machine language, so you had better have your Editor/Assembler and Monitor by that time.

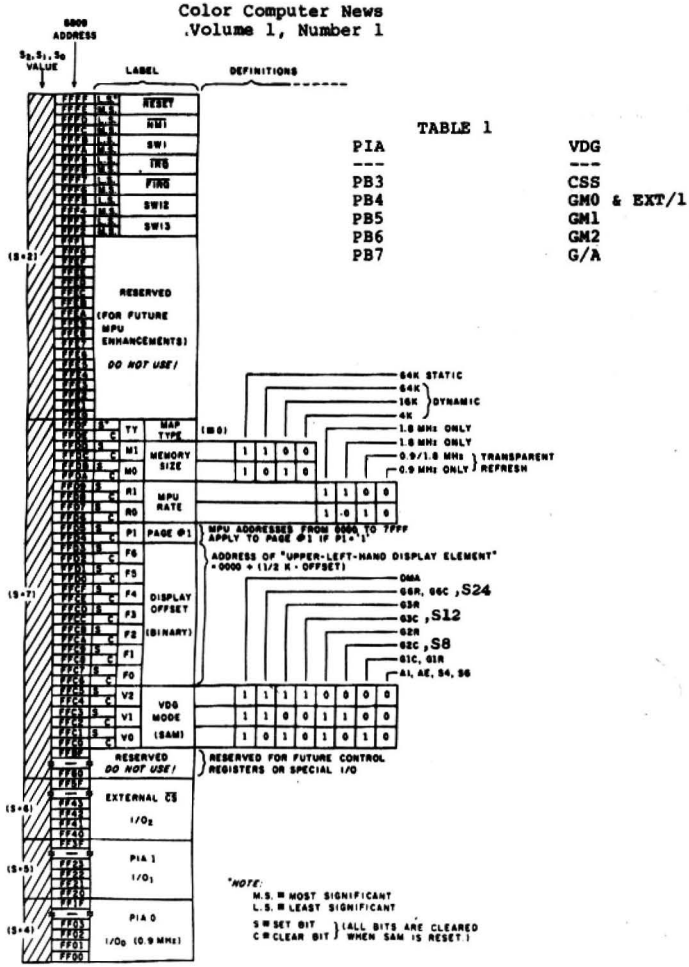


TABLE 1

PIA	VDG
PB3	CSS
PB4	GM0 & EXT/1
PB5	GM1
PB6	GM2
PB7	G/A

Figure 1: Memory map of the Color Computer. The SAM registers and interrupt vectors are detailed.

MACHINE LANGUAGE PROGRAMMING

Addressing refers to the location of the data needed to complete an instruction. Some instructions are complete in thenmselves while other instructions require more information before they can operate. The various addressing modes allow different ways of locating that data. Before I get a lot of letters on the subject, let me explain that this is a generalizaiton and the exceptions will be covered as needed. But if you think of it as data the flow of understanding will be much faster.

In Inherent addressing the opcode is complete in itself. An instruction using Inherent addressing will have nothing in it's operand field. An example of this mode would look like: 4F. 4F is the opcode for the instruction CLRA.

In Immediate addressing the data used by the operation is the byte(s) immediately following the opcode. For example if we used LDD #$FFAD, the data used by the operation would be $FFAD and in memory it would look like: CC FF AD. CC being the opcode for the instruction LDD when using the Immediate addressing mode. If this part of our program were assembled at $2000 it would look like:

$2000 CC $2001 FF $2002 AD

$FFAD is the data needed and as you can see it Immediately follows the opcode. The # specifies that we wish to use the immediate addressing mode.

Extended addressing means that the 2 bytes immediately following the opcode contain the address of the data. Let's change our example above to use Extended addressing. LDD > $FFAD means to load D with the data contained at $FFAD and because D is a 16 bit register we will also get the data at $FFAE. Let's assume that our program is still located at $2000 in memory it would look like:

$2000 FC $2001 FF $2002 AD $FFAD ZZ $FFAE ZZ

FC is the opcode for LDD when using the Extended mode of addressing and when the MPU reads this from memory it will fetch the next two bytes of memory to discover the address of the data which in this case is $FFAD. The > forces the Extended addressing mode although it is usually not needed to specify an addressing mode.

Direct addressing is the first mode that used the DP register and is the fastest mode we have discussed so far. To use this mode you must first SETDP to the "page" of memory that you wish to use (upon start-up of the MPU it is set to page 00) and it is then used to specify the MSB of the address when the data is contained. You then only need to specify the lower byte of the address where the data is located. We'll continue to use the example above but now with direct addressing we will assume that we have already SETDP to page $FF. The source code would look like: LDD <$AD, the assembler knows that we are using Diret addressing and in memory it would look like:

$2000 DC $2001 AD $FFAD ZZ $FFAE ZZ

When the MPU reads the DC opcode it then makes the contents of the DP the top 8 bits of the address and the byte following the opcode into the lower 8 bits of the address and in our example that will form the address $FFAD and since D is a 16 bit register we get the data at $FFAE also. The < specifies Direct addressing to the assembler but is almost never necessary.

Register addressing is the addressing mode that specifies a register(s) as the data that we are looking for. To determine the opcodes for this mode you need the following table:

0000=D	0101=PC
0001=X	1000=A
0010=Y	1001=B
0011=U	1010=CC
0100=S	1011=DP

As n example we will use TFR X,Y since this is a two byte opcode the second will be formed form the above table. The first nybble is one register and the second is the other we will use 00010010 for X,Y or 12 in Hex since the opcode for TFR is 1F in memory it would look like:

$2000 1F $2001 12

There is no specifier for Register addressing because the instruction specifies it within itself.

Indexed addressing uses one of the Index registers (X,Y,S,U and occasionally PC) as part of the Effective Address of the data. In addition an Offset may be used with the Index register. The offset may be +/- 4, 7 or 15 bits and the offset may be either constant or specified by an accumulator. For our example let's use: LDD A,X this will load D with the contents of the address specified by X plus the offset in A. In memory this will look like:

$2000 EC $2001 96

Where 96 represents the binary number 10010110. B7 specifies that it is not a 4 bit constant offset, B6-5 specify that we are using the X register (00=X, 01=Y, 10=U, 11=S binary) B4 specifies that we are non Indirect (0=non Indirect, 1=Indirect) B3-0 are the addressing mode field. In addition we can Index address Indirectly as in the examples earlier and we can use Auto Increment/Decrement Indexed addressing. This mode allows the Index register to be incremented or decremented while being used. The programmer may specify a 1 or 2 byte increment to the register after accessing it for the address it contains or a 2 byte decrement before accessing the address it contains. The source form would be: LDD ,X++ to load D with the data pointed to by X and then it would be incremented by 2. Indirect would be specified by enclosing the operand in brackets (unfortunately I can print brackets on this computer).

Relative addressing is used by adding a signed constant to the address contained by the PC. This is used with a conditional branch instruction and if the branch is taken the PC plus the constant offset become the new value contained by the PC. For example BEQ $FF,PCR would add $FF to the address in the PC is the Z FLAG is set and continue execution at the address now in the PC. The offset can be either 8 or 16 bits. For a 16 bit branch the instruction would read LBEG and the offset would be a 2 byte value ($A1E2) for example). You may also add Indirection to this instruction by having the offset contained in another register LBEQ X,PCR.

Extended Indirect is when the 2 bytes after the opcode contain the address of the location of the data.

Absolute Indirect is the addressing mode used by the MPU during reset and interrupt handling. More on that later.

The 6809's instruction set divides easily into six types of instructions. The addressing modes we just looked at are involved mostly with the Branch type instructions but the others are: Loads, Arithmetic, Logical, Tests and Interrupts. It's nearly impossible for anyone to write a program that uses only one type of instruction, but we need to study them in a logical order. For no reasons other than the fact that I decided to do it this way we will go in this order: Load, Arithmetic, Branch, Test, Logical and the Interrupts.

The load group of instructions are involved in the process of placing or moving data to, from, or between registers and/or memory locations. Before we start, let's look at the simplest of all of the instructions, NOP. NOP stands for No Operation and it does just that, nothing. It's used as a place holder or a timing delay.

The load group of instructions come in two sizes, 8 and 16 bits. They consist of loads and stores. The loads place data into a register and the stores place data into a memory location. CLR is another load type intruction and it puts the number 00 into either a register or memory location. CLRA and CLRB load the A and B registers with 00 and are the only 8 bit versions of this instruction. You may also execute CLRD,CLRX etc. They may use Direct, Extended or Indexed addressing modes as explained in the section on addressing modes.

The load group consists of LDA and LDB in the 8 bit group and various forms in the 16 bit group. The addressing modes that may be used by this instruction are: Direct, Extended, Immediate and the various forms of Indexed.

The store group may be used to load a memory location with the contents of a register. Direct, Extended and Indexed are the addressing modes. For example, STA $FFAA would store the data in register A into memory location $FFAA, STD $FFAA would store the two bytes of data contained in the D accumulator into memory locations $FFAA and $FFAB, since the D accumulator is 16 bits wide it requires two bytes of memory to hold it's contents.

Another instruction is used to Load the Effective Address into a register in preparation for a branch instruction. This instruction LEA performs the math necessary to compute the effective address and put it in a register for future use.

The last type of load instruction controls loads between registers themselves. There are two types, TFR and EXG. TFR causes the contents of one register to be moved into another register of the same size while EXG causes the two registers to exchange values.

This covers the load group of instructions. I realize that we are going rapidly but the intention of this article is to prepare you for writing and understanding the programs that we will write shortly. It is impossible to fully understand without really applying the basics. As we write the programs all of this will become very clear.

The Arithmetic group of instructions consist of adds, subtracts and one multiply. The first ADD is also an 8 or 16 bit operation depending on the register(s) involved. The 8 bit form is used with the two 8 bit accumulators A and B. This series of instructions is very straight forward in that it performs a strict ADD. For example ADDA $FFAA will add the data in the A register with the data at memory location $FFAA and place the result in the A register. ADDD $FFAA would add the content of the D register with the contents of memory locations $FFAA and $FFAB and place the result in the D register. The addressing modes available to this instruction are: Immediate, Direct, Indexed and Extended. There is another form of ADD referred to as ADC which performs the usual ADD operation but it also includes the value that is the Carry FLAG. This allows higher precision in the operation or series of operations.

Subtractions are much the same as the Add operations with the apparent difference. There is also a subtract with Carry which is the equivalent of ADC and is called SBC.

The next Arithmetic operation is MUL which performs an unsigned multiply of the contents of A register and B register the result appears in the D register.

The next forms of Arithmetic instruction are INC and DEC and they simply INCrement or DECrement a register or memory location. There are also instructions to Decimal Adjust the Accumulator for BCD operations (DAA) an operation to cause the contents of a register to be changed to a negative value (NEG) and to sign extend a value by taking it from an 8 bit register and placing it in a 16 bit register which is used to prepare for higher precision results (SEX). I had nothing to do with the naming of any of the instructions.

The last type of Arithmetic operations are the rotate group and are perhaps the most confusing of all of this group. Shifts may be performed either to the right or left and what happens is that the bits in the byte are rotated to the direction specified, for example ASL (Arithmetic Shift left) would change the binary pattern 10110100 to 01101000 because it moves all of the bits to the left and puts a 0 into the now empty first position and bit 7 is placed into the Carry flag. The forms of this type of operation are: ASL (Arithmetic Shift Left) ASR (Arithmetic Shift Right). The right shift causes bit 0 to be placed into the Carry flag bit 7 is held as it was.

All of the branch instruction have names that fully describe their operation as follows:

BCC	Branch if Carry flag is clear
BCS	Branch if Carry flag is set
BEQ	Branch if the last operation obtained a zero result thereby setting the Z flag
BGE	Branch if >=0, if the result of last operation was
BGT	Branch if the register used in the last operation was greater than the memory location used.
BHI	essentially the same as above, branch if the register was Higher
BSH	Branch if the register was >=the memory location used by the last operation
BLE	Branch if the register used in the last operation was <= the memory location used
BLT	Branch if the register was < the memory location
BLO	is essentially the same as BLE
BLS	is essentially the same as BLO
BMI	Branch if the result of the last operation caused a negative number
BNE	Branch if the register and memory location used in the last instruction were not the same

BPL	Branch if the result of the last operation caused a positive result
BRA	Branch Always. Branch no matter what the result of the last operation was
BRN	Never branch, another NOP
BSR	Branch to a subroutine
BVC	Branch if V flag is clear
BVS	Branch if V flag is set
JMP	causes program execution to continue at the specified address
JSR	causes a jump to a subroutine
RTS	causes a return from a subroutine

Most of these motion instructions are conditional and are used in decision making instruction, controlling the flow of program execution based on conditions specified.

The test group of instructions perform operations such as BIT which compares a register with a memory location and sets the Flags as necessary. CMP compare a register with a memory location and sets flags based on whether the register is higher, lower or equal to the memory location. TST is essentially the same as BIT. Many of the Branch instructions perform their own tests and are, in my opinion, much more reliable and easier to work with.

The logical group of instructions require some knowledge of Boolen Algebra and it is beyond the scope of this article to impart such knowledge. If you aren't sure of your abilities in this area, drop me a line and I'll either annswer your question directly or recommend some books. The logical instructions supported by the 6809 are: AND, OR, EOR (exclusive OR), COMplement, LSL (Logical Shift Left) and LSR (Logical Shift Right). We will explain these instructions when the need to use them arrives. In the meantime, I strongly recommend that you brush up on logical operations.

The Remaining instructions consist of a few more rotates such as ROL and ROR. Both of these rotate with the Carry flag as part of the rotate. ROL rotates all of the bits left and the Carry flag becomes the new least significant bit while ROR rotates right with the Carry flag becoming the new most significant bit. The next four instructions have to do with the stacks. PUSH takes the contents of the register(s) specified and PUSHes them onto the stack. The source form of this instruction is PSH and the name of the stack that you want to push it/them to. PSHS A,D,X would push the contents of the A,B and X registers on the S stack. PSHU X,Y would push X and Y onto the U stack. The purpose for "stacking" registers is to save the contents for future use while freeing up the register to do something else. The opposite of PSH is PULL and it works exactly the opposite of PSH. Any good 6809 Assembler will allow assigning register(s) labels. We could assign the registers A,B,X and Y to the label MAIN by issuing the Assembler instruction: MAIN REG A,B,X,Y. We'll talk about labels when we get into programming.

All that's left are the interrupt instructions. We'll save those for later also as they are better understood after much of this gets clearer.

We have two books now about the 6809. One is THE MC6809 COOKBOOK by Carl Warren and the other is 6809 ASSEMBLY LANGUAGE PROGRAMMING by Lance Leventhal. Both of these have good and bad points. Cookbook is good if you want an overview and ALP is good if you are ready for something a little heavier. Cookbook spends a lot of time reviewing Motorola's 6809D4 evaluation board and even more talking about VTL-09 which is a tiny Basic available for 6809. I'm not very interested in either one. ALP goes into a lot of examples and is in my opinion the better of the two books. Both are good but neither is enough in themselves. For that matter neither is this article. The MC6809 Cookbook retails for $6.96 and 6809 Assembly Language Programming is $16.95. 6809 Assembly Language programming is at least $10.00 better.

Unitl next time if you have any questions or think that something isn't clear enough let me know. Next time we'll start programming and I strongly recommend that you obtain an Assembler or at least a monitor program. How about writing a monitor in Basic and sending it in so others can use it?

Color Invaders By Spectral Associates
141 Harvard
Tacoma, WA 98466

Color Invaders is a copy of the arcade game "Invaders" and is a good rendition of the original in many details. If you're not an "arcade freak", the object of the game is to kill all of the moon monsters before they can reach the earth. The monsters are arranged in 8 columns of 6 and advance toward the bottom of the screen in a zig-zag pattern, all the way across the screen before moving down a row. In addition there is a mystery invader, which looks more like a flying saucer on both versions, which moves more rapidly than the rest of the invaders. A hit on this one give you extra points.

Spectral's version is very much the same with the exception of an additional feature. The new feature is a "shield" or a short blue line that can be used to block the invader's blasts. The shield is a good thought but the thing moves so fast that it takes more concentration to control it than it does to fight the creatures without it. The sound created by Spectral's invaders is as annoying as the arcade version's. It ranges in tone from a love-sick cricket to a demented toad. The arcade version is just as bad, if not worse.

Apple Invaders is another version of this arcade game and I compared the two versions almost side by side. In reality there was about a two hour gap between viewing each version. In my opinion Spectral's version won in all ways but one. Spectral's graphics are equal with the Apple version but the playability* is much better. The Apple verison uses joysticks and Apple joysticks are well known to be extremely slow. This is a definite handicap in a game like this one. Spectral's version uses the arrow keys to move your turret and the space bar to fire the cannon, a much better choice in my opinion. The shield mentioned earlier is controlled by the 1 and 2 keys. Apple invaders did win in the sound department. The author of the Apple version apparently decided against using the static sounds of the arcade game.

Spectral's Color Invaders in a well-thought out version of the popular arcade game and is probably the best Color Computer game out now for arcade freaks and other good people.

Frog Race
By B.E. Erickson

In the course of seeking out software for resale I came across some odd occurances. One of the techniques I use is to sit down at the computer with either a stack of magazines or some other source of ads. Recently I was doing this with the Radio Shack sourcebook and since I was looking for Color Computer software I came upon an ad for a game called Frog Race by B.E. Erickson. The listing said that it was for Color Computer and that he would sell this one for $3.00 as an example of the other things he had written. I sent him my $3.00 and a letter requesting dealer information. The response was quick but it came from the Software Stockpile* and included the dealer information I had requested and a cassette marked Frog Race. After "doing" the mail I attempted to load the game into my Color Computer and it traveled from one end of the tape to the other and never found it. Fearing that I had inadvertently received a blank tape, I listened to it and heard the unmistakeable sounds of Model I Level II Basic. Being a software vendor I'm aware that it is possible to ship a customer the wrong version of a program so I loaded the game into my Model I so that I could at least see how the game fared. My five year old son, Aaron, is the offical game tester and the test is this: the game is loaded and the instructions are read to him and we then time how long it takes for him to lose interest. He placed his bet on the number of the frog he wanted to race and the screen cleared. A square was drawn on the screen and several numbers appeared within it. After a short delay the screen started a rapid clear and redraw sequence with the numbers in new positions. The game ended when one of the numbers reached the edge of the screen. Total elapsed time was 27 seconds. Aaron had already left the room. I asked him if he wanted to play another game, he didn't. In order to be fair I played the game, I was impressed with Aaron's stamina.

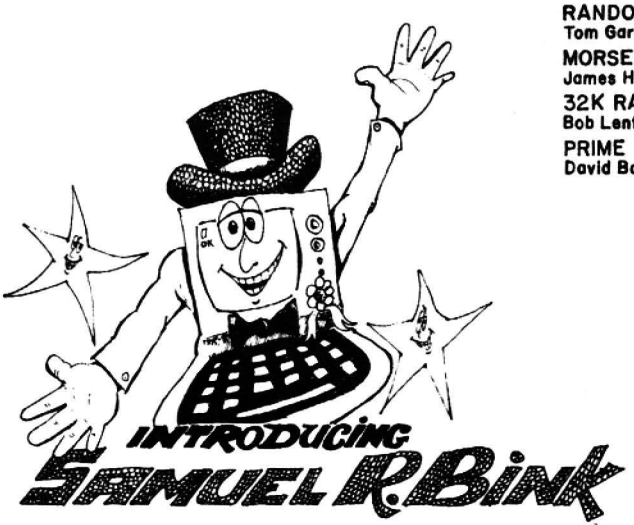
The game is a real turkey and I'm fairly well convinced that the Color Computer version is just a translation of the same thing, since Erickson had versions for the Pet as well.

The Software Stockpile
9437 Ironwood
Des Plaines, IL 60016

$2.50

Color Computer News

JULY/AUGUST 1981
Volume 1 Number 2



Color Computer News is published bi-monthly by REMarkable Software.

OUR FEARLESS CREW!

REMarks

I found that I was unintentionally given two pieces of bad advice about the last issue. The first was from a postal employee about not sending the magazine bulk rate. If you'd like to see how bad that advice was look at the stamp on the back of last issue, 52 cents each!! Multiply that by over 1000 and you'll see why my knees got weak. I changed printers at the last minute due to a quality and delivery problem. I found some typos too late to change them. I reread the survey and it sounded like I was going to throw away your surveys without reading them. The index had two typos alone. How can anyone misspell "how"? Most of all I worried about not getting enough articles to fill up this issue. Even with all of the problems I was proud of the first issue and with your help we'll get better as time goes on.

In order to keep the small amount of hair that I have left I'm making the following changes.

First, the magazine will be mailed bulk rate. If you need first class delivery it's an extra $ .50 per issue to cover the higher rate and the extra handling.

Second, White Enterprises will be doing the master copy. White Enterprises is remotely affiliated with REMarkable Software, and sells some excellent educational and business software for all of the TRS-80s. You can contact White Enterprises by writing to: White Enterprises, 432 Rutledge, Pentwater MI 49449. The reason another software house is doing the typesetting is that their printer can justify proportionally. That is, if you look at the articles in the last issue, the right margin was made even by adding spaces between words, proportional justification adds tiny spaces between letters. The difference is dramatic. Another thing about proportional justification is that it will allow more letters per line. This will perhaps make the magazine look smaller when it isn't. We tried a few pages and the difference is about 6 lines per page, extended over the entire magazine that could amount to a page or two each issue.

Third, deadlines will be enforced for everyone. The magazine gets mailed on the 24th of the month preceeding the cover date. Therfore deadline is one month prior to the publication date. This is much shorter than most magazines. But I think we can live with this for a long time.

What do you think about going monthly? This would mean not only more regularity but the disadvantage of a smaller magazine. I'm prepared to go to press monthly but does greater frequency outweigh the disadvantages of a smaller magazine and higher cost per year? You have to decide. If we change to monthly it also means that I have to depend on you for more articles, but the quality must stay the same. I'll judge by the number of letters in favor plus the number of articles recieved.

REMarks

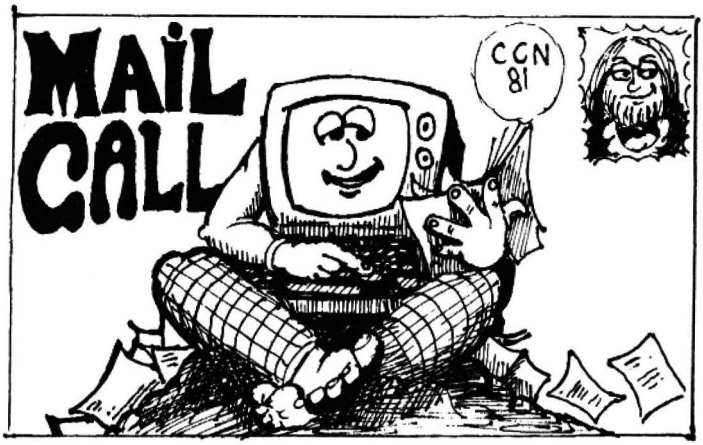
The classified ad section is still unused, perhaps because I haven't pushed it enough. The prices are $5.00 per half inch for personal listings and $15.00 for business listings. This is a split page so judge your listing accordingly.

You should be aware that, because the way the Color Computer hardware is configured, piggy-backing RAM chips to upgrade to 32K is incompatable with graphics. The Color Computer will not move the video display into the new bank without additional modification. The Micro Works has contributed an article about how to alleviate this problem for those of you that learned the hard way or feel you need 32K. But, please read the article carefully before you start cutting traces.

I just saw Radio Shack's new CC manual. The thing is pretty good. They actually show you how to get into hi-res, use the cassette and all of the other "stuff" that the earlier manual left out. But I was really quite upset by the section about hi-res, they take great pains to tell you that it's for "technical types" and that if you make a "typo" you may have to reset the computer. First, I'm not a "technical type" but I couldn't find anything there that I was afraid of. "You may have to reset the computer if you make an error", so what. It's not like the computer will blow up. If I have to reset my computer a million times to learn something new then I'll reset it a million times. Nothing irritates me more than to hear someone imply that anything is to too hard for someone else to learn. I guess the clincher was the last paragraph on the first page where, after warning us about how hard all of this Machine language and direct memory addressing is, states, "Still with us? O.K. now that we've warned you....". Further more, to use hi-res you don't need any machine language just a few POKEs from Basic. Direct memory addressing(?) I guess they mean that the POKE must address a memory location directly (I didn't know it could work any other way). Still there's enough there that you should get one if you can. The service manual is also available now for $8.00. Your local Radio store may think that this is a controled circulation item, if they do point out that it is in their latest catalog.

Reader service cards are really good things if you keep in mind the way magazines deal with them. I've gotten quite a few calls from people wanting to know why I haven't answered their request from the 80 Microcomputing reader service card. The reason is that I haven't got them yet. The usual routine for "bingo cards" is the publisher waits for all of the cards to come in (at least 4 weeks after the publication date of the magazine), they are then entered into their (or a service bureau's) computer and sorted by advertiser. Then they print sheets of mailing labels for all the advertisers, stuff them in envelopes and mail them out bulk rate. Total elapsed time is 6 to 10 weeks from the date you mailed it in depending on which magazine and the volume of responses. We then must enter the names into our computer and sort out all of the duplicates, printout the new names, stuff all the envelopes and mail our reply. Our total time for reader service cards is about 2 days. Compare this with the people that wrote directly, they got the magazine before I got the reader service cards. I like reader service cards and will continue to use them, but if you need or want a fast reply you have to write, not just to us but everyone. One last item on response, many of you didn't get personal replys to your subscriptions and I'm really sorry but the last two weeks before it goes to the printer are busy.

I was amazed at the response to the survey by both the number of responses and the answers. I'm shocked at the number of people with Extended Basic, I really thought that they would be the minority. I was glad to see that everyone thought we are off to a good start. The format has changed a little due to your responses. The change to two columns per page was mixed but most everyone wanted the margin to stay wide so they could put the magazine in a three ring binder.



HELP!
I'm looking for a POKE statement that QUICKLY turns on the cassette motor (MOTOR ON has a delay too long for my critical program). Have you found one yet?
Kenneth Armstrong
Chicago

Try POKE 65313,4 to turn it on and POKE 65313,52 to turn it off.

Can text from RAM be saved to tape from CompuServe?
Bruce Gustafson
Roscoe, IL

Perhaps you may be interested in some of the things I have found while wandering around the CC. Many of these things appeared in a letter to the editor which I wrote to Creative Computing and appeared in the March issue. All addresses are given in decimal. The current cursor position is stored in locations 136 and 137. The contents of these locations range from 1024 to 1536, which corresponds to the range of "normal" alpha video RAM addresses. Location 282 controls lower case/reverse video alpha mode. When this location is non-zero, one gets lower case (reserve video). It is normally set at 255 ($FF) and when one hits SHIFT 0, it gets negated, thus changing it to zero. Since I frequently hit SHIFT 0 when I mean to hit SHIFT 9 for close parenthesis, I POKE a non-zero value into 282 (other than 255) which effectively disables SHIFT 0. The line number currently being executed while a program is running is stored at locations 104, 105. I haven't found a great deal of use for that one, but there it is! (I feel it will become more usefull when I get some more machine language programs going).
Alexander Frazer Jr
Fort Lauderdale, FL

I have seen references to 32K RAM upgrades, which I cannot do myself due to bad eyesight. Do you or do you know of someone or a company that will do it for me.
Donn Jones
Canal Winchester, OH

Can anyone lend a hand?

```
0 CLEAR 500
10 CLS:L=0
20 READ AA$
25 IF LEN(AA$)/32+L>12 THEN FOR I=0 TO 1000:NEXT:L=0:CLS
30 IF AA$="END" THEN 9999
40 IF LEN(AA$)<32 THENPRINTAA$:L=L+1
50 IF LEN(AA$)=32 THEN PRINTAA$;
60 IF LEN(AA$)<33 THEN 100
70 FOR I=32 TO 1 STEP -1
80 IF MID$(AA$,I,1)<>" " THEN 90
85 PRINTLEFT$(AA$,I);:IF I<32 THEN PRINT
86 AA$=MID$(AA$,I+1)
87 GOTO 40
90 NEXT I
100 IF L<12 THEN 20
110 FOR I=0 TO 1000:NEXT
120 GOTO 10
200 DATA"REMarkable Software"
210 DATA"P.O. BOX 1192"
220 DATA"MUSKEGON, MI 49443"
230 DATA" "
240 DATA" "
250 DATA"DEAR BILL,"
260 DATA" "
270 DATA" "
280 DATA"HAVING READ YOUR FIRST ISSUE AND SENT IN MY SUBSCRIPTION
AND SURVEY, I WOULD ALSO LIKE TO CONTRIBUTE A PROGRAM TO THE USER'S
GROUP."
290 DATA"FOLLOWING THIS LETTER IS A GAME CALLED BLOCK THAT WILL RUN
IN 4K. I HOPE THAT THE CCN AND THE USER'S GROUP CONTINUES TO GROW
AND PROSPER."
310 DATA"            SINCERELY,"
330 DATA"            GREG R ESTEP"
350 DATA"            CORTE MADERA"
360 DATA"            CA    94925"
9990 DATA "END"
```

Appending (merging) programs from tape is accomplished by changing the values at addresses 25 and 26 to point to an address behind the Basic program in RAM. The new address should be the beginning of variable storage as indicated by the pointer at 27 and 28. After loading the second program the pointer at 25 and 26 must be restored to the start of Basic at 1531. The second program must have higher line numbers than the first. To merge the two programs type in the following basic statements in command mode (no line numbers):

```
1 CLOAD"first program"
2 PRINT PEEK (28)
3 POKE 25, PEEK(27)
4 POKE 26, PEEK(28) - 2
5 CLOAD"second program"
6 POKE 25, 6
7 POKE 26, 1
```

If PEEK(28) yields a value < 2 then steps 3 and 4 would be:

```
3 POKE 25, PEEK(27) - 1
4 POKE 26, PEEK(28) + 254
```

Robert Huxter
Media, PA

Color Computer Hi-Res Graphics

By Tom Rosenbaum

One of the persistent mysteries of the Color Computer is the use of the high resolution graphics modes. As Extended Color Basic becomes more prevalent, more people will be exposed to the capabilities of high resolution but will be frustrated by the slowness of it. Virtually the only way to get around this limitation is to write programs in machine language, but in order to do that, one must learn how the graphics modes operate.

The display of the Color Computer may be tailored to any one of a number of different display modes. These modes are controlled by the MC6847 Video Display Generator (VDG). A summary of the VDG modes is contained in Table 1.

In order to access the various display modes of the Color Computer, it is necessary to program both the VDG and the Synchronous Address Multiplexer (SAM). The SAM provides data from the Random Access Memory (RAM) for the VDG to process and display. If the VDG and the SAM are not put into the same mode, a garbage display will be the result. This is because the VDG will be trying to process data for a particular display mode but the SAM will be providing data for a different display mode.

Before attempting to use the various display modes of the Color Computer, one must become aware of the manner in which the display information is stored in the computer. All of the video display data is memory mapped. Simply put, it means that the information seen on the display is stored in the memory of the computer as opposed to having its own special memory just for the video display. The normal display for BASIC programs requires 512 bytes and is located from 1024 to 1536. Try POKEing data into those addresses and see what happens. In most black and white display systems, such as the TRS 80 Model III, one bit in the display memory corresponds to one pixel (display element). If the bit is a one, the pixel is on (white); if the bit is a zero, the pixel is off (black). Some of the Color Computer modes use only two colors and store video display data as decribed above. The other modes use at least four colors - therefore, at least one pair of bits must be used to indicate the color of a pixel. All of the four color graphics modes divide a byte into four pairs of bits and each pair of bits defines the color for that pixel.

The 6847 VDG supports three basically different types of display modes:

1. ALPHANUMERIC
2. SEMIGRAPHIC
3. GRAPHIC

The alphanumeric mode allows alphanumeric characcters to be displayed. Lowercase is displayed in inverse video - this is a built - in hardware feature of the 6847. The semigraphic modes partition the display area into a group of display blocks. Each block is subdivided into a number of graphic sub-blocks. Only one color may be specified for each block - all of the sub-blocks must be the same color as the color of the block. Each sub-block may be on (the color of the block) or off (black). The semi-4, semi-6 modes are coarse resolution which will generally not be used in programs. Semi-4 is used in BASICs SET and RESET functions (not PSET and PRESET). Semi-8, 12 and 24 use a complicated adressing method and are inefficient - 38 0/0 of the memory used in the video display is wasted. For these reasons you will generally not find much use for the semigraphic modes. Their only redeeming feature is that they allow all eight colors to be displayed at the same time.

The graphics modes (1C, 1R, 2C, 2R, 3C, 3R, 6C, 6R) are the most powerful and will be used by most high resolution programs. The 1R, 2R, 3R and 6R modes are basically black and white modes in which one bit of a display byte turns a pixel on (foreground color) or off (black). The 1C, 2C, 3C and 4C modes are color modes which each display byte is divided into four pairs of bits. Each pair of bits specifies one of four colors for the pixel it controls. The price which you pay for using the higher resolution graphics modes is having to use more memory for the video display. Table 2 summarizes the graphics display modes.

Color Computer Hi-Res Graphics
By Tom Rosenbaum

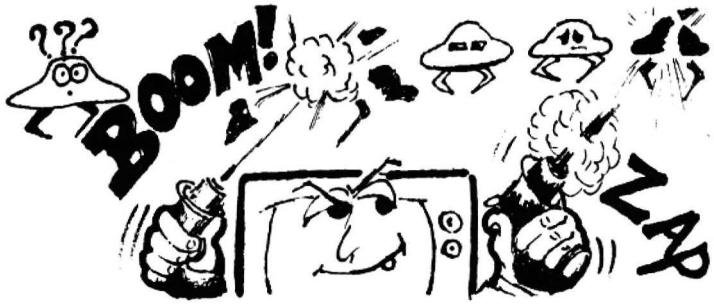
In order to better understand how to use the high resolution graphics modes, a sample program will be written. This program will use the highest resolution color graphics mode, 6C. This mode requires 6K of RAM and the video display will begin at $400 (The dollar sign indicates that the number following is in hexadecimal format). This program will draw an invader from a Space Invaders program on the screen and move it around under the control of the right joystick. The program goes into a continuous loop in which it samples the joystick data, erases the old invader and draws a new invader at the new joystick horizontal and vertical coordinates. The invader is only allowed to occupy one of 32 horizontal and one of 64 vertical positions. These constraints are observed merely for ease of programming. The invader "shape" is a block of 16 bytes 2 wide by 8 deep - putting different values into this block of data will change the shape of the invader.

Typing "X" will escape the program and return to BASIC.

TABLE 1. MODE SELECT

MC6883 and VDG MODE REGISTERS			PIA REGISTER BITS HEX ADDRESS (FF22)								DATA BITS		ALPHA/GRAPHIC MODE SELECT
V_2	V_1	V_0	7	6	5	4	3	2	1	0	7	6	
0	0	0	0	X	X	0	CSS	N	N	N	0	0	Alphanumerics
0	0	0	0	X	X	0	CSS	N	N	N	0	1	Alphanumerics Inverted
0	0	0	0	X	X	0	X	N	N	N	1	X	Semigraphics - 4
0	0	0	0	X	X	1	CSS	N	N	N	X	X	Semigraphics - 6
0	1	0	0	X	X	0	X	N	N	N	1	X	Semigraphics - 8
1	0	0	0	X	X	0	X	N	N	N	1	X	Semigraphics - 12
1	1	0	0	X	X	0	X	N	N	N	1	X	Semigraphics - 24
0	0	1	1	0	0	0	CSS	N	N	N	X	X	64 x 64 Color Graphics
0	0	1	1	0	0	1	CSS	N	N	N	X	X	128 x 64 Graphics
0	1	0	1	0	1	0	CSS	N	N	N	X	X	128 x 64 Color Graphics
0	1	1	1	0	1	1	CSS	N	N	N	X	X	128 x 96 Graphics
1	0	0	1	1	0	0	CSS	N	N	N	X	X	128 x 96 Color Graphics
1	0	1	1	1	0	1	CSS	N	N	N	X	X	128 x 192 Graphics
1	1	0	1	1	1	0	CSS	N	N	N	X	X	128 x 192 Color Graphics
1	1	0	1	1	1	1	CSS	N	N	N	X	X	256 x 192 Graphics

X DON'T CARE
N DO NOT CHANGE



COLOR SET	DATA BIT S	Color			Resolution		Data Byte(s)	Comments
		Character Color	Background	Border	Columns x Rows	Detail		
0	0	Green	Black	Black	32 x 16	8 dots — 1 … 7; 12 dots — 1 … 5	0 ASCII code	The *Alphanumeric Internal* mode uses an internal character generator which contains the following five dot by seven dot characters, to A B C D E F G H I J K L M N O P Q R S T U V W X Y Z ! / ( ) + — . SP 1 " # $ % & ' * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = ?
	1	Black	Green					
1	0	Orange	Black	Black				
	1	Black	Orange					
X	X	L2 C2 C1 C0 Color 0 X X X Black 1 0 0 0 Green 1 0 0 1 Yellow 1 0 1 0 Blue 1 0 1 1 Red 1 1 0 0 Buff 1 1 0 1 Cyan 1 1 1 0 Magenta 1 1 1 1 Orange		Black	64 x 32	L_1 L_2 one element; L_3 L_4	1 C_0 C_1 C_2 L_3 L_4 L_1 L_2	The *Semigraphics-4* mode uses an internal "coarse graphics" generator in which a rectangle (eight dots by 12 dots) is divided into four equal parts. The luminance of each part is determined by a corresponding bit on the VDG data bus. The color of illuminated parts is determined by three bits. It requires 512 bytes of display memory.
0	X	L2 C1 C0 Color 0 X X X Black 1 0 0 Green 1 0 1 Green 1 1 0 Blue 1 1 1 Red		Black	64 x 48	L_1 L_2; L_3 L_4; L_5 L_6	C_0 C_1 L_1 L_2 L_3 L_4 L_5 L_6	The *Semigraphics-6* mode is similar to the *Semigraphics-4* mode with the following differences: The eight dot by twelve dot rectangle is divided into six equal parts. Color is determined by the two remaining bits. It requires 512 bytes of display memory.
1		0 X X X Black 1 0 0 Buff 1 0 1 Buff 1 1 0 Magenta 1 1 1 Orange						
X	X	L2 C2 C1 C0 Color 0 X X X Black 1 0 0 0 Green 1 0 0 1 Yellow 1 0 1 0 Blue 1 0 1 1 Red 1 1 0 0 Buff 1 1 0 1 Cyan 1 1 1 0 Magenta 1 1 1 1 Orange		Black	64 x 64	L_1 L_2; L_3 L_4; L_5 L_6; L_7 L_8	1 C_0 C_1 C_2 L_1 L_2 X X 1 C_0 C_1 C_2 L_3 L_4 X X 1 C_0 C_1 C_2 L_5 L_6 X X 1 C_0 C_1 C_2 L_7 L_8 X X	The *Semigraphics-8* mode requires four column consecutive addresses,* and produces a 2x4 block. It requires 2048 bytes of display memory.
X	X	L2 C2 C1 C0 Color 0 X X X Black 1 0 0 0 Green 1 0 0 1 Yellow 1 0 1 0 Blue 1 0 1 1 Red 1 1 0 0 Buff 1 1 0 1 Cyan 1 1 1 0 Magenta 1 1 1 1 Orange		Black	64 x 96	L_1 L_2; L_3 L_4; L_5 L_6; L_7 L_8; L_9 L_{10}; L_{11} L_{12}	1 C_0 C_1 C_2 L_1 L_2 X X 1 C_0 C_1 C_2 L_3 L_4 X X 1 C_0 C_1 C_2 L_5 L_6 X X 1 C_0 C_1 C_2 X X L_9 L_{10} 1 C_0 C_1 C_2 X X L_{11} L_{12}	The *Semigraphics-12* mode requires six column consecutive addresses,* and produces a 2x6 block. It requires 3072 bytes of display memory.
X	X	L2 C2 C1 C0 Color 0 X X X Black 1 0 0 0 Green 1 0 0 1 Yellow 1 0 1 0 Blue 1 0 1 1 Red 1 1 0 0 Buff 1 1 0 1 Cyan 1 1 1 0 Magenta 1 1 1 1 Orange		Black	64 x 192	L_1 L_2; L_3 L_4; L_5 L_6; L_7 L_8; L_9 L_{10}; L_{11} L_{12}; L_{13} L_{14}; L_{15} L_{16}; L_{17} L_{18}; L_{19} L_{20}; L_{21} L_{22}; L_{23} L_{24}	1 C_0 C_1 C_2 L_1 L_2 X X 1 C_0 C_1 C_2 L_3 L_4 X X 1 C_0 C_1 C_2 L_5 L_6 X X 1 C_0 C_1 C_2 L_7 L_8 X X 1 C_0 C_1 C_2 L_9 L_{10} X X 1 C_0 C_1 C_2 L_{11} L_{12} X X 1 C_0 C_1 C_2 X X L_{13} L_{14} 1 C_0 C_1 C_2 X X L_{15} L_{16} 1 C_0 C_1 C_2 X X L_{17} L_{18} 1 C_0 C_1 C_2 X X L_{19} L_{20} 1 C_0 C_1 C_2 X X L_{21} L_{22}	The *Semigraphics-24* mode requires twelve column consecutive addresses,* and produces a 2x12 block. It requires 6144 bytes of display memory.
0	X	C1 C0 Color 0 0 Green 0 1 Yellow 1 0 Blue 1 1 Red		Green	64 x 64	E_1 E_2 E_3 E_4	C_0 C_1 C_2 C_3 C_4 C_5 C_6	The *Graphics-1C* mode uses 1024 bytes of display RAM in which one pair of bits specifies one picture element.
1		0 0 Buff 0 1 Cyan 1 0 Magenta 1 1 Orange		Buff				
0	X	L2 Color 0 Black 1 Green		Green	128 x 64	L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8	L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8	The *Graphics-1R* mode uses 1024 bytes of display RAM in which one bit specifies one picture element.
1		0 Black 1 Buff		Buff				
0	X	Same colors as Graphics one C		Green	128 x 64	E_1 E_2 E_3 E_4	C_0 C_1 C_2 C_3 C_4 C_5 C_6	The *Graphics-2C* mode uses 2048 bytes of display RAM in which one pair of bits specifies one picture element.
1				Buff				
0	X	Same colors as Graphics one R		Green	128 x 96	L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8	L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8	The *Graphics-3R* mode uses 1536 bytes of display RAM in which one bit specifies one picture element.
1				Buff				
0	X	Same colors as Graphics one C		Green	128 x 96	E_1 E_2 E_3 E_4	C_0 C_1 C_2 C_3 C_4 C_5 C_6	The *Graphics-3C* mode uses 3072 bytes of display RAM in which one pair of bits specifies one picture element.
1				Buff				
0	X	Same colors as Graphics one R		Green	128 x 192	L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8	L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8	The *Graphics-3R* mode uses 3072 bytes of display RAM in which one bit specifies one picture element.
1				Buff				
0	X	Same colors as Graphics one C		Green	128 x 192	E_1 E_2 E_3 E_4	C_0 C_1 C_2 C_3 C_4 C_5 C_6	The *Graphics-6C* mode uses 6144 bytes of display RAM in which one pair of bits specifies one picture element.
1				Buff				
0	X	Same colors as Graphics one R		Green	256 x 192	L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8	L_1 L_2 L_3 L_4 L_5 L_6 L_7 L_8	The *Graphics-6R* mode uses 6144 bytes of display RAM in which one bit specifies one picture element.
1				Buff				

```
                        **SAMPLE PROGRAM TO DEMONSTRATE HIGH
                        **RESOLUTION GRAPHICS

                A00A    JOYST   EQU     $A00A
                A000    KEY     EQU     $A000       KEYBOARD DRIVER ENTRY
                0400    VIDRAM  EQU     $400        TOP OF VIDEO RAM

1C00                            ORG     VIDRAM+$1800 PUT PROGRAM ON TOP OF VIDEO
                        *                  DISPLAY

                        **SET UP 6C GRAPHICS MODE

1C00 86   E0                    LDA     #$E0
1C02 B7   FF22                  STA     $FF22       PROGRAM THE VDG (6847)
1C05 B7   FFC3                  STA     $FFC3
1C08 B7   FFC5                  STA     $FFC5       PROGRAM THE SAM (74LS783)

                        **CLEAR THE SCREEN

1C0B 4F                         CLRA
1C0C 5F                         CLRB
1C0D 8E   0400                  LDX     #VIDRAM     START OF VIDEO SCREEN
1C10 ED   81            LOOP0   STD     ,X++        CLEAR TWO BYTES
1C12 8C   1C00                  CMPX    #VIDRAM+$1800 ARE WE AT BOTTOM OF SCREEN
1C15 25   F9                    BLO     LOOP0       NO

                        **$200 & $201 ARE TEMP STORAGE LOCATIONS FOR
                        **THE HORIZONTAL AND VERTICAL COORDINATES OF
                        **THE INVADER

1C17 F7   0200                  STB     $200        START THE INVADER AT THE UPPER
1C1A F7   0201                  STB     $201        HAND CORNER OF THE SCREEN

                        **MAIN PROGRAM WHICH MOVES THE INVADER

1C1D AD   9F A000       MAIN    JSR     [KEY]       CHECK FOR THE "X" KEY
1C21 81   58                    CMPA    #'X         IS IT?
1C23 26   03                    BNE     LOOP1       NO
1C25 7E   A027                  JMP     $A027       YES; GO BACK TO BASIC
1C28 AD   9F A00A       LOOP1   JSR     [JOYST]     GET THE JOYSTICK DATA
                        *HORIZONTAL DATA TO $15A
                        *VERTICAL DATA TO $15B

                        **THIS CODE WILL ERASE THE INVADER

1C2C 8D   30                    BSR     CALCAD      GET SCREEN ADDR OF INVADER
1C2E 86   08                    LDA     #8          INVADER OCCUPIES 8 VERT ROWS
1C30 6F   80            LOOP2   CLR     ,X+         CLEAR THE FIRST ROW
1C32 6F   84                    CLR     ,X          OF THE INVADER
1C34 30   88 1F                 LEAX    31,X        MOVE TO THE NEXT ROW
1C37 4A                         DECA                CHECKED ALL 8 ROWS?
1C38 26   F6                    BNE     LOOP2       NO

                        **GET THE NEW INVADER ADDR AND STORE IT
                        **IN THE TEMP STORAGE LOCATION

1C3A B6   015A                  LDA     $15A        GET NEW HORIZONTAL ADDR
1C3D B7   0200                  STA     $200        STORE IT
1C40 B6   015B                  LDA     $15B        GET NEW VERTICAL ADDR
1C43 B7   0201                  STA     $201        STORE IT
1C46 8D   16                    BSR     CALCAD      GET ABSOLUTE SCREEN ADDR
```

```
                        *           OF THE NEW INVADER ADDR
1C48 33   8D 002C       LEAU    TABLE,PCR GET ADDR OF INVADER DATA
1C4C 86   08            LDA     #8        8 INVADER ROWS TO MOVE
1C4E E6   C0      LOOP3 LDB     ,U+       GET ONE BYTE OF INVADER
1C50 E7   80            STB     ,X+       DISPLAY IT
1C52 E6   C0            LDB     ,U+       GET ANOTHER BYTE OF INVADER
1C54 E7   84            STB     ,X        DISPLAY IT
1C56 30   88 1F         LEAX    31,X      MOVE TO NEXT ROW
1C59 4A                 DECA              MOVED ALL 8?
1C5A 26   F2            BNE     LOOP3     NO
1C5C 20   BF            BRA     MAIN      GO BACK TO MAIN LOOP


                        **THIS ROUTINE WILL TAKE AN X-COORDINATE (0-63)
                        **STORED IN $200 AND A Y-COORDINATE (0-63)
                        **STORED IN $201 AND CONVERT THEM INTO AN
                        **ABSOLUTE SCREEN ADDRESS FOR THE INVADER

1C5E B6   0201    CALCAD LDA    $201      GET VERTICAL ADDR
1C61 81   3D            CMPA    #63-2     ROW 61 IS THE LAST ROW THE INVADER
                        *           MAY OCCUPY WITHOUT HAVING PART OF
                        *           IT EXTEND INTO RAM ABOVE THE VIDEO
                        *           DISPLAY AREA
1C63 25   02            BLO     LOOP10    INVADER IS NOT AT BOTTOM
1C65 86   3D            LDA     #61       HERE IT IS AT THE BOTTOM
1C67 C6   60      LOOP10 LDB    #96       96 BYTES FOR EACH VERTICAL
                        *         INVADER POSTION
1C69 3D                 MUL               GET VERTICAL OFFSET OF INVADER
1C6A 1F   02            TFR     D,Y       STORE IT IN Y
1C6C F6   0200          LDB     $200      GET HORIZONTAL ADDR OF INVADER
1C6F 54                 LSRB              DIVIDE BY TWO - THE INVADER CAN ONLY BE
                        *       IN ONE OF 32 HORIZONTAL POSITIONS
1C70 4F                 CLRA              CLEAR HIGH ORDER BYTE OF D REG
1C71 30   AB            LEAX    D,Y       ADD HOR AND VER OFFSETS AND PUT 'EM IN X
1C73 30   89 0400       LEAX    VIDRAM,X  ADD IN THE START OF VIDEO DISPLAY
1C77 39                 RTS


                        **THIS TABLE DEFINES THE SHAPE OF THE INVADER

1C78 02 00 0A 80  TABLE FCB     2,0,$A,$80
1C7C 2A A0 A6 68        FCB     $2A,$A0,$A6,$68
1C80 2A A0 30 30        FCB     $2A,$A0,$30,$30
1C84 C0 0C 30 30        FCB     $C0,$C,$30,$30

                        END
```

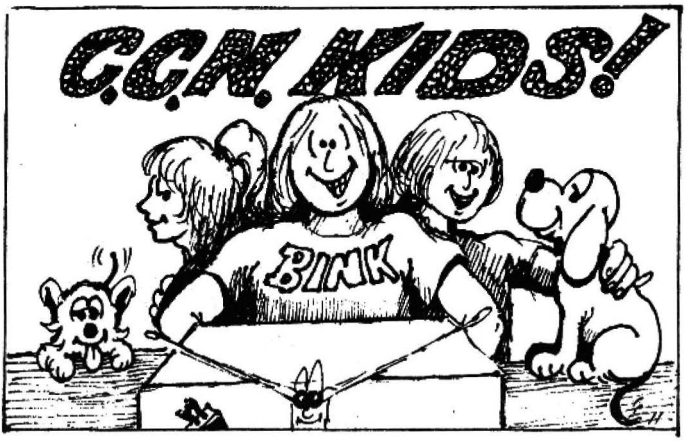
Mary, Lisa, Kathy and Sue...

Where are you? We have had many fine articles and programs sent to us. Thanks for the terrific response! One thing does concern us, however. We have not had anything submitted by women. Why not? We need you. WE know there are many of you out there with color computers. We also know that you are creating your own programs. Let our readers know, too!

We are always looking for good articles. But we want to hear from the women as well as the men. It would also encourage other women to "get into" computers if they knew what other women, like themselves, are doing.

So...let us know. We'll be waiting to hear from you.

Several people asked when I was going to run a mod for higher clock speed, I doubt that I ever will. To make the Color Computer run faster just type POKE 65495,0 and to slow it back down POKE 65494,0. DON'T use high speed during I/O.



Here it is, the very first KID'S PAGE and it's all for you!! This page will be written by and for kids age 12 and under. We need your questions, comments, stories, drawings, programs and what-evers. The only exception to the age rule will be the editor, who is an ancient 29 year old and the publisher, who is extremely young and 27 (they're both a couple of kids anyway).

To get started we have a program written by a five year old girl (now seven) and two short ones by one of the old kids.

We hope you like it and will start sending us some really neat things you have done. Please send your questions and drawings, too. We want this page to be just what YOU want it to be. Next issue's best program will recieve a SAM Coloring Book.

```
5 REM BY HEATHER SIAS
10 CLS
20 A$=" THE DOG CHASED THE CAT"
30 FOR X=0 TO 450
40 PRINT@X,A$
50 NEXT X
60 GOTO 30
```

```
10 INPUT"WHAT IS YOUR NAME";A$
20 FOR X=LEN(A$) TO 1 STEP -1
30 B$=B$+MID$(A$,X,1)
40 NEXT X
50 PRINT"HELLO ";B$
```

6809 Machine Code

Last issue we looked at addressing and most of the opcodes. This month we'll back off a little and just build and/or look at some tools.

To write machine code you need an understanding of the MPU and a way to get the machine code into the machine. We'll learn more and more about the MPU as we go along but this month let's look at how to get the machine code in.

You could build a house with just a hammer, or fix your car with just a wrench, but it wouldn't be fun or efficient. The same is true of writing machine code. You could write the code with just the POKE statement in Basic, but I wouldn't care to. Common tools for machine language programming are: Editors, Assemblers, Monitors, Single Steppers, Disassemblers, and a few other odd ball programs. We'll look at the functions each of these perform and them write a tool or two.

Editor/Assembler is the most common tool. The editor is used to prepare what is called a Source program, which is our program written in mnemonics. The assembler translates the source program into an Object program, or code that the computer can execute directly. A monitor is a program that allows you to examine memory and change it. A good monitor also allows you to read and write tapes or disk files containing a machine language program. A disassembler is the opposite of an assembler in that it changes Object code into Source code. Single Steppers are programs that cause the machine code program to run very slowly so that you can see exactly what is happening. Good single steppers disassemble as they step.

That was fast. With the exception of the Editor and the Assembler, all of the programs memtioned above are "debugging" tools. If I had to choose one tool out of all of the above I guess I would choose the Monitor, so I guess we can call the Monitor a wrench.

The biggest problem with the Color Computer right now is the completely empty tool box. I don't wish to leave anyone out but as of right now I am aware of the following tools for the CC:

Monitors	The Micro Works
	Computer Ware
	Color Computer News
	Data Soft, Inc
Disassemblers	Soft Sector Marketing
	The Micro Works
	Data Soft, Inc
Editor/Assemblers	Computer Ware
	The Micro Works
	Data Soft, Inc
Single Steppers	Data Soft, Inc

(for the sake of accuracy, Data Soft produces a monitor called Sigmon that does all of the above, they are not separate programs)

If you have any experience with any of the above we could use a review of each of them. Before anyone gets upset with me, I know there must be others out there and if any of you vendors think I left you out let me know.

Believe it or not this issue's main project is written in Basic. It's an almost full featured Monitor. I'll give you the documentation for it and then the Code itself for you to type in. Don't knock it too much, I wrote it all one Saturday morning. All the commands require the name of the command, a starting address and an ending address. The commands are EDIT, ASCII DUMP, HEX DUMP, SAVE, LOAD, BREAKPOINT, and EXECUTE.

The EDIT command allows you to modify memory and enter a hand assembled machine language program. ASCII Dump shows the ASCII equivilent of each memory location within the specified range. HEX dump shows the hexidecimal value at each memory location. SAVE will allow you to save the machine language program on cassette tape, note that the monitor must be used to reload the program. LOAD is used to enter the programs saved by the SAVE command back into memory. BREAKPOINT sets or resets breakpoints (forces the program to stop execution and return to the monitor). EXECUTE allows you to run your machine language program with some degree of control. The last command is BASIC. BASIC writes a Basic program which will POKE your program into memory and execute it without needing the monitor in memory.

The program created by the BASIC command is useful for writing programs assisted by machine code. If you have Extended Basic you could change the BASIC subroutine to use the CLOADM command and delete the hex and decimal routines and replace them with &H and HEX$.

Here it is! CCN'S first tool and the first entry in the contributed software library.

```
10 REM
20 REM COLOR COMPUTER BASIC MONITOR
30 REM CCN JULY/AUGUST 1981
40 REM BY BILL SIAS
50 REM
60 HEX$="0123456789ABCDEF": CLS
70 PRINT"     CCN BASIC MONITOR": PRINT
100 INPUT"COMMAND";C$,BA$,EA$
102 H$=BA$: GOSUB 10000: BA=D: H$=EA$: GOSUB 10000: EA=D
110 IF LEFT$(C$,2)="ED" THEN 1000
120 IF LEFT$(C$,2)="AS" THEN 2000
130 IF LEFT$(C$,2)="HE" THEN 3000
140 IF LEFT$(C$,2)="SA" THEN 4000
150 IF LEFT$(C$,2)="LO" THEN 5000
160 IF LEFT$(C$,2)="BR" THEN 6000
170 IF LEFT$(C$,2)="EX" THEN 7000
180 IF LEFT$(C$,2)="BA" THEN 8000
200 PRINT"UNIDENTIFIED COMMAND": FOR TIME=1 TO 100: NEXT
205 GOTO 70
999 REM EDIT MODE
1000 CLS: FOR ADDR=BA TO EA
1010 D=PEEK(ADDR): GOSUB 12000
1020 PRINT A$;: INPUT H$: GOSUB 10000: BYTE=D
1030 POKE ADDR,BYTE
1040 NEXT ADDR
1050 GOTO 70
1999 REM ASCII DUMP
2000 CLS: FOR ADDR=BA TO EA STEP 15
2020 D=ADDR: GOSUB 11000: PRINT A$
2030 FOR OF=0 TO 14
2040 PRINT CHR$(PEEK(ADDR+OF));" ";
2050 NEXT OF: PRINT
2060 NEXT ADDR: GOTO 70
3000 CLS: FOR ADDR=BA TO EA STEP 4
3020 D=ADDR: GOSUB 11000: PRINT A$;
3030 FOR OF=0 TO 3
3035 D=PEEK(ADDR+OF)
3037 GOSUB 12000
3040 PRINT TAB((OF*5)+7)A$;
3050 NEXT OF
3055 PRINT
3060 NEXT ADDR
3070 FOR T=1 TO 200: NEXT: GOTO 70
3999 REM SAVE A MACHINE LANGUAGE TAPE (well sort of)
4000 CLS: INPUT"NAME";NA$
4010 OPEN"O",-1,NA$
4020 PRINT#-1,BA
4030 FOR ADDR=BA TO EA
4040 PRINT#-1,PEEK(ADDR)
4050 NEXT
4060 PRINT#-1,999
4070 CLOSE: GOTO 70
4999 REM LOAD A MACHINE LANGUAGE TAPE
```

```
5000 CLS: INPUT"NAME";NA$
5010 OPEN"I",-1,NA$
5020 INPUT#-1,BA
5030 INPUT#-1,BYTE
5035 IF BYTE=999 THEN CLOSE: GOTO 70
5040 POKE BA,BYTE
5050 BA=BA+1
5060 GOTO 5030
5999 REM SET OR RESET BREAKPOINT
6000 CLS: INPUT"SET OR RESET";Z$
6010 IF LEFT$(Z$,1)="R" THEN FOR OF=0 TO 2: POKE BP+OF,OLD(OF):
NEXT OF: GOTO 70
6020 INPUT"BREAKPOINT AT";H$: GOSUB 10000: BP=D
6030 FOR OF=0 TO 2: OLD(OF)=PEEK(BP+OF): NEXT OF
6035 POKE BP,14: POKE BP+1,180: POKE BP+2,244
6040 GOTO 70
6999 REM EXECUTE MACHINE LANGUAGE PROGRAM
7000 INPUT"ARGUMENT TO BE PASSED";AR
7005 INPUT"ENTRY POINT";H1$
7010 H$=H1$:GOSUB 10000
7020 DEFUSR(0)=D
7030 RV=USR(AR)
7035 PRINT"RETURNED VALUE =";RV
7040 GOTO 70
7999 REM WRITE BASIC TAPE
8000 CLS: INPUT"NAME";NA$
8002 OPEN"O",-1,NA$: A$="10 FOR X="+STR$(BA)+" TO "+STR$(EA)+":
READ A: POKE X,A: NEXT X"
8010 PRINT#-1,A$: LN=20
8020 FOR ADDR=BA TO EA STEP 10
8030 A$=STR$(LN)+" DATA"
8040 FOR OF=0 TO 9: A$=A$+STR$(PEEK(ADDR+OF)):IF OF<9 THEN A$=A$
+","
8045 NEXT OF
8050 PRINT#-1,A$: LN=LN+10: A$="": NEXT ADDR
8055 INPUT"ENTRY POINT";H$: GOSUB 10000: EP=D
8056 A$=STR$(LN+10)+" EXEC "+STR$(EP)
8057 PRINT#-1,A$
8060 CLOSE: GOTO 70
9900 DATA 1, 16, 256, 4096
9999 REM HEX TO DECIMAL SUBROUTINE
10000 D=0: RESTORE
10010 Z=LEN(H$)
10020 FOR K=Z TO 1 STEP -1
10030 READ M
10040 FOR J=1 TO 16
10050 IF MID$(H$,K,1)=MID$(HEX$,J,1) THEN Z=J-1: J=16
10060 NEXT J
10070 D=D+Z*M
10080 NEXT K
10090 RETURN
10999 REM DECIMAL TO HEX SUBROUTINE
11000 A$="": H4=INT(D/4096)
11010 H3=INT((D-H4*4096)/256)
11020 H2=INT((D-((H4*4096)+(H3*256)))/16)
11030 H1=D-((H4*4096)+(H3*256)+(H2*16))
11035 A$=MID$(HEX$,H4+1,1)+MID$(HEX$,H3+1,1)+MID$(HEX$,H2+1,1)+M
ID$(HEX$,H1+1,1)
```

```
11040 RETURN
11999 REM 2 DIGIT DECIMAL TO HEX
12000 A$="": H2=INT(D/16)
12010 H1=D-H2*16
12015 A$=MID$(HEX$,H2+1,1)+MID$(HEX$,H1+1,1)
12020 RETURN
```

If you have another computer you can use it there also, it doesn't care what MPU it operates on, so it should work with any Basic that has PEEK, POKE and MID$, with the exception of the BASIC command which will work only with disk or other ascii storage device (like CC's cassette) and the BREAKPOINT, the only change needed in breakpoint is to change the codes to whatever it is on the other machine you use. If you only have 4K you should remove all REM lines, shorten the variable names to 2 letters (in fact, everyone should, I just like flashy variables) and combine lines wherever possible. The 4 and 16K versions are available from the CCN library for $7.95, which also saves the chore of typing it in.

If the phone calls I've been getting are any indication I would say that the next thing to discuss is how you read or use the information given in an article about machine language programming. The explanation is both simple and complex. A good assembler produces a six column display, the first column contains addresses, the next contains op-codes, followed by the labels used by the programmer, next is the mnemonic column, then the operand column and last the comment column. If you have an assembler you would just type in the label, mnemonic, operand and the comment columns and let the assembler do the work. If you have a monitor you would edit memory at the locations shown in the address column to the values in the opcode column. Either method will put the machine language program into memory or on cassette tape. The next step is to read the comments and try to figure what the program is to do and last you need to debug the program. Debugging machine language programs will force you to learn more than any amount of reading or even programming on your own. The way to do this is to mentally divide the program into small pieces and test each of these pieces alone. This is done with the BREAKPOINT command in our Basic monitor or as detailed in the manual for the other monitors on the market. Set a breakpoint at the end point of the part of the program you are testing and execute the program, at the breakpoint the monitor will return control to you and you may modify the program as necessary by either correcting misentries or inserting new code as needed. As we progress and learn more about the 6809 all of these problems will fade. The disaster now is that not only must you learn a new "language" you also have to become a software mechanic and try to learn all of the new jargon all at once. We all went through it at one point or another and if you stick with it you'll master it all.

The next tool is perhaps the easiest to write, if you don't care about speed or efficiency. Disassemblers are handy programs and extremely simple to write in Basic, but very slow in operation. There are a lot of approaches to disassemblers. The fastest Basic version would be just a giant program full of IF/THEN statements. The slowest but more memory efficient would be a READ/DATA type of program, i.e. put all of the mnemonics in DATA statements in order and have a PEEK statement followed by a FOR/NEXT loop from 0 to the value that you PEEKed and READ one item with each execution of the loop. When you fall out of the loop you have the correct mnemonic, you then RESTORE and PEEK again. The fastest version would be to take the chart on the next few pages and figure out the algorithm used. I'm going to save my Basic Disassembler for next issue and let's have a contest. Who can write the fastest Basic Disassembler?

Cassette Files

by Richard A. White

The Basic commands for cassette file use are in the 4K Basic, but nowhere do the manuals tell you how to use them. These commands are the same as those in Model I and Model III disk basic for sequential file. It took me a while and more than a few bucks worth of books and magazines to figure this out. Cassette operation with previous machines has been a pain at best so interest in and published liturature on cassette file systems is minimal. The Color Computer saves and loads reliably at 1500 rather than unreliably at 500 baud as with Model I so using sequential filing techniques on tape may return. In addition the Color Computer is not particularly fussy about which tape you use. I have quite a few programs on Radio Shack's 3 for $1.99 C-30 cassettes, but you need to adjust the volume control when going from one tape type to another. Computer quality tape is not necessary.

The key to loading sequential files to cassette is to write your program so the numerical data or strings are in subscripted variables and then PRINT#1 these using a FOR-NEXT loop. The file is loaded back into the computer is the same way using INPUT#-1 or LINE INPUT#1 if you have Extended Basic.

Listing 1 is a file save and load subroutine taken from a program I rewrote from a magazine article. The basic program which had been written on an HP 3000, hardly a personal computer, needed considerable compression to fit the Color Computer, but worked exactly as it did on the big machine when I was done. Where has our toy gone? Anyway, in this program points are stored as their subscripted X and Y values and lines are identified by their start point L1(K) and end point L2(K). In the listing we start by entering whether we want to store the file "O" for out or load the file "I" for in. Line 1010 lets us set up the recorder at the proper place on the tape and set it to play or record. Finally, the file name is entered and the Color Computer does the rest using FOR/NEXT loops. The program is dimensioned to handle 100 points and 30 lines. If these numbers of points and lines are not defined, zeros are filed and loaded back. The end of file EOF is not really needed but is shown to illustrate its use. Line 1080 is a memory saving trick. Where you exit a FOR-TO loop early, it is good to set your count variable at or above the highest loop value and then use a NEXT. This cancels the count out of memory freeing space that otherwise would be held open looking for the count to continue.

The basic subroutine in Listing 1 can be adopted by changing the number and names of the variables and final values in the FOR-TO loops. The listing shows numerical variables, but string variables work as well.

Subroutine to Save or Load a Cassette File.

```
1000 INPUT"TO SAVE A FILE ENTER 0,
TO LOAD ENTER 1";E$
1010 INPUT"SET RECORDER TO RECORD
OR PLAY. PRESS ENTER ONCE FOR
MOTOR ON THEN AGAIN FOR MOTOR
OFF";I1: AUDIO ON: MOTOR ON: INPUT
I1: MOTOR OFF
1020 INPUT"ENTER FILE NAME 8
CHARACTERS OR LESS";NA$
1030 OPEN E$,-1,NA$: IF E$="I" THEN
1060
1040 FOR K=1 TO 100: PRINT#-1, X(K),
Y(K): NEXT
1050 FOR K=1 TO 30: PRINT#-1, L1(K),
L2(K): NEXT: GOTO 1090
1060 FOR K=1 TO 100: INPUT#-1, X(K),
Y(K): IF -1=EOF(-1) THEN 1080 ELSE
NEXT
1070 FOR K=1 TO 30: INPUT#-1, L1(K),
L2(K): IF -1=EOF(-1) THEN 1080 ELSE
NEXT
1080 K=100: NEXT
1090 CLOSE-1: GOTO XXXX
```

This permits operating the recorder to position tape.

This line saves data to tape.

This line also saves data to tape.

This line inputs the data from the tape. The EOF is probably not needed since we print the same number of statements as we input.

Close file. GOTO could be RETURN.

```
10 PMODE 3,1: SCREEN 1,1: PCLS
20 LINE(76,24)-(160,84),PSET,BF
30 LINE(160,84)-(180,116),PSET
40 LINE(76,84)-(53,116),PSET
50 LINE(60,124)-(176,124),PSET
60 LINE(53,116)-(60,124),PSET
70 LINE(180,116)-(176,124),PSET
80 LINE(80,88)-(152,88),PSET
90 LINE(68,116)-(164,116),PSET
100 LINE(80,88)-(68,116),PSET
110 LINE(152,88)-(164,116),PSET
120 FOR X=84 TO 152 STEP 8
130 FOR Y=92 TO 110 STEP 4
140 PSET(X,Y,2): NEXT Y,X
150 LINE(90,108)-(140,112),PSET,BF
160 CIRCLE(160,40),10
170 CIRCLE(77,40),10
180 LINE(67,41)-(50,64),PSET
190 LINE(70,82)-(50,64),PSET
200 LINE(74,50)-(64,64),PSET
210 LINE(76,76)-(64,64),PSET
220 CIRCLE(75,79),7
230 CIRCLE(71,86),3
240 LINE(170,40)-(182,62),PSET
250 LINE(182,62)-(166,80),PSET
260 LINE(162,54)-(168,62),PSET
270 LINE(168,62)-(160,72),PSET
280 CIRCLE(161,79),7
290 CIRCLE(165,86),3
300 LINE(80,28)-(156,80),PRESET,BF
310 CIRCLE (104,42),10
320 CIRCLE(120,40),10
330 CIRCLE(114,47),25,4,1,0,.40
340 CIRCLE(106,46),5
350 CIRCLE(122,44),5
354 CIRCLE(112,58),7,,1,.15,.50
356 LINE(108,57)-(112,48),PSET
360 PAINT(0,0),3,4
370 PAINT(74,52),3,4
380 PAINT(163,57),3,4
385 FOR X=1 TO 100
390 L$="L"+STR$(RND(255))
400 PLAY L$
420 CIRCLE(114,47),27,4,1,0,.30
430 PLAY CHR$(RND(7)+64)
440 CIRCLE(114,47),27,1,1,0,.30
450 NEXT
460 GOTO 385
```

Space Boredom
by Andrew Hubbel

"You are at the helm of a Starship. Imagine that your video screen is a window looking into outer space. Your laser sights are in the center of this "viewscreen". Try to destroy as many enemy ships as possible before colliding with them." So starts the introduction to Quasar Commander.

Actually, however, the game resembles a shooting gallery more than a starship. After selecting among several pages of options (skill level, time or shots limit, etc.) you begin the actual game. Your targets consist initially of a large number of moving dots which, if you "manuver" skillfully grow into three different types of targets! Scouts, which are fairly well behaved and vaguely resemble some Space Invaders, are the most numerous and are worth three points each when shot. (under some options, however, you must destroy them to get to the other targets.) Battle Cruisers, which resemble conventional aircraft, are worth 15 points apiece but are extremely hard to destroy. (each must fill nearly one-quarter of the entire screen before your shots will destroy it.) If you let any of the enemy ships collide with you (generally it will be a Battle Cruiser which refuses to die) you lose 10 points.

I have several criticisms of this game. First, it is essentially black and white. (Actually it has dark purple detail on a pale yellow background with colored instructions and red flashes when a ship is destroyed.) While this does not necessarily make it a bad game, this is, after all the "Color" computer.

Second, the joystick controls are difficult to use. The left joystick controls speed and has the firing button while the right joystick controls horizontal and vertical motion. These are analog controls - the null position is simply an arbitrary point and the slightest movement of the joystick alters the controlled function. There is no tacitle feedback and it is very difficult to co-ordinate both joysticks while watching the screen.

Another complaint is that the scoring system does not work as described in the instruction manual. When you reach 100 points your score is decreased by 100 and you are supposed to receive a bonus of 50 "minutes" or 15 shots depending on the limit option chosen. What actually happens is that your remaining time is set to 49 "minutes" or your remaining shots to 15, which, if you happen to be doing well, may actually be a penalty.

Perhaps the biggest criticism of this game, however, is that it does not realistically portray what is promised. There are no external reference points (i.e. stars) and no illusion of depth. Your gun sight never moves. Instead the entire background moves together. Some larger targets will occasionally move against the background, but it appears that one has lost control of their movement rather than that they are closer. All "distant" objects are the same size with no identifing characteristics or indication of relative distance. As they grow in size, changing shape and growing independantly, they still appear to be moving in only two dimensions rather than three. The result appears to be partially random, partially controlled shooting gallery, not a starship.

Quaser Commander seems to be grossly overpriced. A few fanatics may enjoy these games, but I suspect that most Color Computer owners, like me, would prefer something more realistic and entertaining, particularly if it is going to cost $30 - $40. I am certainly not going to waste any more money on Program Paks, at least until I actually see one that is worth playing (and paying for).

For those who get Extended Basic and find their 16K only prints 8487 on PRINT MEM, typing PCLEAR 1 will get 13095. The March issue of R/S TRS-80 Microcomputer News said we would have 14.5K. I spent an hour trying to ask R/S on the hot-line and never got through.
Ralph O.R. Schubert
Tulsa, OK

Random Thoughts
by Tom Garcia

There are three different Color Computers from Radio Shack. First, there is the basic (no pun intended) Plain Jane model with standard Color BASIC and 4K of RAM. At least it's called a 4K model. Enter a PRINT MEM command and you wil find that you only have 2,343 bytes to work with. Where did the rest of the memory go? The computer is using some of "your" RAM for it's internal work. Well, you can write a lot of programs with 2.3K bytes. As a beginner I sort of like the limited memory. I think that it is helping me to become a better programmer when I have to write programs in such a way as to conserve my limited amount of memory. I haven't run out yet so I guess I'm doing OK.

Computer number two is the extended basic model upgraded to 16K of RAM. It might cost you a hundred dollars or so to hire out the modification work but you can do it yourself for about a third of the price. Here is what I suggest that you do: Go to (or should that be GOTO?) your friendly Radio Shack store and buy a copy of Part #26-3001/3002. That is the TRS-80 Color Computer Service Manual. While you are there you might also consider picking up part number 276-1574 which is an IC insertion and removal tool kit. The manual costs $9.95 and the tool kit is $6.95. Now go home and read the manual. You are bound to learn something new about your computer and if you are at all mechanically inclined the parts list and the schematic are "must have" information. Your 16K upgrade project will involve the info on page 14 which is disassembly and reassembly dope, plus page 65 for printed circuit board depiction. Ready to give it a try? Obtain 8 RAM chips, type 4116, to replace the factory installed 4K chips. There are two jumpers that have to be moved from the factory installed 4K position to the immediately adjacent 16K pin. You can find these by looking very carefully at the above mentioned page 65. Check between U4 and U8 for one of them. The other one is just to the right of the 40 pin U10 SAM chip. After you have replaced the eight chips and moved the two jumpers you should have a 16K machine. Well, not quite 16K. The computer will still use almost a K and a half of RAM for it's own purposes, such as video display generation. Oh, the 4116 chips are available from many sources and should cost no more than $4 each. Don't let anyone sell you any jumpers. You would have needed them for the Model 1 but you won't need them for Color Computer modification.

Computer number three is thee 16K Extended Color BASIC model. That's the one that runs $200 more list price and you will have to decide (or probably already have if you are reading COLOR COMPUTER NEWS) if it's worth the extra money. From what I have read concerning Extended Color it is just not too exciting when I consider what I use a computer for. I use it for number manipulating, string handling, file keeping, and word processing. I don't play games or draw pictures. It may be that I don't know what I'm missing but only time will tell. I keep thinking that something else may come along that I would rather plug into the ROM expansion socket than the Radio Shack part. My local dealer wants over a hundred bucks for the part plus $25 to install it. Gee, sure would like to have the renumber feature that is part of Extended Basic. So far, the big disadvantage to my lack of the 16K Extended Color option is that I can't run my friends programs on my computer. They can run mine OK but they use commands that I can't execute when I try to load and run their programs. That, plus the future availability of commercial programs that will require the Extended Color will probably force me to upgrade at some point down the road.

How can I get Extended Basic for less than $99.00.
Mark Lockwood

First of all you really can't get the Extended Basic for $99.00. Radio Shack charges $17.50 to install it, which makes Extended Basic cost $116.50. But if you would like to get it for $99.00 you could try ordering part number 26-3018 from your local Radio Shack.

The Incredible Shrinking Program

by James Haan

About two years ago when I first got my Model 1 one of the first programs I wrote was a program to send morse code. The first version worked great and only used 14K of RAM. As an exercise in self disipline I decided that the next step was to make the program operate on a 4K Model 1. After several attempts I succeeded in making the program operate at the target memory size and amazingly enough you couldn't tell the difference between the two versions unless you listed them. In March of this year I got a Color Computer and in memory of the good old days I decided to rewrite the old program to run on the new machine. The first problem was to discover a method of turning the cassette on and off rapidly, the Z80 OUT command doesn't exist on the Color's 6809. After experimenting and reading, Bill and I found that memory location 65313 is connected to the cassette relay. Oddly enough only the numbers 4 and 52 will cause anything noticable to happen. 4 turns the cassette on and 52 turns it off as you can see by looking at lines 8 and 9 in the listing below. The first version ran great in a 16K machine and I again saw the challenge of self disipline before me. It seems that a 4K Color Computer really only has a little over 2K to operate with after scratchpad and overhead. This was going to be a bigger challenge than before due to the fact that the Model 1 version was compressed to 2476 bytes and now I had to compress the compressed version to run in 2400 bytes of memory. By examining the elements of Morse code you will see that many of the characters are combinations of other characters and by using this information it is possible to send characters that the program never defines. For example, since the letter T is just a single Dah (or dash to non-hams) I was able to leave the variable A undefined and jump to the subroutine that sends Dahs and the subroutine would return after a single execution. If you examine the code closely you will see other characters that use this techinque. One word of caution, the program uses very few spaces between commands so check your typing carefully before you decide that the listing is incorrect.The spacing that most listings contain were eliminated deliberataly to conserve memory and the program should be typed in exactly the same way. Be careful, debugging a program with no spaces can be difficult.

```
1 CLEAR70
2 CLS:R=0:PRINT"TO SEND CQ ENTER 1":PRINT"TO ANSWER CQ 2":
PRINT"TO CONT QSO 3":PRINT"FOR PRACTICE 4":INPUT"FOR QSO 5";Y
3 INPUT"HOW FAST";B:A=350/B:IFY=1THEN7
4 IFY=4THEN7
5 INPUT"HIS CALL SIGN";Z$:IFY=2THEN7
6 INPUT"HIS NAME";N$:INPUT"HIS RST";R$
7 ON Y GOSUB26,28,36,18,36
8 POKE65313,4:FORN=1TOA:NEXT:POKE65313,52:FORN=1TOA:NEXT:RETURN
9 POKE65313,4:FORN=1TOA*3:NEXT:POKE65313,52:FORN=1TOA:NEXT:RETURN
10 S=LEN(C$):FORT=1TOS:FORN=1TOA*2:NEXTN
11 F$=MID$(C$,T,1):PRINTF$;:IFF$=" "THEN15
12 U=ASC(F$):U=U-47:IFU<1THEN131
13 ON U GOSUB129,108,111,114,117,119,121,123,125,127,131,131,131,
131,131,131,131,50,53,55,57,59,61,63,65,67,69,71,73,75,78,81,
83,85,87,89,9,93,95,97,99,102,105
14 IFF$<>" "THEN16
15 FORN=1TOA*8:NEXT
16 NEXT T:IFR=1THEN2
17 RETURN
18 CLS
19 C$="":FORQ=1TO5:K=RND(36):K=K+47:IFK>57THEN21
20 K$=CHR$(K):GOTO23
21 K=K+7:K$=CHR$(K):GOTO23
22 GOTO19
23 C$=C$+K$:NEXT
24 C$=C$+" ":GOSUB10
```

```
25 GOTO19
26 C$="CQ CQ CQ DE WB8QHX WB8QHX WB8QHX K
27 R=1:GOTO10
28 FORK=1TO3
29 C$=Z$+" ":GOSUB10
30 NEXT
31 C$="DE ":GOSUB10
32 FORK=1TO3
33 C$="WB8QHX ":GOSUB10
34 NEXT
35 R=1:C$="K":GOTO10
36 C$=Z$+" DE WB8QHX":GOSUB10
37 IFN$=""THEN40
38 C$=" RR FB "+N$+" ":GOSUB10
39 IFY=3THEN48
40 C$="YUOR RST IS ":GOSUB10
41 C$=R$+" "+R$:GOSUB10
42 C$=" MY QTH IS MUSKEGON,MICH MUSKEGON,
MICH. MY NAME IS JIM JIM. ":GOSUB10
43 IFN$=""THEN45
44 C$="WELL "+N$:GOSUB10
45 C$=" IT IS TIME TO TURN IT BACK TO YOU ":
GOSUB10
46 C$=" HOW DO YOU COPY ":GOSUB10
47 C$=Z$+" DE WB8QHX K":R=1:GOTO10
48 PRINT:INPUT"IT IS YOUR TURN TO SEND.
TO SIGN OFF PRESS ENTER";L$
49 GOTO44
50 GOSUB8:GOTO9
53 GOSUB78:GOTO67
55 GOSUB78:GOTO78
57 GOSUB78
59 GOTO8
61 GOSUB67:GOTO78
63 GOSUB9:GOTO78
65 GOSUB67
67 GOSUB8:GOTO8
69 GOSUB50:GOTO75
71 GOSUB78:GOTO9
73 GOSUB50:GOTO67
75 GOSUB9:GOTO9
78 GOSUB9:GOTO8
81 GOSUB75:GOTO9
83 GOSUB50:GOTO78
85 GOSUB75:GOTO50
87 GOSUB50:GOTO8
89 GOSUB67:GOTO8
93 GOSUB67:GOTO9
95 GOSUB67:GOTO50
97 GOSUB50:GOTO9
99 GOSUB78:GOTO50
102 GOSUB78:GOTO75
105 GOSUB75:GOTO67
108 GOSUB97:GOTO75
111 GOSUB67:GOTO81
114 GOSUB89:GOTO75
117 GOSUB65:GOTO9
119 GOSUB65:GOTO8
121 GOSUB9:GOTO65
123 GOSUB75:GOTO89
125 GOSUB81:GOTO67
127 GOSUB81:GOTO78
129 GOSUB81:GOTO75
131 IFF$<>"","THEN134
132 GOSUB63:GOTO97
134 IFF$<>"","THENRETURN
135 GOSUB87:GOTO71
```

32K RAM for the Color Computer

by Bob Lentz

1. To provide an additional 16K of RAM to a 16K Color Computer, obtain eight more 4116 dynamic RAM chips. These may be "piggy-backed" by placing them over the existing chips and soldering all pins (except Pin-4) onto the corresponding pins of the chip below. Pin-4 should be bent up out of the way to connect in the next step. See figure 1 for a sample piggy-backed chip.

2. Connect together the eight Pin-4s which were left unconnected in the step above. Connect these through a 33 Ohm resistor to Pin-35 of the 6883 SAM chip (U10). See figure 2 for a diagram of the completed modification.

At this point, your computer has 32K of RAM; you may put it back together and try it. (Note: the 4/16K jumpers on the PC board should be left in the 16K position, and no software changes are necessary). Try typing PRINT MEM. The exact value of MEM depends on how much RAM your BASIC takes up, but in any case it should be over 24000.

If you wish to be able to put the video display into the upper bank of RAM, you will need to continue with steps three and four below. This would be necessary if, for example, you wanted to declare more than 16K worth of Hi-Res screens under Extended Basic. For most applications the following steps are not necessary.

3. Remove the PC board from the box, and remove the ground shields to expose the bottom of the board. Cut the traces shown in figure 3. Also cut the trace on the top of the board which connects to Pin-11 of the 74LS273 (U6). See figure 4 for a diagram of this.

4. Refer to the before-and-after figures (5 and 6). Wire the circuit shown in figure 6 using the unused sections of the 74LS273 to latch the display data from either of the two RAS lines.

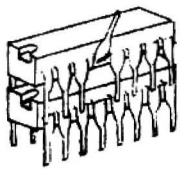
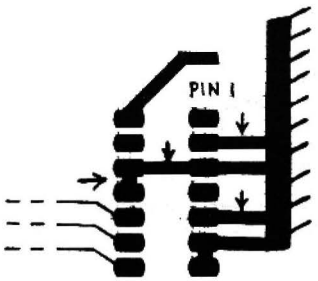


FIGURE 1.

FIGURE 2.



U29 - CIRCUIT SIDE
ARROWS INDICATE TRACE CUTS

FIGURE 3.

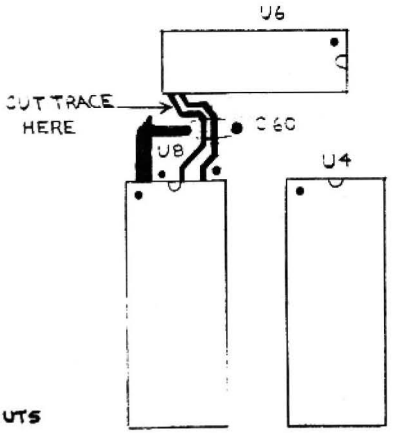


FIGURE 4.

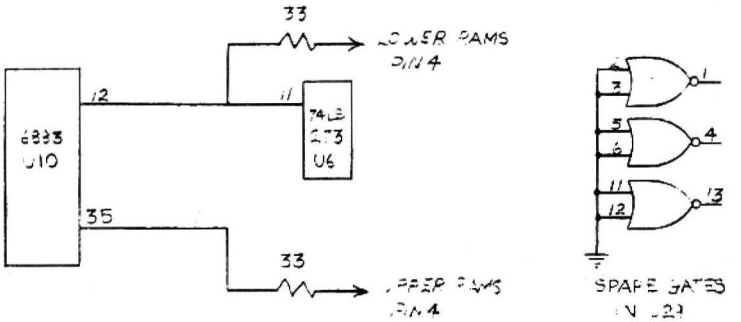


FIGURE 5.

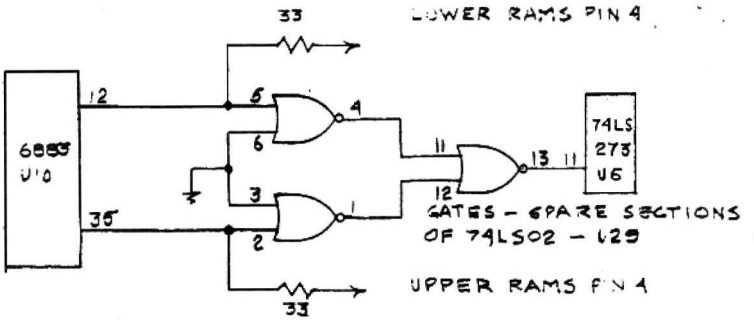


FIGURE 6.

Prime Number Generator
by David Bodnar

Prime numbers have always held a magical fascination for me and for many of my 6th grade math students. Ever since pocket calculators made long division no more difficult than simple addition I have challenged my students to find 4, 5 and 6 digit prime numbers.

We test numbers for primeness* by dividing the number by all of the primes up to the number's square root. If none of these division problems come out evenly then the number is prime.

I developed the following program to generate a list of primes. It is based on the test procedures outlined above and it is the fastest prime number generator I have seen.

One unique feature of the program is the array A(N). It is an internal list of prime numbers that is updated each time a new prime number is found.

The program is written in non-Extended Basic and requires 16K to generate a fairly large list of primes. If you have Extended Basic you must PCLEAR 1 before running to free enough memory to dimension array A(N). You could also change line 140 to IF X / A(N) >= SQR(X) THEN NEXT X since the Extended Basic has a square root function.

I recommend that you remove all REMarks and spaces and renumber by 10s to allow maximum memory.

* I make up words too!

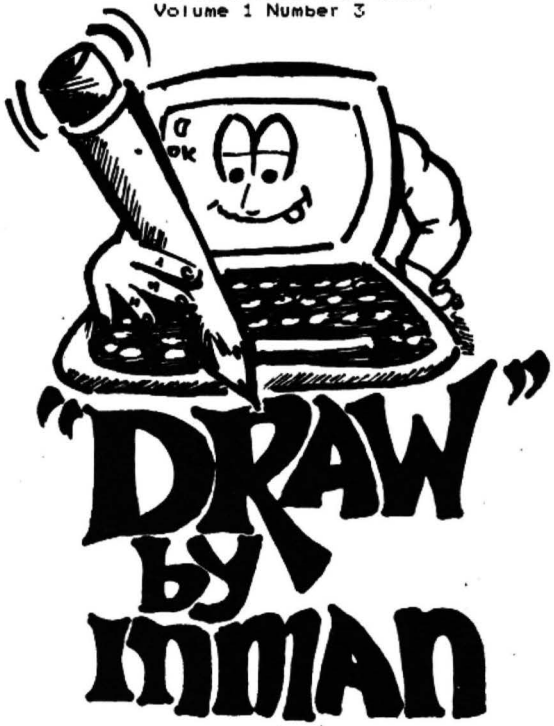
```
5 PCLEAR 1 'OPTIONAL OF EXTENDED BASIC
10 REM GENERATES A LIST OF PRIMES A(N) IS THE LIST
30 REM THIS DIM A IS ALL 16K MEMORY CAN HOLD
40 DIM A(2400)
50 Z=2400*2400
60 CLS
70 A(1)=3
80 PRINT 2;
90 FOR X=3 TO Z STEP 2
100 'N IS COUNTER FOR TEST PRIME ARRAY
110 N=0
120 N=N+1
130 IF X=A(N) THEN 180:'IF X=3 THEN PRINT AND ADD TO ARRAY
140 IF X/A(N)-INT(X/A(N))=0 THEN NEXT X: 'DOES X DIVIDED BY    TEST PRIME
COME OUT EVEN?
150 IF A(N)>X/A(N) THEN 180:'IS TEST PRIME GREATER THAN    SQR(X)?
160 GOTO 120: 'RETURN FOR NEXT LARGER DIVISOR
170 IF W=2400 THEN 220: ' IF UP TO LIMIT END
180 W=W+1: 'COUNTS THE PRIMES
190 A(W)=X: 'PUT NEW PRIME INTO ARRAY OF PRIMES
200 PRINT X;
210 NEXT X
220 END
230 D. BODNAR- 4-29-81
```

Were you aware that Radio Shack will send at no charge a write-up called "High Resolution Graphics Using Color Basic". Call the Hot line at (800) 433-1679.
Thomas Ernst
and
Dennis Hay-Chapman

$2.50

Color Computer News

September/October 1981
Volume 1 Number 3



Color Computer News is published bi-monthly by REMarkable Software.

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" & "CBUG"
SWITCH SELECTABLE,
ON ROM PACK.

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

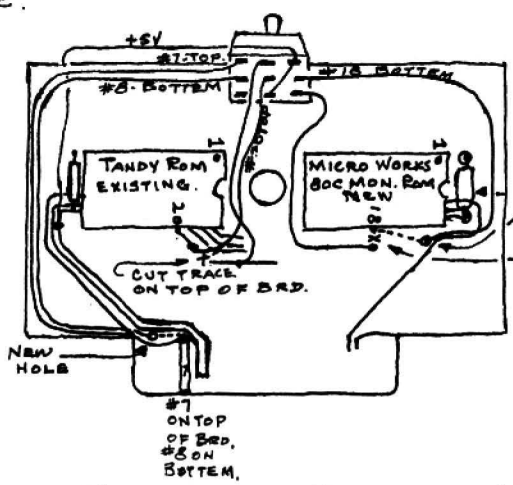
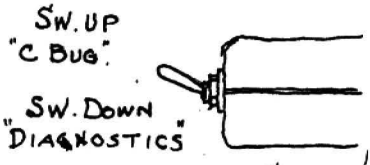


FIG 1.



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.)

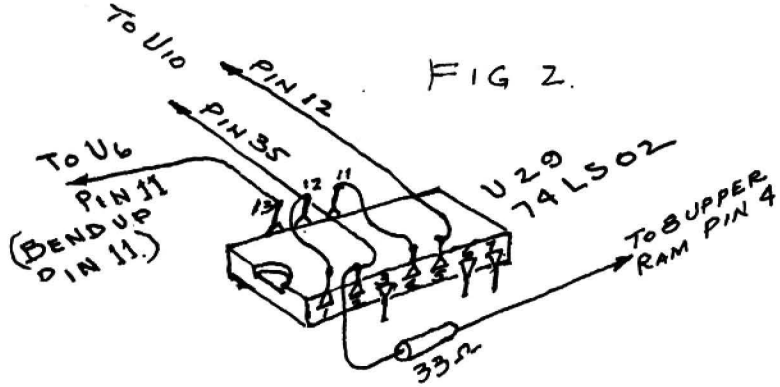


FIG 2.

```
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
```

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 ↗

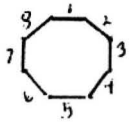
DRAW

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.

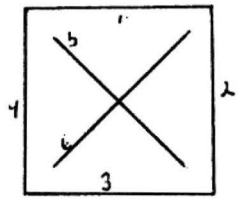


1. R40
2. F40
3. D40
4. G40
5. L40
6. H40
7. U40
8. E40

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 start left of last position

minus sign indicated start up from last 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
```

```
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.


```
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 "C" and "O" at the same time, and only "C" 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
A24E    OFFSET TABLE POINTER
A251    BEYOND "?"
A253    THEN CONTROL
A255    CHECK SHIFT
A257    IF >"+" THEN INVERT SHIFT
A259    SKIP - SHIFT OK
A25B    INVERT SHIFT
A25D    TEST SHIFT
A25E    OK - GO DEBOUNCE
A260    ADD $10 TO MAKE NUMERIC
A262    GO DEBOUNCE
A264    TIMES 2 FOR TABLE INDEX
A265    CHECK SHIFT
A267    SKIP IF NO SHIFT
A269    PLUS ONE IF SHIFT
A26A    GET ASCII FROM TABLE
A26C    GO DEBOUNCE
A26E    UP ARROW - UNDERLINE
A270    DOWN ARROW - "["
A272    LEFT ARROW
A274    RIGHT ARROW - "]"
A276    SPACE BAR
A278    ZERO
A27A    ENTER
A27C    CLEAR - "\"
A27E    BREAK
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)
O										
O	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$(I)
530 IF F=0 THEN S=S+1
540 NEXT I
550 IF F)0 THEN FORJ=1TO400:NEXTJ
560 S=100*S/N
570 FOR I=0 TO S
580 CLS(I)
590 PRINT @198,"YOUR SCORE IS";
:PRINT USING "###.##";S;:PRINT "%";
600 SOUND 20*(I+1),5
610 NEXT I
620 REM GRAPHICS ROUTINE
630 PMODE 4,1:PCLS
640 SCREEN 1,1
650 CIRCLE (128,96),50
660 CIRCLE (153,77),5
670 CIRCLE (103,77),5
680 CIRCLE (128,96),5
690 IF S)=65 THEN CIRCLE (128,96),
35,,1,,1,,4 ELSE CIRCLE (128,146),
35,,1,,6,,9
700 IF S)=65 THEN A$="T2L402FL8G#G#O3L
4C#C#L8FFL4FC#L8C#O2L4G#G#L8FF"
:PLAY A$
710 IF S<65 THEN A$="T402L2FL8FL1B-"
:B$="O2L3FL4B-O3L2D"
: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

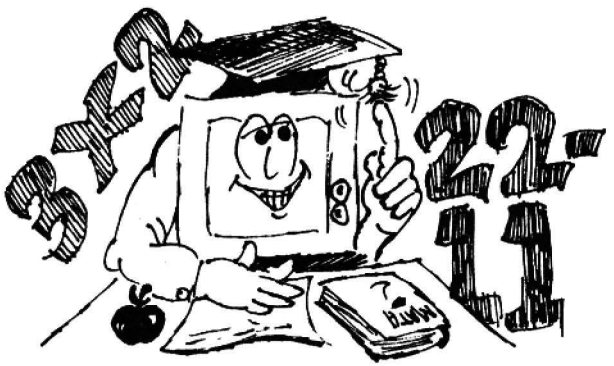
Kid's Page

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 PMODE3,2
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
```

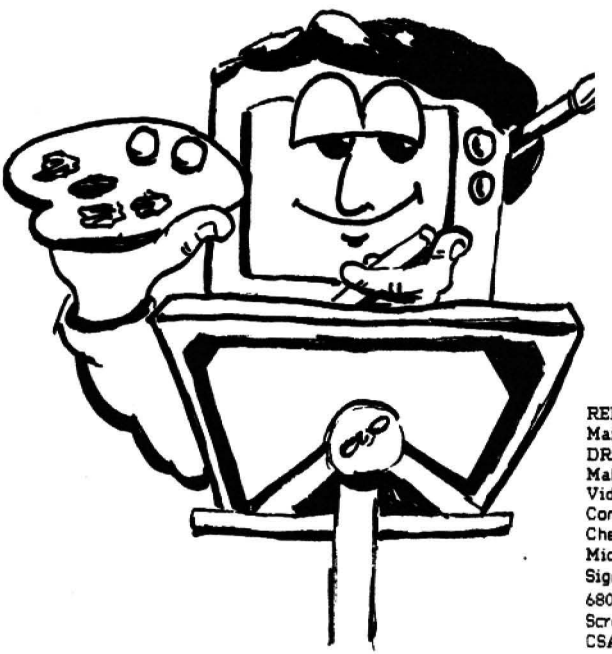
```
                        *           MULTIPLE OF EACH
                        *           PRIME
0004 1755               PRIMES RMB 2*NUMPRM    PRIMES WHICH
                        *           HAVE BEEN DISCOVERED
                        *           SO FAR
0005 1B3D               TEST   RMB 2           NUMBER BEING TESTED
                        *           TO SEE IF IT IS PRIME
0006 1B3F               MP     RMB 2           NUMBER OF PRIMES
                        *           FOUND SO FAR
                        ********************************
                        *
                        *  START HERE
                        *
0007 1B41 CC0001        START  LDD #1          INITIALIZE LOOP
0008 1B44 FD1B3D               STD TEST
0009 1B47 CC0000               LDD #0          ZERO FOUND SO FAR
0010 1B4A FD1B3F               STD MP
                        *
                        *  OUTER LOOP - TRY NEXT NUMBER
                        *  TO SEE IF IT IS PRIME
                        *
0011 1B4D FC1B3D        NEXPRM LDD TEST
0012 1B50 C30001               ADDD #1         NEXT CANDIDATE
0013 1B53 FD1B3D               STD TEST
0014 1B56 BE1B3F               LDX MP          INDEX INTO TABLE
0015 1B59 ED891755             STD PRIMES,X    := TEST
0016 1B5D ED89136D             STD CCNT,X      TIMES ONE
0017 1B61 8EFFFE               LDX #-2         TO START AT ZERO
                        *
                        *  INNER LOOP - TRY FACTORS
                        *
0018 1B64 3002          NEXTES LEAX 2,X        COUNT THRU TABLE
0019 1B66 10A389136D    BUMPQ  CMPD CCNT,X     MULTIPLY OK?
0020 1B6B 2311                 BLS NOBUMP      IF NOT NEEDED
0021 1B6D EC89136D             LDD CCNT,X
0022 1B71 E3891755             ADDD PRIMES,X   ADD 1 MORE
0023 1B75 ED89136D             STD CCNT,X
0024 1B79 FC1B3D               LDD TEST        RESTORE D
0025 1B7C 20E8                 BRA BUMPQ       TRY AGAIN
0026 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.
                        *
0027 1B80 BC1B3F               CMPX MP         MADE IT TO END?
0028 1B83 26C8                 BNE NEXPRM      NO, LOOP
0029 1B85 3002                 LEAX 2,X        INC. # OF PRIMES
0030 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.
0031 1B8A 8D07                  BSR OUTPUT      PRINT NUMBER

0032 1B8C 8C03E8                CMPX #2*NUMPRM  DONE?
0033 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.
0034 1B91 3F                    SWI

0035 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.
                        *
0036 1B93 3416          OUTPUT  PSHS D,X        SAVE REGISTERS
0037 1B95 308D0024              LEAX TAB10,PCR
0038 1B99 327F                  LEAS -1,S       ROOM FOR COUNT

0039 1B9B 6FE4          A@      CLR 0,S         CLEAR COUNT
0040 1B9D 6CE4          B@      INC 0,S         COUNT SUBTRACTS
0041 1B9F A384                  SUBD 0,X        TRY SUBTRACT
0042 1BA1 24FA                  BHS B@          BRANCH IF IT FIT
0043 1BA3 E381                  ADDD ,X++       ADD IT BACK ON
0044 1BA5 3406                  PSHS D
0045 1BA7 A662                  LDA 2,S         GET COUNT
0046 1BA9 8B2F                  ADDA #'0-1      MAKE ASCII
0047 1BAB BDA30A                JSR $A30A       OUTPUT CHAR
0048 1BAE 3506                  PULS D
0049 1BB0 6D01                  TST 1,X         END OF TABLE?
0050 1BB2 26E7                  BNE A@          LOOP IF NOT END

0051 1BB4 860D                  LDA #$0D
0052 1BB6 BDA30A                JSR $A30A       CARRIAGE RETURN
0053 1BB9 3261                  LEAS 1,S        CLEAN UP STACK
0054 1BBB 3596                  PULS D,X,PC

0055 1BBD 271003E800    TAB10   FDB 10000,1000,100,10,1,0

0056 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
```

$2.50

Color Computer News

November/December 1981
Volume 1 Number 4



Color Computer News is published bi-monthly by REMarkable Software.

REMarks
by Bill Sias

The price increase in last issue's ad was in error. Someone got overly greedy and announced next years prices effective this October and even that price was wrong. My sincerely appology for any inconvience this may have caused anyone. Those folks that missed the "deadline" were credited with 8 issues instead of six. The actual price starting January 1, 1982 is $18.00 dollars for 12 monthly issues.

To clear up some apparient misunderstanding, Computer Plus includes a free copy of Color Computer News with every Color Computer that they sell. They do not, however, give away free subscriptions.

The first came from a fellow named Victor Andrews in the form of a phone call. It turns out that Soft Sector Marketing sells some CC software as well as their Models 1 and 3 programs. Anyway Andrew has donated his disassembler program to all of us as my usual Assembler article. Thank you Andrew!!

Transformation Technologies' C.C. Writer was reviewed in a syndicated newspaper column called breaker breaker by Fred Simon. The article appeared Sunday September 13 and talks about CB, Home computers and the "lack-at least on the part of Tandy Corp. (Radio Shack)" of "a good program that will do 'word processing' for the 16K or 32K TRS-80 Color Computer". Mr. Simon then talks specifically about Bill Dye and C.C. Writer and promises to keep everyone informed about new software. It's good to see more support for the Color Computer especially in a newspaper column.

We are continuing to grow at a rapid pace but I need your help. The vast majority of the articles are being written by a minority of the readers. We need your article NOW. In case you didn't know we do pay for articles and we have the easiest submission requirements of anyone in the industry. Acceptable formats are, in order of my preference: Color Computer tape or disk data file, TRS-80 Model 1 tape or disk, Flex disk, direct link via modem, double spaced typewritten, or hand written. Although all of the above formats are acceptable and I am not overly fussy about any of the above but the following will hinder yourchances of being published: hand written programs of more than a few lines or typewritten/printed programs of more than a page. All material submitted becomes our property and will not be returned unless submitted with a self addressed return mailer with sufficent postage attached. Acceptable topics are: programs of any sort, tutorials, hardware mods/descriptions, programming tricks/tips, software reviews, hardware reviews. The list is almost endless.

We have a few copies of Ole #1 back in stock. Please, if you would like one, send your order in under separate cover and include the word "Back Issue" between REMarkable Software and P.O. Box 1192. The cost is $2.50 and if you would like first class mail include an additional $.75. First class copies will be mailed every Friday, others will wait until we have enough orders to send bulk. They will be sold on a first come first served basis and when these are gone there will be no more. When we do run out I'll return your check, no charge cards. Number 2 is in stock, same deal except include $.75 for first classs postage. Number 3 is gone.

For several months now we've talked about changing into a monthly publication, well I have good news for you. Starting with the January issue we will be monthly! We have more than doubled the number of pages in the first issue now and I'm discovering that we're having a rough time keeping the information current so we've done it! I'm depending on you folks to remember that we need those articles more than ever.

Dear Bill,
The price was too good to be true, a 32K memory kit for $12.00 (less memory). This is just what we all have been looking for. An easy way to upgrade our Color Computers to 32K and for only $12.00.
Yes it is too good to be true, but Culpepper Computer Designs' ad says just that. "All parts and detailed instructions (less memory)". Now I'm one to try to save a little money when I can, so I ordered the kit. It took about four weeks to get here. That isn't too bad for only 600 miles, even with the Postal Service what it is. But what I received is the real story. I got a piece of wire 18" long and a small clip. Yes, that's it. Oh yes, the detailed instructions. The instructions are a real joke, no reference to any registers in the connection from Pin 4 to Pin 35 of the SAM. The test program POKEs the number 16 into memory, that's it.
I think that this type of advertising borders on illegal. This is another example of advertising ripoff. Anyone who wants to upgrade should follow the instructions in the July/August issue of CCN and save their $12.00.
The address of Culpepper Computer Designs is: 502 S. East Street, Culpepper, VA 22701.
But save your money this is a real ripoff.
Bob Vaughan
120 Eastpoint Dr.
Charleston, WV 25311

Dear Sir:
Take away a little heat.
With 16K of RAM I found that the temperature under the top cover was 49.5 degrees above the ambient air after running 2 hours. By painting the top cover, part # AZ5846, a dull black inside and outside, (being careful to not paint the fingers where contact is made with the top cover support), the temperature was 42.5 degrees above the ambient air temperature. The temperature rise will vary somewhat with the program being run.
How much does 32K increase the temperature?
I have also noticed that the front cover of the magazine says that "Color Computer is a trademark of the Tandy Corporation". I do not believe this is correct. TRS-80 Color Computer, yes, but not Color Computer. I have a device called the Color Computer which was made years ago, before Radio Shack even thought of their Color Computer.
Charles Worstell
36012 Military Rd S.
Auburn, WA 98002

* You are correct. When I typeset the trademark notice back in April I forgot about the Micro Chroma and the various other "Color Computers".

In the last two issues of the Microcomputer Newsletter, Tandy's official support for their computers, the statement has been made that it was impossible to recover the last graphic page when using Extended Basic. In other words, there is no possible way to execute a PCLEAR 0. It gives me great pleasure to again reinforce the fact that a group of hobbyists with a challenge can accomplish more that a corporation with paid professional programmers. Thanks to the challenge put forth in those newsletters hobbyists all over the country have proven that we know more about the Color Computer than Tandy does. To wit:

To PCLEAR 0 type:
POKE 1536,0: POKE 25,6: POKE 26,1: NEW
Thanks to Phil Beistel
Dave Bodnar

If you haven't heard about the "PCLEAR 0" yet, try this:
POKE 25,6: POKE 27,6: POKE 29,6: POKE 31,6 <ENTER>
PRINT MEM should give 14631 .. and it is useable for programs contrary to what RS says it's latest Microcomputer Newsletter.
R. Wayne Day
1779 Continental Dr.
Blue Mound, TX 76131

Greetings:
Since memory locations 25-32 are pointers, changing the point should make the "other" page of graphics memory open to programmers. On power up 25, 27, 29 and 31 hold 30 but after PCLEAR 1 those locations hold 12 therefore POKEing 6 into those locations moves the point back to the start of the graphics pages.

Pointer:
25, 26 = beginning of Basic program
27, 28 = beginning of simple variables
29, 30 = beginning of subscripted variables
31, 32 = beginning of free memory
H. D. Bassett

This is just a sampling of the letters I've received on this subject. The moral of the story is: Never tell a hobbyist that he can't.

I need help on the 4 pin (Din) I/O port for my Ham Radio Interface. I have been unable to find this info for serial interface. I don't need modem. I just want to talk to my HK terminal any help or reference will be appreciated.
Jesse F. Lee
W5GCJ
3105 Edgewood
San Angelo, TX 76903

For Mark Lockwood & Others
You can get Extended Basic for $90.00 from Sound Center Radio Shack, White Rock Shopping Center, Los Alamos, NM 87544 Phone (505) 672-9824
Roland C. Wong
West Covina, CA 91792

Dear Sir,
I have your Sept/Oct issue of Color Computer News. I tried programming "Spellit" pg 27. It works great but I can't get it to record the words and definitions on tape. I'm not very familar with programming yet and hope that you can tell me why I'm having this problem. I notice that line 460 immediately follows line 400. Are lines 410 - 450 missing? Please help!

Lillian V. Panagakos

* Oops! Here are the missing lines!
410 INPUT #-1,N
420 FOR I=1 TO N
430 INPUT #-1,W$(I): INPUT #-1,D$(I)
440 NEXT I
450 CLOSE
We print the program listings directly from a running copy of the program, but apparently when we cut it for layout those lines went on the floor instead of the layout sheet.

Gentlemen,
Great magazine! Has anyone devised a way to "DRAW" with angles other than the 45, 90 degrees etc given as options? I.E. DRAW at 37 degrees?
Sincerely
Ralph Coleman
2306 Griffin Rd
Churchville, NY 14428

Dear Bill;
Enclosed are two listings for a screen dump to a printer. One was assembled on Micro Works SDS80C editor assembler monitor. The other is in Basic which loads into RAM in the Exatron "Thing".
CCN is getting better every issue. Keep up the good work. I especially like the articles on machine language. They are very helpful to a novice like me.
Sincerely,
James C. Whitaker
2821 Reagan #102
Dallas, TX 75219

```
100 FOR I=1 TO 50
110 READ B
130 POKE 51500+I,B
140 NEXT I
150 DATA 16, 142, 4, 0, 198, 32, 166, 160,
129, 63, 34, 4, 139, 96, 32, 6, 129
160 DATA 95, 37, 2, 128, 64, 189, 162, 191,
90, 38, 234, 134, 13, 189, 162, 191
170 DATA 142, 0, 111, 18, 18, 48, 31, 38,
252, 16, 140, 6, 0, 37, 212, 57, 0
210 DEF USR0=51501
220 A=USR(0)
```

```
            NAM  SCRNDMP1 8/
25/81
*
**SCREEN DUMP TO PRINTER
**ASSEMBLED ON MICRO WORKS
**SDS80C ROMPAK
**BY JAMES WHITAKER
**DALLAS, TEXAS
*
KEY     EQU $A000 POLL KEYBOARD
VIDRAM  EQU $400  START OF VIDEO
PRINT   EQU $A2BF SEND A TO PRNTR
*
**PRINT OUT ROUTINE-SCREEN DUMP
*
START    LDY #VIDRAM ADDR 1ST CHAR
A@        LDB #$20    PRINT 32 CHARS
B@       LDA ,Y+     GET CHAR
            CMPA #$3F   IS IT BELOW 'A
            BHI C@      NO, TRY AGAIN
            ADDA #$40   CHANGE TO ASCII
            BRA D@      SEND IT
```

```
C@      CMPA #$5F   IS IT ABOVE 'Z
        BLO D@      NO, SEND IT
        SUBA #$40   CHANGE TO ASCII
D@    JSR PRINT   SEND IT TO BUFF
        DECB        IS IT LINE END
        BNE B@      NO, GET NEXT CHAR
        LDA #$0     CARRIAGE RETURN
        JSR PRINT   PRINT IT
        LDX #$111   1 MILLISEC DELAY
        NOP
        NOP
T@      LEAX -1,X
        BNE T@      END DELAY
        CMPY #$600  IS IT END VID
        BLO A@      NO, GET NEXT LINE
*
**GO TO BASIC
*
        RTS         RETURN TO BASIC PROG.
        END
```

Dear Bill,
Here is my check for an additional 6 issues. Bill, I will have to agree with Mr. Harrison (Mail Call Sept/Oct). CCN is a tool I use as Reference, it's more than a magazine. Bill, did you know that Radio Shack is now selling a 32K upgrade for the CC, the computer they said could not be upgraded more than 16K!! HA HA
Say Bill I thought we were going to be able to buy the programs on tape that are in CCN. Keep up the good work and let's go monthly.
A Happy Reader,
Robert Salyer

* The 32K upgrade uses 64K chips with the upper bank uncertified. You can buy the programs on either tape or Exatron format disks. The prices are $7.95 for cassette and $10.95 for the disks. They also include some extra programs that readers have submitted. I haven't said much about it because they don't contain many programs yet.

Dear Sir;
I have two questions to ask you:
1) Is there any way to bypass the break key from BASIC?
2) Is it possible to divert error messages so they will not halt the execution of a BASIC program (i.e. ON ERROR GOTO)?
I have found a need for both of these in my every day programming.
Yours Truly,
Tom Markson
366 Newburn Drive
Pittsburgh, PA 15216

Dear Sirs,(CCN)
If CCN prints machine language or assembly language programs would you please explain to novices like me how to run them!! or what we need in order to run them. Be detailed. Yes, I have Extended Basic, but we are not experienced in changing programs to Extended BASIC from 4K. So please take the trouble to PRINT ALTERNATE LINES COMPLETELY in CCN.
With Regards
Joel Cohen
5 Terry Terrace
Livingston, NJ 07039

Dear Bill;
After our conversation the other day I felt that I should drop you a quick note to explain just how I came to purchase my Color Computer. My neighbor had just informed me that he had this new computer that I ought to see. I must admit the whole thing sounded rather boring to me, but I was going to be a nice guy and go see this silly computer of his. He showed me the usual things that one feels he must do to impress someone with their new "toy" and to this point I was not too impressed. I have to tell you Bill, that I have never been too taken with the arcade type games and sure enough this was the next item that he forced upon me. OK, so this little keyboard can play games. Big Deal!!!
What happen next, Bill, is quite incredible. out came this adventure game called "Black Sanctum". It was my turn to operate the computer, there I was standing out in the woods with a cabin in sight and it was starting to snow. The computer asked me what I should do. With a little prompting from my neighbor, I said, "go cabin", and much to my surprise the computer informed me that I was at the cabin door and "what should I do". A strange feeling began to overcome me and I felt that I was being transported to another medium. kind of a twilight zone that had taken control of me. This little game had me hooked for the next three hours and then I had to tear myself away. The following day at work, I found myself calling my neighbor to see if he had gotten any further in the adventure. I rushed home early from work and stopped at his house to take another look at this "game". Now we were in a maze of passages and corridors that ran either under or behind the cabin. In the ensuing days I spent hours in front of his computer working on

the "Sanctum" with a glazed look on my face. I dragged my wife over to see this machine that was taking her husband away from her. I secretly hoped that she too was going to be taken with this thing that by now had a tight hold on me, for, I knew that sooner or later I was going to have to talk to her about this computer that I just had to have.

There is no question, Bill, that at this point I was hooked and just had to have that computer. To further drive me crazy, my neighbor informed me that he had solved the mystery of the "Black Sanctum" but he would not tell me how. That did it. It was off to the computer dealers to find the best price and get that little bugger in my home so that I could go back to work on the "Sanctum". And I did. I too have solved the mystery of the "Sanctum" after several hours of talking first to my Color Computer and then myself. Now I wonder just who is this who is clever enough to write a program of this nature. I can live without ever knowing his name, but I have to warn him, My wife would like to have a few words with him...

Sincerely Yours
Thomas L. Mix
3424 College N.E.
Grand Rapids, MI 49505

```
1 REM THIS IS TIEFITE BY JML 4-4
-81
2 CLS:PRINT@165,"WELCOME TO TIE
FITE"
3 PRINT@480,"JML 4-4-81"
4 FOR D=1 TO 2000:NEXT D
5 AS=0:SC=0
6 PRINT@320,"THERE ARE 10 TIE FI
GHTERS"
7 PRINT@362,"TO SHOT DOWN JEDI"
8 FOR D=1 TO 2000:NEXT D
10 CLS(0):GOSUB 500:FOR D=1 TO 1
000:NEXT D:GOTO 400
20 SET(H,V,0)
30 SET(H-1,V,0)
40 SET(H-1,V-1,0)
50 SET(H-1,V+1,0)
60 SET(H+1,V,0)
70 SET(H+1,V-1,0)
80 SET(H+1,V+1,0)
90 RETURN
100 SET(H-1,V,0)
110 SET(H,V-1,0)
120 SET(H,V+1,0)
130 SET(H-2,V,0)
140 SET(H-2,V-1,0)
150 SET(H-2,V-2,0)
160 SET(H-2,V+1,0)
170 SET(H-2,V+2,0)
180 SET(H+1,V,0)
190 SET(H+2,V,0)
200 SET(H+2,V-1,0)
210 SET(H+2,V-2,0)
220 SET(H+2,V+1,0)
230 SET(H+2,V+2,0)
240 RETURN
250 SET(H,V-1,0)
260 SET(H,V+1,0)
270 SET(H+1,V-1,0)
280 SET(H,V,0)
300 X=JOYSTK(0):Y=31
301 IF X>63 THEN X=63
302 IF X<1 THEN X=1
310 SET(X,Y,0)
315 RESET(X,Y)
320 P=PEEK(65280)
325 IF P=126 OR P=254 THEN 330 E
LSE RETURN
330 FOR M=30 TO 1 STEP-1
331 SET(X,M,0)
332 RESET(X,M)
333 IF X=H AND M=V THEN 340 ELSE
 334
334 IF X=H+1 AND M=V OR X=H-1 AN
D M=V THEN 340 ELSE 338
338 NEXT M
339 RETURN
340 SC=SC+1:IF SC=10 THEN 2000 E
LSE 341
341 GOSUB 800:GOTO 400
400 H=RND(53)+5:V=RND(25)+3
401 GOSUB 300
405 SET(H,V,0)
406 FOR D=1 TO 10:GOSUB 300:NEXT
 D
407 GOSUB 300
409 CLS(0)
410 GOSUB 20
420 GOSUB 300
430 FOR D=1 TO 10:GOSUB 300:NEXT
 D
435 CLS(0)
440 GOSUB 100
445 GOSUB 300
450 FOR D=1 TO 10:GOSUB 300:NEXT
 D
460 CLS(0)
464 SOUND 200,3:SOUND 185,1:SOUN
D 150,1
465 AS=AS+1
466 IF AS=5 THEN 467 ELSE 470
467 CLS:PRINT@170,"TOO BAD JEDI!
"
```

continued on page 36

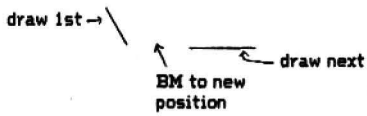
DRAW II

by Don Inman

This is a continuation of the discussion of the use of the DRAW statement begun in the September October issue of Color Computer News. In that issue, several options used with the Extended Color Basic DRAW statement were discussed:

B for Blank - don't draw
M for Move - move to new X,Y position
U,L,D,R,E,F,G,H - draw commands for various directions
Relative Motion - move to a new position relative to the last position
Catenating strings to form the DRAW statement

Draw statements are executed quite quickly and lines can be connected to create geometric shapes as illustrated in the September/October issue. We also showed how to "lift the drawing pen" between lines to a new starting position as:



The No Update Option

You can also draw several lines from some starting point but in different directions. The N option says "No update", which means "return to the origin of this line after drawing it."

Selecting a starting point near the center of the screen, you might use:

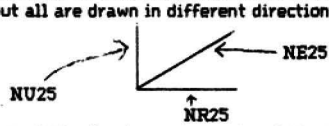
```
DRAW "BM128,96; NU25; NE25; NR25"
```

Draw a line Up 25 units and return to the starting point

Draw a line 45 degrees from Up 25 units and return to the starting point

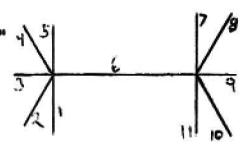
Draw a line Right 25 Units and return to the starting point

When this statement is executed, three lines are drawn. Each line originates at the point 128,96, but all are drawn in different directions.



Let's get a little fancier and draw the following figure using the N option.

```
100 'SET THE SCREEN
110 PMODE 4,1: PCLS: SCREEN 1,1
200 'DRAW FIRST 5
210 DRAW"BM100,96; ND25; NG25; NL25; NH25; NU25"
220 FOR X=1 TO 1000: NEXT X
300 'DRAW NUMBER 6
310 DRAW"R56"
320 FOR X=1 TO 1000: NEXT X
400 'DRAW LAST 5
410 DRAW"NU25; NE25; NR25; NF25; ND25"
420 GOTO 420
```



You could write a variation of the previous program that would draw a "snowflake" pattern.

DRAW II

Angle Option

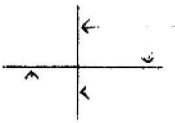
Lines can be rotated 0, 90, 180, and 270 degrees with the Angle command, A. The amount the line is rotated is designated by a subscript following the letter A:

A0 - draw at the direction specified
A1 - add 90 degrees to the direction specified
A2 - add 180 degrees to the direction specified
A3 - add 270 degrees to the direction specified

This option might be used in a FOR-NEXT loop to create 4 different lines.

```
210 A$(1)="A0": A$(2)="A1": A$(3)="A2": A$(4)="A3"
220 FOR X=1 TO 4
230   DRAW A$(X)+"BM128,96;U25"
240 NEXT X
```

The display would produce:

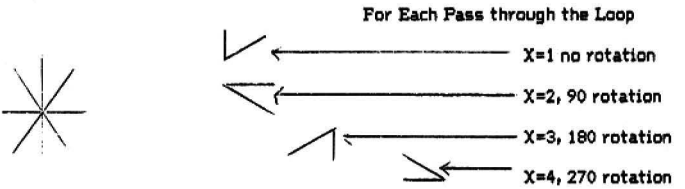


X=1, no rotation = Up
X=2, U+90 = right
X=3, U+180 = down
x=4, U+270 = left

Or, if you want 8 lines instead of four, change line 230 to:

```
230 DRAW A$(X)+"BM128,96; U25; BM128,96; E25"
```

From this change the display would be:



You might try drawing the "snowflake" using this method.

Drawing Nothing

You might think it useless to draw a blank line, but sometimes it might come in handy. Since the Color Computer doesn't like to mix text and graphics on the screen, you probably will want to DRAW some block letters in the graphics mode at times. Suppose you want to draw the letter, F.

By inserting a Blank move at the correct places, you can create the letter with a continuous DRAW statement:

```
DRAW"BM140,80; L25; D50; BU25; R25"
                              ↑— move up 25 but don't draw
```

This creates an "F" with segments shown by arrows (solid are drawn, dotted is not drawn).

Try the following program on your Color Computer to see what it produces.

```
100 'SET SCREEN
110 PMODE 4,1: PCLS: SCREEN 1,1

200 'DRAW A WORD
210 DRAW"BM100,100;L10U20R10;BR15;L10D20R10U20;
BR5;D20R10;BR15;U20L10D20R10;BR5;U20R10D10L10F10"
220 GOTO 220
```

Hint: What kind of a computer is this?

DRAW II

Color

Since this is a color computer, don't you think it's time we put some color into the DRAW statement? This can be done with the command:

C for color Cx — x is a color code (0-8) selected from those possible for the mode and color set in use

This time we'll select PMODE 3 which provides 4 colors from one of the two color sets:

Color Set 0	Color Set 1
green	buff
yellow	cyan
blue	magenta
red	orange

Insert the Color command at the beginning of the DRAW string:

```
PMODE 3,1: PCLS: SCREEN 1,0
DRAW"C3; BM100,100; L10 U20 R20"
```

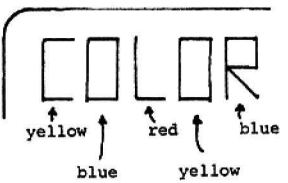
This would display:

C ←a blue C on a green background

Several C commands may be inserted in a DRAW statement to change colors as in the following program.

```
100 'SET SCREEN
110 PMODE 3,1: PCLS: SCREEN 1,0

200 'DRAW COLORFUL WORD
210 DRAW"C2;BM100,100;L10U20R10BR15;
      C3;L10D20R10U20BR5;
      C4;D20R10BR15"
220 DRAW"C2;U20L10D20R10BR5;C3;U20R10D10L10F10"
230 GOTO 230
```

Now, you see on the display:



Many variations could be made such as:

a) randomly selecting the color codes
b) changing to color set 1
c) changing colors for segments of letters etc.

Scale

The SCALE command lets you enlarge or reduce the size of a display created by a DRAW statement. After a SCALE command has been executed, all absolute and relative motion commands will be reduced or enlarged according to the specified SCALE factor. In other words, the SCALE factor stays in effect (within a given program) until a new scale is specified. The SCALE command is:

S for scale Sx — x is an integer (1-62)

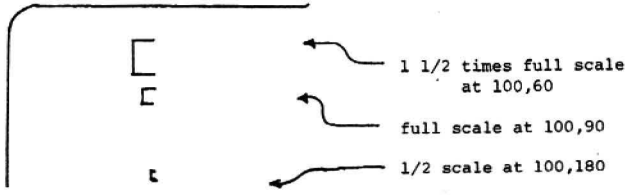
If no scale factor is specified, the computer uses S4, Scale factors for various values of x are:

x	Scale Factor
1	1/4
2	2/4 or 1/2
3	3/4
4	4/4 or full scale
5	5/4
.	.
.	.
.	.
62	62/4 or 15 1/2 times full scale

The following program prints the letter C in three different sizes specified by the scale commands:

```
S2 = half scale
S4 = full scale
S6 = 1 1/2 times full scale

100 'SET SCREEN
110 PMODE 3,1: PCLS: SCREEN 1,1

200 'DRAW & CHANGE SCALE
210 FOR X=2 TO 6 STEP 2
220  S$="S"+STR$(X): Y$=STR$(360/X)
230  DRAW$$+"BM100,"+Y$+":L10U20R10"
240 NEXT X
250 GOTO 250
```

Display:



That's all for this time. Next issue, we'll discuss using substrings within a DRAW statement. We'll also show some applications to try to tie all the DRAW options together (tighten the DRAW strings).

Making Education More Colorful.

by David Bodnar

When I was given the opportunity to write an educational column for COLOR COMPUTER NEWS, I had to make several assumptions about the readers of this column and what they would expect to find in it. My first assumption was that they either owned, worked with or planned to purchase a COLOR COMPUTER.

The second assumption was a bit tougher. Why would a reader be interested in a column dedicated to education? As I thought about it I realized that virtually every owner of a Color Computer needed information about the computer and it's operation. I also realized that each of us has a strong desire to educate our family, friends and colleagues about the potential that the COLOR COMPUTER has for us.

This interest is not confined to learning about computers but it also includes using the computer as a tool to teach ourselves and others about a myriad of topics.

My own interest can be traced back nearly 15 years, but it really intensified last year when I purchased a Color Computer for use in my classroom. Since then my school district has funded a research project designed to test the feasability of teaching 5th and 6th grade students with and about computers. I will keep you informed about its progress so that we all may benefit from this experience.

As you may know there is very little educational software available for the COLOR COMPUTER. If we are to make wise purchases we should have some idea of a program's potential and limitations before buying it. I plan on evaluating software by using it myself and, where appropriate, by having my students take a long, HARD look at it.

This column will also contain suggestions for making programs more "AMICABLE" (I hate "friendly") to your audience, regardless of its age. We will try to end with a short program that may be of interest to classroom teachers, students and parents alike. The first of these appears below.

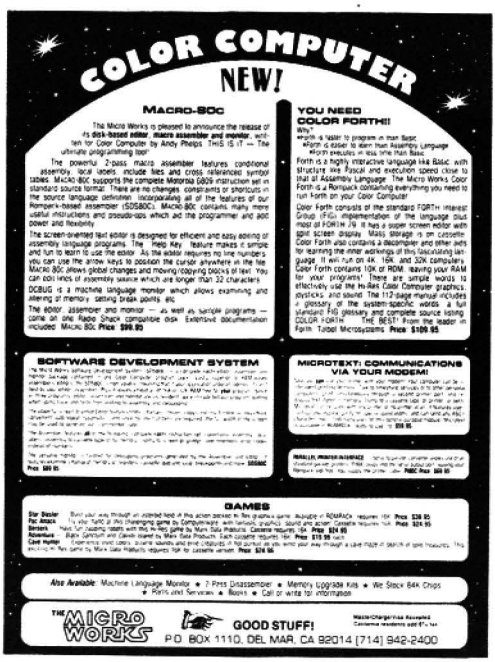
This program began as a tool to help solve letter substitution puzzles that appear in GAMES MAGAZINE. These puzzles consist of a simple letter substition code that turns a common phrase or riddle into a meaningless jumble of letters. For example, the phrase "COMPUTERS ARE LOTS OF FUN!!" could become "DEYNKSLFA MFL ZESA EO OKQ" if all of the C's became D's and all of the U's became K's, etc.

When you run the program it asks you to type a phrase. (The length is limited to 255 characters including spaces and punctuation.) The program then pauses for several seconds to create a random substitution table that will turn all A's to J's, all B's to C's or whatever. It then displays the coded phrase and asks for a letter to change. If you wish to change A's to M's, key A in response to "CHANGE FROM" and M in response to "CHANGE TO". The phrase will appear below the original with all A's replaced by lower case M's. This helps you to keep track of what has been changed. You may not take back a change after it has been entered, but if you get hopelessly lost type "XX" and you will restart with the same phrase.

This program is by no means complete. You may wish to add a feature that will tell you if you have correctly solved the puzzle, although it is usually quite obvious when you are done. You might also add a feature that would allow you to move back 1, 2 or more moves to save starting over if you make errors. It has lots of possibilities ... let me know what you do with it.

```
100 CLEAR 1000: DIM C(26), F1(26), FL(26)
200 CLS: PRINT" TYPE A PHRASE": LINE INPUT B$: CLS: L=LEN(B$)
300 A$=STRING$(L," ")' CREATE SECOND STRING TO BE CHANGED TO CODE
399 '(400-600)-SELECT RANDOM CODE - F1 INSURES AGAINST DUPLICATION OF
LETTERS
400 FOR X=1 TO 26: PRINT@32*6+3,"GETTING RANDOM LETTERS":
PRINT32*7+10,USING"##/26";X
500 C(X)=RND(26): IF F1(C(X))=1 OR C(X)=X THEN 500: ELSE F1(C(X))=1
600 NEXT X: CLS
699 ' (700-1100) CHANGE LETTERS TO CODED EQUIVALENT
700 FOR X=1 TO L
800 A$=ASC(MID$(B$,X,1))-64
```

```
900 IF AS<1 OR AS>26 THEN MID$(A$,X,1)=MID$(B$,X,1): GOTO 1100'IF NOT A
LETTER LET IT ALONE
1000 MID$(A$,X,1)=CHR$(C(AS)+64)
1100 NEXT X
1200 'DECODE BY PLAYER
1250 C$=A$
1300 CLS: PRINT@32*2,A$: FOR X=1 TO 6: FL(X)=0: NEXT X
1400 PRINT@32*13,"CHANGE FROM";: INPUT F$: IF F$="XX" THEN A$=C$: GOTO
1200 ELSE: INPUT"TO";T$: IF T$=F$ THEN 1400
1500 FOR X=1 TO 26: IF FL(X)=1 AND ASC(F$)-64-X THE SOUND 150,10: GOTO 1400
ELSE NEXT X 'IF YOU ALREADY CHANGED THE LETTER THEN IGNORE
1600 FL(ASC(F$)-64)=1 'FLAG TO SHOW LETTER WAS CHANGED
1699 '(1700-1900) CHANGE ALL SELECTED LETTERS TO NEW VALUE
1700 FOR X=1 TO LEN(A$)
1800 IF MID$(A$,X,1)=F$ THEN MID$(A$,X,1)=CHR$(ASC(T$)+32)' +32 MAKES IT A
LOWER CASE LETTER
1900 NEXT X
2000 PRINT@32*8,A$: SOUND 100,1: GOTO 1400
```




```
 1 PRINT@33,"THIS PROGRAM WILL
   BUILD A TREE":PRINT@167,"BY
   THOMAS L. MIX":PRINT@199,
   "3424 COLLEGE N,E,":PRINT@231,
    "GRAND RAPIDS,MICH. 49505"
2 FOR R=1TO460*4
3 NEXT R
5 CLS(5)
10 CLS(0):FORH=0TO63:SET(H,29,5):
   NEXT H
15 FOR V=10TO29:SET(31,V,5):NEXT V
18 H=RND(29):H=H+34
20 FOR V=15TO35 STEP-1:SET(31,V,5)
22 NEXT V
30 H=RND(62):V=RND(20)
40 IF POINT(H,V) THEN 30
50 Q=RND(8):SET(H,V,Q)
60 IF POINT(H,V+1) THEN 30
70 IF POINT(H,V-1) THEN 30
80 IF POINT(H+1,V) THEN 30
85 IF POINT(H-1,V) THEN 30
90 RESET(H,V)
99 GOTO30
```

Videotex
by Gregory H. Cegielski
2029A South 14th Street
Milwaukee, WI 53204

All of the following addresses are shown as Decimal ($HEX).

VIDEOTEX™, from Radio Shack, works this way: It loads with the CLOADM command starting at address 1536 ($0600) with the last location at 3839 ($0EFF); the EXEC pointer is loaded with 1728 ($06C0), and that is where program execution begins. (the EXEC pointer is at location 157 ($009D)).

The first instruction shuts off the IRQ (timer) and FIRQ (cartridge) interrupts; nothing can interfere with VIDEOTEX. A branch is then made around the body of VIDEOTEX, and a routine is entered which rewrites VIDEOTEX at 268 ($010C) and continues for 2080 ($0820) bytes, ending just short of said rewriting routine, at 2348 ($092C). The effective entry address into VIDEOTEX, during the rewriting, lands at 609 ($0261). After the rewriting is complete processor control is transfered to 609 ($0261), which is a NOP ($12), and the actual VIDEOTEX begins. The RESET FLAG, located at 113 ($0071) is then loaded with a HEX ($55), and the RESET VECTOR at 114 ($0072) and 115 ($0073) is loaded with 609 ($0261). For this last reason, pressing the RESET button causes the computer to jump back to 609 ($0261) and reenter VIDEOTEX, until power to the computer is shut off. This can be changed with a poke to address 2103 ($0837) with any value other than 85 ($0055). Without the HEX ($55) reset flag the computer will reenter BASIC; however this will reconfigure about the first half of VIDEOTEX, and you won't be able to use VIDEOTEX again without reloading.

To get around this, I used the same rewriting sequence to first duplicate VIDEOTEX at 30208 ($7600) to 32319 ($7E3F), and have that copy then make a second copy to configure the area below as the original would have. (I have 32K, although this could also be done with any size RAM). At machine speed, there is, of course, no detectable time difference going through an extra copy.

The actual code to do this is as follows:

ADDRESS	MACH. CODE	WHAT HAPPENS
1723 ($06BB)	1A 50	Shuts off IRQ & FIRQ
1725 ($06BD)	16 08 40	Branch to end of VIDEOTEX plus 1
1728 ($06C0) to 3839 ($0EFF)	NORMAL VIDEOTEX - bring in with CLOADM	
3840 ($0F00)	30 8D F7 BC	Puts address of start into X-register
3844 ($0F04)	10 8E 08 3F	Puts length into Y-register
3848 ($0F08)	CE 76 00	New location into U-register
3851 ($0F0B)	A7 C0	Move A's contents into U's address, increase U by 1
3855 ($0F0F)	31 3F	Decrease Y by 1
3857 ($0F13)	7E 76 00	Jump to the copy you just made
3862 ($0F16)	00	To keep BASIC happy

Actually, using just PEEK and POKE you could move your VIDEOTEX up to 30208 ($7600); there were two other factors in my choice of using this routine.

First, because the normal screen is located at 1024 ($0400) to 1535 ($05FF) I wanted to include just before any program coming in from tape a screen of instructions and phone numbers of various CBBS's - - after all, why just look at a blank green screen while waiting for the program to load, may as well have some instructions and the bright colors sure impress visitors.

Using the Extended BASIC CSAVEM command to:

CSAVEM "VIDEOTEX",&H0400,&H0F16,&H06BB

will also write to tape, and reload with CLOADM, whatever is on the screen. Since CSAVEM can be used as a program line, all that is necessary is to first have your program fill in the screen as you desire, and as a near final step save the screen contents and your VIDEOTEX which closely follows. Had you put VIDEOTEX way up in memory to start there could be a lot of extra loading inbetween. When using CSAVEM you can also put a GOTO to loop back to CSAVEM and make enough copies so that you don't have to always rewind. Put a timer, i.e. FOR ZZ=0 TO 500: NEXT between the steps, and you'll have a slight pause between your copies.

The other reason is that there may be times when it may be desirable to offset load your VIDEOTEX (if some memory contained a program that should not be altered, the Micro Works CBUG tape monitor at 1536 ($0600) for example) (And let me add that everybody, everybody, who is serious about their Color Computer should have the Micro Works ROM CBUG MONITOR, which has allowed me more insight into the machine than anything else).

Since I have 32K and VIDEOTEX only stores 26 pages, I ran the Micro Works disassembler on it and found out that what it does is check for the hard wire jumper on PIA $FF22 to determine if the machine is 4K or 16K.

If the jumper is on 16K it stores in register B the value 26 ($1A), otherwise it stores just 2. If you have 32K, you can increase the "off line" storage to 58 pages by a POKE 2112,58:EXEC; although you will want to make this change permanent if you decide to go with the reprogramming of VIDEOTEX already mentioned. In that case you will have to limit your "offline" storage to 53 pages, because the copy of VIDEOTEX you will make up at 30207 ($7600) will need some room, and BASIC's stack can't be allowed to fall into your VIDEOTEX copy if and when you reset to BASIC.

For 32K and Extended BASIC, here is a short program which will at least give you a working copy and 53 pages of storage, but will not fill your screen automatically from tape:

```
10 A=30208
20 POKE 2103,255 REM PERMITS RESET
30 POKE 2112,53  REM STORES 53 PAGES
40 FOR X=1728 TO 3839
50 POKE A, PEEK(X)
55 PRINT CHR$(PEEK(X)); REM (optional)
60 A=A+1: NEXT X
70 CSAVEM "ANY NAME",30208,32319,30208
80 FOR Z=1 TO 500: NEXT Z
90 GOTO 70
```

You few guys with 64K can store 122 pages; POKE 2112,122; if you've got the memory it'll goto 250.

We all should write personal letters of appreciation to the capable engineers at Motorola who gave us this machine.

```
10 'RACECAR' BY PEGGY SCHUBERT
20 'RS COLOR COMPUTER, 4K
30 CLS
40 PRINT"RACECAR
          BY PEGGY SCHUBERT
50 PRINT"PRESS LEFT ARROW TO MOV
E LEFT
60 PRINT"PRESS RIGHT ARROW TO MO
VE RIGHT
70 PRINT"PRESS <ENTER> TO START
80 INPUT X
90 CLS
100 C=4
110 D=RND(3)-2:L=RND(12)
120 FOR Y=1 TO L
130 R=R+D
140 IF R>22THEN R=22:D=D-2
150 IF R<1THEN R=1:D=D+2
160 PRINT@480+R,CHR$(149);CHR$(2
07);CHR$(207);CHR$(207);CHR$(207
```

continued on page 44

COMMENT CORNER

by Andrew Phelps
The Micro Works

The following is a list of comments which could be added to a disassembly listing of the Color Computer ROM. Two sections are given here. The first is the serial output driver in the Color BASIC ROM which is used to send a character to the printer. The second is a serial Input/Output driver from Extended BASIC which allows use of the RS232 port in both directions.

These routines may be called from Assembly Language programs in order to use the Color Computer with a variety of serial peripherals or communication links. New routines for special applications may be written using these as models.

Variables, areas, and routines -

Addr	Comments
0095	BAUD RATE
0097	PRINTER RETURN DELAY
009B	PRINTER WIDTH
009C	PRINT HEAD POSITION
A2BF	START OF PRINTER OUT
A2FB	SEND MARK BIT
A302	DELAY HALF BIT TIME
A7D3	DELAY ROUTINE

Line-by-line comments -

Addr	Comments
A2BF	SAVE CCR,A,B,X
A2C1	INHIBIT INTERRUPTS
A2C3	SEND A STOP BIT
A2C5	REMOVE MSB, ADD START BIT
A2C6	INITIALIZE BIT COUNTER
A2C8	START LOOP; SAVE COUNTER
A2CA	CLEAR B
A2CB	GET BIT TO SEND
A2CC	MOVE BIT INTO B
A2CD	MOVE TO BIT 1 IN B
A2CE	OUTPUT TO $FF20
A2D1	DELAY HALF BIT TIME
A2D3	3 NOPS; PATCH IN CODE
A2D6	OTHER HALF BIT DELAY
A2D8	RESTORE COUNTER
A2DA	COUNT DOWN BITS
A2DB	LOOP FOR EACH BIT
A2DD	SEND STOP BIT
A2DF	RESTORE INTERRUPTS & A
A2E1	WAS IT A CARRIAGE RETURN?
A2E3	IF IT WAS, GO DELAY
A2E5	INCREMENT HEAD POSITION
A2E7	GET HEAD POSITION
A2E9	SAME AS PRINTER WIDTH?
A2EB	IF LOWER, DON'T DELAY
A2ED	RESET HEAD POSITION
A2EF	DELAY FOR C/R
A2F1	DELAY SOME MORE
A2F3	GET RS232 INPUT
A2F6	MOVE INPUT BIT TO CARRY
A2F7	LOOP TIL PRINTER NOT BUSY
A2F9	RESTORE B,X; RETURN
A2FB	GET A "2" (OUTPUT HIGH)
A2FD	STORE TO OUTPUT
A300	CALL DELAY HALF BIT
A302	GET BAUD RATE CONSTANT
A304	SKIP TWO BYTES
A305	GET C/R DELAY CONSTANT
A307	JUMP TO DELAY ROUTINE
A7D3	DECREMENT X REGISTER
A7D5	LOOP TIL ZERO
A7D7	RETURN

EXTENDED BASIC SERIAL I/O

Variables, Areas, and Routines -

Addr	Comments
00E6	BAUD RATE CONSTANT
00E7	INPUT TIMOUT CONSTANT
008A	ALWAYS CONTAINS ZERO
8DBC	INPUT RS232 CHARACTER
8DE6	GET BIT OR TIME OUT
8DF7	DELAY ONE BIT TIME
8E0C	SEND RS232 CHARACTER

Line-by-line Comments -

Addr	Comments
8DBC	SET CONDITION CODE "ZERO"
8DBD	SAVE CCR, B, X
8DBF	INHIBIT INTERRUPTS
8DC1	GET TIMOUT CONSTANT
8DC3	GET A ZERO TO X
8DC5	GET A BIT
8DC7	WAIT UNTIL STOP BIT
8DC9	GET A BIT
8DCB	WAIT UNTIL START BIT
8DCD	WAIT HALF BIT TIME
8DCF	INITIALIZE BIT MASK (=1)
8DD1	SAVE MASK ON STACK
8DD3	CLEAR INPUT BYTE
8DD4	WAIT BIT TIME
8DD6	READ INPUT PORT

```
8DD9  MOVE INPUT BIT TO CARRY
8DDA  BRANCH IF INPUT BIT ZERO
8DDC  OR BIT MASK INTO A
8DDE  SHIFT BIT MASK
8DE0  LOOP FOR NEXT BIT
8DE2  BUMP MASK OFF STACK
8DE4  RESTORE & RETURN
8DE6  READ INPUT PORT
8DE9  MOVE INPUT BIT INTO CARRY
8DEA  INCREMENT X
8DEC  IF NONZERO, RETURN OK
8DEE  COUNT DOWN A
8DEF  IF NONZERO, RETURN OK
8DF1  TIMOUT; REMOVE RET ADDR
8DF3  RESTORE REGISTERS
8DF5  MAKE CONDITION CODES <>0
8DF6  RETURN TO CALLING PROGRAM
8DF7  CALL FOR HALF BIT DELAY
8DF9  SAVE A REG
8DFB  GET BAUD RATE CONSTANT
8DFD  DELAY THREE CYCLES
8DFF  COUNT DOWN
8E00  LOOP FOR DELAY
8E02  RESTORE A AND RETURN
8E0C  SAVE REGISTERS
8E0E  INHIBIT INTERRUPTS
8E10  DELAY 1 BIT (STOP BIT)
8E12  DELAY (2ND STOP BIT)
8E14  SEND A ZERO FOR START BIT
8E17  DELAY FOR START BIT
8E19  START WITH BIT ZERO
8E1B  PUT BIT MASK ON STACK
8E1D  GET DATA BYTE
8E1F  MASK FOR CORRECT BIT
8E21  IF ZERO, GO SEND ZERO
8E23  IF NOT ZERO USE A "2"
8E25  SEND THE BIT
8E28  DELAY FOR ONE BIT TIME
8E2A  MOVE BIT MASK OVER
8E2C  IF MORE BITS, LOOP
8E2E  USE A "2" FOR STOP BITS
8E30  START THE STOP BIT
8E33  CLEAN MASK OFF STACK
8E35  RESTORE AND RETURN
```

QUESTION: What is RS232?

RS232 is a method of sending bytes of data along a single wire. It is used for talking to printers, modems, and CRT terminals.

How can data be sent with just one signal wire?

RS232 data is sent serially; that is, one bit at a time. Bit zero is sent first, then bit one, and so on. Each bit is sent for a predetermined time.

How do you send a bit?

In the Color Computer, a bit is sent by writing it to bit one of location $FF20. Writing $02 to $FF20 will send a "one" by putting -12 volts on the RS232 output line. Writing $00 to $FF20 will send a "zero" by putting +12 volts on the RS232 output line.

How long is the output left at the value for each bit?

That depends on the baud rate. The standard rate for the Color Computer is 600 baud, or 600 bits per second. Therefore there is a programmed delay of 1/600th of a second after each bit is shifted out.

How can a receiving device know when the first bit starts?

When no data is being sent, a "one" value is left on the RS232 line. Before each character is sent, a "zero" is sent for one bit time. This is called a start bit and warns the receiving device that data bits follow. After all data bits are sent, a "one" is sent for at least one bit time to allow the receiving device to process the character before the next start bit is sent.

Does the stop bit always come after the data bits?
The stop bit comes in between bytes, so it doesn't matter if it is thought of as coming before or after the other bits. The Radio Shack printer routine puts one stop bit before and another after the other bits.

How do I receive a bit?

The RS232 input line is connected to bit 0 of location $FF22. LDA $FF22 / ANDA #1 will put the zero or one into the A register.

Then how do I receive a byte?

First, wait for the input to go to a zero. This is the start bit. Now, wait for 1 1/2 bit time so that you can sample in the middle of each bit. Then, do the following eight times: Shift in the next bit, then wait one bit time. When all bits are in, you are done.

How many bits are sent?

Although eight bits can be sent, ASCII characters only need seven bits. The Radio Shack printer routine only sends seven bits, but it sends two stop bits so that the byte is long enough even if the the printer expects eight bits. The output routine at $8E0C sends eight bits.

How is the RS232 port usually hooked up in the Color Computer?

The serial out line is normally assumed to be hooked to a printer. The serial in line is not used for serial in (since printers don't send anything anyway) but rather for a printer busy line: one means busy, zero means ready. The remaining line is called carrier detect and is not used for printers.

What is carrier detect for?

Carrier detect is used by modems to tell the computer that a carrier tone is present. It is edge-triggered; that is, it can can be used to see if the line has gone from one to zero, or from zero to one. It can be used to generate an interrupt.

What are the "Printer width" and "Head position" variables in Basic's printer routine?

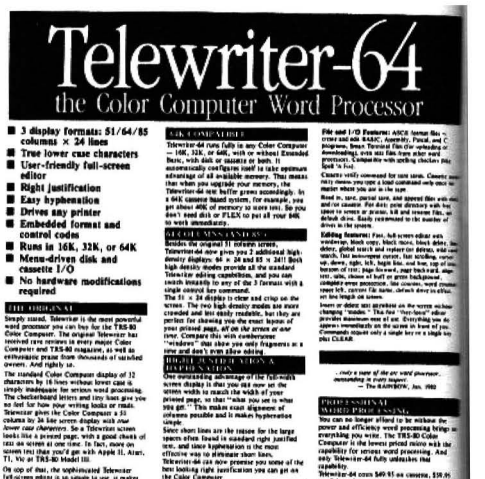
The only purpose of these is to provide a short time delay whenever the computer thinks a carriage return is being done. This may make it unnecessary to use the printer busy input.

If I am hooked up to a two-way device like a modem, can I send and receive at the same time?

Sure. A program like that is just a little harder to write.

So what is RS232, anyway?

It's a way of communicating, not just to a printer but to mainframes and other computers, to time bases or appliance controllers, to plotters or local networks. It's your computer's link to the outside world.



Telewriter-64

the Color Computer Word Processor

- 3 display formats: 51/64/85 columns x 24 lines
- True lower case characters
- User-friendly full-screen editor
- Right justification
- Easy hyphenation
- Drives any printer
- Embedded format and control codes
- Runs in 16K, 32K, or 64K
- Menu-driven disk and cassette I/O
- No hardware modifications required

THE ORIGINAL

Simply stated, Telewriter is the most powerful word processor you can buy for the TRS-80 Color Computer. The original Telewriter has received rave reviews in every major Color Computer and TRS-80 magazine, as well as enthusiastic praise from thousands of satisfied owners. And rightly so.

The standard Color Computer display of 32 characters by 16 lines without lower case is clearly inadequate for serious word processing. The checkerboard letters and tiny boot give you no feel for how your writing looks or reads. Telewriter gives the Color Computer a 51 columns by 24 line screen display with *true lower case characters*. So a Telewriter screen looks like a printed page, with a good chunk of text on screen at one time. In fact, more on screen (no) than you'd get with Apple II, Atari, TI, Vic or TRS-80 Model III.

On top of that, the sophisticated Telewriter full-screen editor is so simple to use, it makes writing fun. With single-letter mnemonic commands, and menu-driven I/O and formatting, Telewriter surpasses all others for user friendliness and pure power.

Telewriter's chain printing feature means that the size of your text is never limited by the amount of memory you have, and Telewriter's advanced cassette handler gives you a powerful word processor without the major additional cost of a disk.

...one of the best programs for the Color Computer I have seen.
— Color Computer News, Jan. 1982

THE WORLD'S BEST

But now we've added more power to Telewriter. Not just bells and whistles, but major features that give you total control over your writing. We call this new supercharged version Telewriter-64. For two reasons.

64K COMPATIBLE

Telewriter-64 runs fully in any Color Computer (16K, 32K, or 64K, with or without Extended Basic, with disk or cassette or both. It automatically configures itself to take optimum advantage of all available memory. This means that when you upgrade your memory, the Telewriter-64 text buffer grows accordingly. In a 64K cassette based system, for example, you get almost 40K of memory to store text. So you don't need disk or FLEX to get all your 64K to work immediately.

80 COLUMNS AND MORE

Besides the original 51 column screen, Telewriter-64 now gives you 2 additional high density displays: 64 x 24 and 85 x 24!! Both high density modes provide all the standard Telewriter editing capabilities, and you can switch instantly to any of the 3 formats with a single control key command.

The 51 x 24 display is clear and crisp on the screen. The two high density modes are more crowded and less easily readable, but they are perfect for showing you the exact layout of your printed page, *all on the screen at one time*. Compare this with cumbersome "windows" that show you only fragments at a time and don't even allow editing.

HIGH RESOLUTION JUSTIFICATION

One outstanding advantage of the full-width screen display is that you can now set the screen width to match the width of your printed page, so that "what you see is what you get." This makes exact alignment of columns possible and it makes hyphenation simple.

Select short lines are the reason for the large spaces often found in standard right justified text, and since hyphenation is the most effective way to eliminate short lines, Telewriter-64 can now prompt you some of the best looking right justification you can get on the Color Computer.

FEATURES & SPECIFICATIONS

Printing and Formatting: Drives any printer (LPII/VI, DMP 100/200, Epson, Okidata, Centronics, NEC, C. Itoh, Smith-Corona, Newmarc, etc)

Embedded control codes give full alphabetic access to multiple printer formats like underlining, subscript, superscript, variable line and type size, dot-graphics, etc.

Dynamic commands format overrides for top, bottom, and left margins; line length, lines per page, line spacing, new page, change page numbering, justification on/ off, double/ double pitch/ elite.

Menu-driven control of these parameters, as well as pause to align forms, page numbering, load text into the text buffer, save text buffer to tape or disk, and more.

Supports single and multi-line headers and automatic centering. Print or save all or any section of the text buffer. Chain print any number of files from cassette or disk.

File and I/O Features: ASCII format files — create and edit BASIC, Assembly, Pascal, and C programs, Issue Terminal Files (for uploading or downloading), even edit files from other word processors. Compatible with spelling checkers like Spell 'n Fix.

Cassette verify command for tape users. Cassette owners rejoice: you enter a load command only once no matter where you are in the tape.

Read in, save, partial save, and append files with one or two cassette. Put data from directory with free space to screen or printer. Kill and rename files, set default drive. Easily recommend to the number of drives in the system.

Editing features: Fast, full screen editor with wordwrap, block copy, block move, block delete, line delete, find/search and replace (or delete), wild card option, find next/previous cursor, line scrolling, cursor up/ down/ left/ right, home, clear, and beep, top of text, bottom of text, tab, backspace, page back/ward, skip/ save, choose of half or entire background, complete cursor positioning, line character, word erase, speech left, cursor file name, search next to edit, set line length on screen.

Insert or delete any quantities on the screen without changing "modes." The five "free-form" editing provides maximum ease of use. Everything you do appears immediately on the screen in front of you. Commands execute only a single key or a single key plus CLEAR.

...truly a state of the art word processor... outstanding in every respect!
— The RAINBOW, Jan. 1982

PROFESSIONAL PAGED PROCESSING

You can no longer afford to be without the power and efficiency word processing brings to everything you write. The TRS-80 Color Computer is the lowest priced micro with the capability for serious word processing. And only Telewriter-64 fully unleashes that capability.

Telewriter-64 costs $49.95 on cassette, $59.95 on disk, and comes complete with over 70 pages of well-written documentation. (The step-by-step tutorial will have your writing with Telewriter-64 in a matter of minutes.)

To order, send check or money order to:

Cognitec
704 Nob Street
Del Mar, CA 92014

Or check your local software store. If you have questions, or would like to order by Visa or Mastercard, call us at (714) 755-1258 (weekdays, 8AM-4PM PST). Dealer inquiries invited.

CLOAD "CHEKBOOK"

by Richard White

Why write a program that can do what a calculator can do? The best answer may be that it is a way to develope programming skills and reading this and keying in the program won't hurt either. Programs develope in funny ways. This one started to grow in my mind when I couldn't balance the checkbook against the bank statement. It turned out that my wife had pulled out one check to get a reimbursement before I got around to checking it off in the checkbook. The program helped me discover that. The program runs in 16K of memory and requires Extended Color Basic. It allows you to enter deposits, checks, adjustments including void checks, deletions, additions and service charges. Other features are; review and edit all entries, save cassette files, mark tax deductable checks and add notes such as payee, what the check was for and the like. At tax time next year, I will simply load in the monthly files from the cassette and review the entries listing the deductions as I go. The whole year will be on a single cassette.

When you first start, you enter the previous balance and first check number if you have not entered a cassette file. After that, the program will automatically present the next check number in order unless you type a different one in. It will automatically reuse the date and amount of the last check if you do not enter these. You may omit the decimal and zeros if an even dollar amount. The program reprints the amount, formatted with the PRINT USING command, and the new balance. Type a T in the last column to indicate a tax deductable item. Finally you are given the note option. A menu of choices and their letter codes is included in the program and can be recalled by typing M <ENTER> at the check entry point of the entry sequence. The menu choices are selected at the same time by typing their key <ENTER>.

Type R <ENTER> to get into the review and edit mode. When the program asks what check number to start with enter one that is before the one you want that you are sure is in the file. The program displays each number as it searches backwards for the target number. You may now review the file pressing <ENTER> for each new entry, or E for Edit, A for Add or D for Delete. In Edit the <ENTER> key is used to step across the line allowing you to enter new data only where needed. A inserts a new entry and D deletes an entire entry. The program will recalculate all the new balances with each change and presents the next entry for review.

Typing F <ENTER> will get you into the File mode. When outputting to tape you are asked for the first and last months in the file and the year, it will then make up a file name, prints it, writes two copies to tape and offers the option to make a backup on a second tape.

The other menu choices are "D" for deposit, "A" for adjustment and "V" for voided check. For a deposit, "DEP" is printed instead of a check number and the amount is added to the last balance rather than subtracted. The adjustment allows for all other transactions like service charges, check charges and perhaps interest in a NOW account. The void check entry permits keeping the check number on file, but automatically enters a $0.00 amount.

I thought I would really get organized before starting to type the program in. I generated a flow chart, defined subroutines, blocked out sets of line numbers for each and put a lot of programs on paper before entering it into the computer. Table 1 outlines the program structure as it finally evolved. The approach worked well up to the edit and review section which grew out of the alloted space and had to be continued above the cassette subroutines. Lines 1-99 are allocated to start-up functions and frequently used general purpose subroutines.

A word about the general utility subroutines. These include such things as the INKEY$ sub, the timer, centering, cassette positioning and others. When I start a program I load a tape with these subroutines and go from there. They become a part of each program and always have the same line numbers, low ones, that are easily remembered and save a byte of two of memory versus numbers over 100 or 1000 each time used.

The menu starts at line 100, a key number to remember. If you get an error or break the program and want to pick up where you left off type GOTO 100and the program will resume without losing any variables. RUN would initialize the program

losing all data previously entered. Data entry subroutines run from line 200 to 370. Since I have tried to use subroutines as much as possible, some lines are a collection of GOSUBs. This makes the program harder to follow but sure saves memory.

I find formatting the screen, particularly when data lines are to be scrolled while maintaining a heading one of programming's harshest punishments. The screen formatting subroutines in lines 410 to 440 are worth some study for ideas you can incorporate into your own programs. Line 410 starts with three PRINT commands that scroll whatever is on the screen up three lines, leaving three blank lines at the bottom. PRINT@0 is then used to print the column headings at the top of the screen. Another PRINT command clears the line under the headings. Then a PRINT@383,"" command positions the cursor at the start of the third line up from the bottom for entry of a check number or menu selection.

Depending of what the next transaction is, CK$ in line 415 may be check number, DEP for deposit or ADJ for adjustment. This is printed with a few blank spaces to assure there is nothing else in the space. PRINT@390,"" moves the cursor for the start of date entry at location 391.

The PRINT@396,"" in line 420 positions the cursor for amount entry. The amount is entered as you would into a calculator with the numerals printing from left to right. Obvious the right end will vary with the size of the number. PRINT USING corrects this, reprinting the entry right justified with two figures after the decimal. If you have entered an even dollar amount without the decimal point and two following zeros, PRINT USING supplies these as well as the dollar sign. The amount itself is contained in a string variable QN$ for filing and retrieval purposes and must be converted to a number using VAL(QN$).

After I had written and debugged the program, I felt that these screen formatting subroutines provided a fairly economical and effective solution for an effective screen display. Now I see some things that are either unnecessary or could be done differently to save memory and to make the program easier to follow. The blank spaces after the string variables are probably unnecessary. More wasteful is line 440 which could have been included in line 435, saving GOSUB statements elsewhere in the program. Debugging a program has two parts, getting the program to work the way you want and then cleaning up the garbage. Lines 415 and 416 are an example. They are so similar that one is unnecessary. 416 was added to solve some problem during debugging, but there must be a better solution.

Another form of program garbage are lines that are not used at all. These arise when you solve a problem by writing some new lines elsewhere and sending the program to them while forgetting to remove the old lines. Not only do these waste memory, they cause confusion when you come back later to improve the program. Try to reserve some time at the end of a debugging session to clean things up.

String variables provide an efficient way to store, file and retrieve both numeric and alphabetic information. Remember that check number, date, amount and balance were in string variables rather than in numeric variables. We see why in line 630 where these variables are simply added with $ separators to form a single variable SN$(K) that holds all the information for a single entry. The variables CK$, DA$, BA$, QN$, TD$ and PY$ are available to be reused for the next entry. There is considerable memory savings versus putting the data into a multiple variable array. The cost is the memory used for the lines to make the string SN$(k) and to later break it up. In this case, I estimate the savings to appear when there are more that 10-15 entries.

Line 650 to 695 are the string disassembly subroutine, the IF - THEN statement in line 650 catches a string containing no data before an FC error occurs in line 675. This was added during debugging and there probably is a better solution. In line 675 the program finds the location in the string of each of the $ data separators. Lines 680 and 690 use LEFT$ and MID$ statements to put the data into it's respective string variables for display. The first time through string disassembly tries one's patience. Once you get the hang of it you will find it rather elegant and satisfying.

At this point in our adventure, we have successfully waded the marsh of string disassembly and find ourselves facing the odorous swamp of review and edit. The primary problem was to get the balance changes in subsequent strings right when

there was an amount changed added or deleted. Use the Program Structure Summary in Table 1 to help guide you through a detailed study.

There are a number of features of the cassette file saving that deserve comment. Lines 870 and 880 input the start month, SM end month EM and the year for the file. In line 880 a file name is made from these entries which line 885 displays. The variable JK is set to 0 here also. In line 890 the file is opened, KJ is set to 0, JK is set to 1 indictating this is the first file save. In line 900, the program starts looking at the date of each SN$(KJ) until it finds the first one for the start month. When the desired SN(KJ) is found, the program goes to 930 to print it to the cassette. The program returns to 900 to get the next SN$(KJ). It tests the date to see that it is not greater than the end month and checks that the last entry with date has not been printed before printing the entry to cassette. If either of the above conditions are met the program goes to 910 where the file is closed and JK is checked to see if this was the first save. If so then a FOR TO NEXT loop is used to time about two seconds of blank tape between saves and the program returns to 890 to do it again. If JK=2, the option to record the file on a backup tape is offered in line 920.

It is always good practice to save a file twice and critical data should be on a backup tape. Borrow the techniques in 885 to 940 to do this in your programs. The file input routine, lines 950-980, are usual using "N"as a count variable and EOF(-1) to find the end of the file.

The last item I want to mention is the PCLEAR 1 in line 10000. From line 1 the program goes to the very last line for PCLEAR 1 and then returns to line 10. This avoids an SN ERROR after the first RUN command after loading and was discussed in a previous COLOR COMPUTER NEWS. It does not avoid an FC ERROR of the computer has not been turned off after running a machine language program and before loading CHECKBK.

TABLE 1

LINES	FUNCTION
1-10	Clear and dimensions
11-22	Utility Subroutines
90-95	File Input Choice & Initial Balance
100-180	Menu & Choice Selection
200-240	Check Input Routine
250-270	Deposit Entry Routine
300-310	Adjustment Routine
350-370	Void check routine
400-440	Screen Format Routines
530-550	Date Subroutine
560-580	Amount Subroutine
590-610	Balance Subroutine
620-630	String Assembly Subroutine
650-695	String Disassembly Subroutine
700	Start Review & Edit Routine
705-715	Find starting Check
720-740	Review Entries Loop
750-770	Edit, Addor Delete Selection
770-795	Start Delete Routine- Continues 1060
800-817	Add line Routine- Continues at 1060
820-830	Edit Mode Balance Adjustment- Continues at 1000
850-860	File Input or Output Selection
870-940	File Output Routine
950-980	File Input Routine
1000-1050	Edit Routine Concluded
1060-1090	String Balance Correction
10000	PCLEAR 1

```
1 GOTO10000
10 CLEAR5000:DIMSN$(100):GOTO90
11 Z=(32-LEN(ZT$))/2:PRINTTAB(Z)
   ZT$:RETURN
17 IFZS*60)TIMER THEN 17 ELSE RE
   TURN
18 PRINT"****TO PROCEED TOUCH AN
   Y KEY****"
19 Z$=INKEY$:IFZ$()""THEN RETURN
   ELSE19
20 PRINT"TO SET TAPE RECORDER AN
   D POSITION TAPE TO SAVE OR LOAD,
   PRESS ANY KEY FOR MOTORON ON AN
  DTHEN ANY KEY FOR MOTOROFF"
21 Z1$=Z$:GOSUB19
22 AUDIOON:MOTORON:GOSUB19:MOTOR
   OFF:Z$=Z1$:RETURN
30 CLS:ZT$="MENU":GOSUB11:ZT$="P
   RESS (KEY) INDICATED":GOSUB11:PR
  INT:PRINT"(1ST DIGIT OF CHECK#)
  THEN REMAINING DIGITS AFTER '?'
  TO START NEW CHECK SEQUENCE":PRI
  NT"(ENTER) FOR NEXT CHECK# IN SE
  QUENCE AUTOMATICALLY"
35 PRINT"(D) FOR DEPOSIT":PRINT"
   (A) FOR ADJUSTMENT":PRINT"(V) FO
  R VOIDED CHECK":PRINT"(R) FOR RE
  VIEW AND EDIT":PRINT"(F) FOR LOA
  D OR SAVE FILE":PRINT"(M) FOR ME
  NU";:RETURN
```

```
90 CLS:PRINT:PRINT"PRESS (F) IF
   TAPE FILE ELSE ANY KEY":GOSUB19:
  IFZ$="F" THEN Z$="I":GOSUB20:PRI
  NT:PRINT"SET RECORDER TO PLAY &
  PRESS ANYKEY":GOSUB19:GOTO950
95 GOSUB410:PRINT@352,"ENTER INI
   TIAL DATE AND BALANCE":CK$="BAL"
  :GOSUB415:GOSUB530:PRINT@404,"";
  :LINEINPUTBA$:GOSUB430:GOSUB435:
  GOSUB440:N=N+1:GOSUB620:GOSUB30:
  GOSUB19:CK=VAL(Z$):CLS:BA=VAL(BA
  $):CLS:GOSUB410:GOTO120
100 POKE65494,0:GOSUB30:GOSUB19:
   CLS:FG=0:CK=VAL(Z$):GOSUB410:GOT
  O120
110 IFE=1 THEN 720
115 GOSUB410:LINEINPUTZ$:CK=VAL(
   Z$)
120 IFZ$="" THEN 200ELSEZ=ASC(Z$
   ):IFZ=13 THEN 200
125 CK=VAL(Z$):IFCK)0 THEN 200
130 IFZ$="D" THEN 250
140 IFZ$="A" THEN 300
150 IFZ$="V" THEN 350
160 IFZ$="R" THEN 700
170 IFZ$="M" THEN 100
180 IFZ$="F" THEN 850ELSE110
200 'ENTER CHECK
205 IFCK=0 THEN 240ELSECK$=Z$:PR
   INT@384,Z$;
210 LINEINPUTZ$:CK$=CK$+Z$:CK=VA
   L(CK$)
215 CH$=CK$:GOSUB415
220 GOSUB530:GOSUB560:GOSUB590:L
   INEINPUTZ$:IFZ$="T" THEN TD$=Z$E
  LSETD$=""
230 GOSUB435:GOSUB440:N=N+1:K=N:
   GOSUB620:GOTO110
240 CK$=STR$(VAL(CH$)+1):CH$=CK$
   :CK=VAL(CK$):GOSUB416:GOTO220
250 'ENTER DEPOSIT
260 CK$="DEP":GOSUB415:GOSUB530:
   GOSUB560:GOSUB270:GOSUB590:GOSUB
  270:TD$="   ":GOTO230
270 QN$=STR$(-VAL(QN$)):RETURN
300 'ENTER ADJUSTMENT
310 CK$="ADJ":GOSUB415:GOSUB530:
   GOSUB560:GOSUB270:GOSUB590:GOSUB
  270:TD$="   ":GOTO230
350 'VOID CHECK
360 LINEINPUTZ$:IFZ$="" THEN CK$
   =STR$(VAL(CH$)+1)ELSECK$=Z$
370 GOSUB415:GOSUB530:QN$="VOID"
   :GOSUB425:TD$="   ":CH$=CK$:GOTO2
  30
400 'SCREEN FORMAT SUBS
410 PRINT:PRINT:PRINT:PRINT@0,"C
   HECK# DATE AMOUNT  BALANCE TAXD"
  :PRINT:PRINT@383,"":RETURN
415 PRINT@384,CK$+"  ":PRINT@390
   ,"";:RETURN
416 PRINT@383,CK$+"  ":PRINT@390
   ,"";:RETURN
420 PRINT@390,DA$+"  ":PRINT@396
   ,"";:RETURN
425 PRINT@395,"";:PRINTUSING"$$#
   ###.##";VAL(QN$):PRINT@405,"";:R
  ETURN
430 PRINT@404,"";:PRINTUSING"$$#
   ###.##";VAL(BA$):PRINT@414,"";:R
  ETURN
435 PRINT@414,TD$+"  ":PRINT@416
   ,"NOTE: ";:RETURN
440 PY$="":LINEINPUTPY$:RETURN
530 'DATE SUB
540 LINEINPUTZ$:IFZ$="" THEN GOS
   UB420:RETURN
550 DA$=Z$:GOSUB420:RETURN
560 'AMT SUB
570 LINEINPUTZ$:IFZ$="" THEN GOS
   UB425:RETURN
580 QN$=Z$:GOSUB425:RETURN
590 'BALANCE SUB
600 BA=BA-VAL(QN$)
610 BA$=STR$(BA):GOSUB430:RETURN
620 'STRING ASSEMBLY
630 SN$(K)=CK$+"$"+DA$+"$"+QN$+"
   $"+BA$+"$"+TD$+"$"+PY$:RETURN
650 IFSN$(K)="" THEN RETURN:'STR
   ING DISASSEMBLY*********
660 LS=LEN(SN$(K)):K1=0:L(0)=0:P
   Y$=""
675 K1=K1+1:L(K1)=INSTR(L(K1-1)+
   1,SN$(K),"$"):IFL(K1))0 THEN 675

680 CK$=LEFT$(SN$(K),L(1)-1):DA$
   =MID$(SN$(K),L(1)+1,L(2)-L(1)-1)
  :QN$=MID$(SN$(K),L(2)+1,L(3)-L(2
  )-1)
690 BA$=MID$(SN$(K),L(3)+1,L(4)-
   L(3)-1):TD$=MID$(SN$(K),L(4)+1,L
  (5)-L(4)-1):IFLS)L(5) THEN PY$=M
  ID$(SN$(K),L(5)+1,LS-L(5))ELSERE
  TURN
695 RETURN
700 'EDIT SUB
705 CLS:INPUT"STARTING CHECK #";
   C8:K=N:E=1:IFC8=0 THEN705:POKE65
  495,0
710 K=K-1:IFK=0 THEN PRINT"CHECK
    # "C8" NOT IN FILE":E=0:POKE654
  94,0:GOTO100
715 GOSUB650:GOSUB415:CK=VAL(CK$
   ):IFC8()CK THEN 710
720 GOSUB410:PRINT@32," PRESS (E
   ) TO EDIT-":PRINT@64,"  PRESS (A
  ) TO ADD":PRINT@96,"   PRESS (D)
   TO DELETE":PRINT"ANY OTHER KEY
  TO PROCEED":PRINT
722 GOSUB415:GOSUB420:GOSUB425:G
   OSUB430:GOSUB435:PRINTPY$:POKE65
  494,0:GOSUB19:IFZ$="E"ORZ$="A"OR
  Z$="D" THEN 750
```

```
725 IFVAL(CK$))0 THEN CH$=CK$
730 IFK=N THEN PRINT"THIS IS LAS
   T ENTRY IN FILE":BA=VAL(BA$):GOS
  UB18:E=0:GOTO100
740 K=K+1:GOSUB650:GOTO720
750 GOSUB410:CK=VAL(Z$):IFCK)00R
   Z$="E" THEN 820
760 POKE65495,0:IFZ$="A" THEN 800
770 IFZ$()"D" THEN PRINT"INVALID
    ENTRY":GOTO720
780 JK=K:N=N-1:K=K-1:GOSUB650:BA
   =VAL(BA$):QN$=""
790 SN$(JK)=SN$(JK+1):JK=JK+1:IF
   JK(N THEN 790ELSEJ=K:GOTO1060
795 K=K-1:J=K:GOTO1060
800 N=N+1:K=K+1:JK=N:BA=VAL(BA$)
   :CK=VAL(CK$)
810 SN$(JK)=SN$(JK-1):JK=JK-1:IF
   JK)K THEN 810ELSEGOSUB410:POKE65
  494,0:LINEINPUTCK$:GOSUB415:GOSU
  B530:GOSUB560:CK=VAL(CK$):IFCK)0
   THEN BA=(BA-VAL(QN$))ELSEBA=(BA
  +VAL(QN$))
815 BA$=STR$(BA):GOSUB430:LINEIN
   PUTZ$:IFZ$="T" THEN TD$="T"
817 GOSUB435:LINEINPUTPY$:GOSUB6
   20:GOSUB660:BA=VAL(BA$):J=K:GOTO
  1060
820 IFVAL(CK$))0 THEN BA=(VAL(BA
   $)+VAL(QN$)):GOTO1020
830 BA=VAL(BA$)-VAL(QN$):GOTO1020
850 'FILE SUB
855 CLS:PRINT:PRINT"KEY IN (I) T
   O INPUT FROM TAPE          (O) TO
   OUTPUT TO TAPE":GOSUB19:GOSUB20
  :CLS:PRINT:IFZ$="I" THEN PRINT"S
  ET RECORDER TO PLAY & PRESS ANYK
  EY":GOSUB19:GOTO950
860 IFZ$="O" THEN PRINT"SET RECO
   RDER TO RECORD & PRESS  ANY KEY"
  :GOSUB19
870 CLS:PRINT:PRINT"ENTER START-
   MONTH AND END-MONTH":PRINT:PRINT
  "ALL TRANSACTIONS IN THESE MONTH
  SWILL BE RECORDED"
880 INPUT"START-MONTH NUMBER ";S
   M$:INPUT"END-MONTH NUMBER ";EM$:
  INPUT"YEAR ";YR$:IFEM$=SM$ THEN
  NF$=SM$+"/"+YR$ELSENF$=SM$+"-"+E
  M$+"/"+YR$
885 PRINT:PRINT:PRINT"PLEASE WRI
   TE DOWN FILE NAME":PRINT:PRINT"
      "NF$:GOSUB18:EM=VAL(EM$):SM=V
  AL(SM$):JK=0
890 OPEN"O",-1,NF$:KJ=0:JK=JK+1
900 KJ=KJ+1:GOSUB650:DA=VAL(LEFT
   $(DA$,2)):IFDA(SM THEN 900
904 IFKJ)N THEN 910
906 IFDA)EM THEN 910ELSE930
910 CLOSE-1:IFJK=1 THEN MOTORON:
   FORKJ=1TO1000:NEXT:GOTO890
920 IFJK=2 THEN CLS:CLOSE-1:PRIN
   T"PRESS (Y) TO SAVE FILE TO BACK
  UPTAPE":GOSUB19:JK=0:IFZ$="Y" TH
  EN 940ELSE100
930 PRINT#-1,SN$(KJ):GOTO900
940 GOSUB20:CLS:PRINT"SET RECORD
   ER TO RECORD & PRESS  ANY KEY":G
  OSUB19:GOTO890
950 PRINT:INPUT"ENTER FILE NAME"
    ;NF$:OPEN"I",-1,NF$:N=0
960 IFEOF(-1) THEN 980ELSEN=N+1
970 INPUT#-1,SN$(N):GOTO960
980 CLOSE-1:PRINT"PRESS (R) TO R
   EVIEW, ANY OTHER KEY TO ADD ENTR
  IES":GOSUB19:IFZ$="R" THEN CLS:K
  =1:E=1:GOTO720
1000 'ENTRY EDIT**********
1010 IFCK)0 THEN CK$=Z$
1015 CK=VAL(CK$)
1020 GOSUB410:PRINT@33,"IN EDIT
   MODE":GOSUB415:GOSUB530:GOSUB560
  :QN=VAL(QN$):IFVAL(CK$))0 THEN B
  A=(BA-QN)ELSEBA=(BA+QN)
1030 BA$=STR$(BA):GOSUB430:LINEI
   NPUTZ$:IFZ$="T" THEN TD$="T"
1040 GOSUB435:LINEINPUTZ$:IFZ$()
   "" THEN PY$=Z$
1050 GOSUB620:GOSUB660:IFQO$=QN$
    THEN 740
1055 BA=VAL(BA$):J=K
1060 POKE65495,0'STRINGS BALANCE
    CORRECTION**********
1070 K=K+1:GOSUB650:QN=VAL(QN$):
   IFVAL(CK$))0 THEN BA=(BA-QN)ELSE
  BA=(BA+QN):POKE65494,0
1080 BA$=STR$(BA):GOSUB620:IF K(
   N THEN1070
1090 K=J:GOSUB650:GOTO720
10000 PCLEAR1:GOTO10
```

Micro Works Disassembler
by Mark Rothstein

I recently received a copy of the Micro Works disassembler. I was so impressed with it that I decided to write this review article. In this review I'm going to discuss:

What Disasm is,
My impressions of the program,
Reasons why I find it useful and how you may use it,
Some of it's varied output features,
A difficulty I had with DISASM and how I solved it.

What is it?

DISASM, a cassette program for the Color Computer written by Andy Phelps, disassembles computer programs that are in RAM or ROM. It requires 16K of RAM and will output a readable listing of a machine language program to either the TV screen or to a printer.

My first impression.

When I first ran the disassembler, I was really surprised. I had written a disassembler for the 8080 a few years ago and I remembered all the features I had either put in, or wanted to put in. DISASM has them all. It will output a listing similar to the source listing of an assembler ... and with cross references. Or it can output a version that can be reassembled with another program.

This 6809 disassembler is a good finished product. It worked as advertised, the first time I ran it. The version I got came with a good set of instructions with examples of different operating modes. These instructions are clear, but I didn't completely understand them at first because there are so many different output modes. Anticipating this, Micro Works built in a set of defaults. With these defaults, when the program asks a question, an "I don't know" answer is acceptable. In their ads and articles in Color Computer News, Micro Works has been suggesting that Color Computer owners disassemble their copy of the BASIC ROMs. "I don't know" answers to all questions automatically cause the program to disassemble the BASIC ROM at addresses $A000 to $BFFF (where the "$" indicates that the value which follows is in hexadecimal). The disassembly listing scrolls up on the TV display. While the program is running, the scrolling can be adjusted or switched to a single step mode; the listing can be stopped and restarted at any location within the BASIC address range.

Reasons why DISASM is a very useful tool.

Now have you ever wondered how Microsoft (they are the people who wrote Color Computer BASIC) programmed the Color Computer to store your program in BASIC, to read the joysticks, to play music or to paint colors? Well DISASM will HELP you figure out how they've done it. Even if you are a seasoned programmer, this is occasionally a rewarding task -- once in a while you'll learn a new trick. When I find a new project which is related to something already done in BASIC, a question I ask is "How does BASIC do it?" and "Are there any idiosyncrasies in the Color Computer that I'd better take note of?".

A specific case in point is my current project; the design of an inexpensive but reliable bar code reader. This will allow Color Computer owners to load programs directly from Color Computer News. (See the whole series of articles in Byte from November 1976 to May 1978, and the Bar Code Loader from Byte publications.) The easiest place to attach a bar code scanner is at the joystick connector. BASIC can scan the black and white bars, but it does this much too slowly; consequently a machine language program must be devised to do the scanning. What does the BASIC joystick code look like then? The answer is to look at addresses $A9A2 thru $AA19. (Incidentally, the Micro Works documentation includes a list of the addresses of many useful function areas in BASIC.) This time an examination of BASIC didn't turn up anything useful, but at other times, it does.

Micro Works Disassembler

Other Example Uses.

The disassembler can even be made to operate on itself. This isn't necessary, though. A complete source listing of the program comes with the documentation.

There's also a very practical use for DISASM. When I generate a machine language program, in testing it I sometimes find that it doesn't work the first time. As I isolate errors, I fix them with patches -- tacked on sections of code. Then I test the patch and move on to other errors. You're supposed to immediately correct the source code and reassemble. I find this a chore, and practically I only reassemble when I have to fix a serious error or I've got a lot of patches and I'm at the end of a work session. Sometimes accidentally a patch doesn't make it into the next assembled version. This can make the next debugging session quite tedious. But now with DISASM, this doesn't happen anymore. I keep a copy of DISASM co-resident in memory with my machine language program and when I make a patch, I document it quickly and easily -- I disassemble it. This technique has been very effective. It's also helped me find fatal flaws in my hand assembled patches. (More on this next time when I review Micro Works Assembler Editor.)

The many output formats.

Now on to the DISASM output formats which deserve comment. DISASM is designed for both narrow (32 columns like the TV and some printers) and wide (64 columns or more) output devices. In order to output all the important information in the program listing, 64 columns are required. In wide output case, the listing may appear as shown in listing 1.

This is an excerpt from a BASIC disassembly. The first block of lines are external references: Note $C000, the address of the program and game cartridges, the addresses of the PIA's, etc. Next come the addresses of the RAM variables (indicated by the "V" label). After that, if you've read Getting Started with Color Basic, you'll recognize the addresses of some of the BASIC routines (with the typos removed) -- Get keyboard data, Output a character, Start the cassette, etc. See pages 269 and 270 in the Getting Started manual.

Now for the columns: Take for example, te instruction line $A012.

```
A012  8637   --  --  .7  -- LDA #37 "7"
```

The first column is the instruction addresses, then comes the object code (8637), then two blank columns where reference information is listed. Column five has the ASCII equivilent of the operand if it is text and is an immediate value (as represented by the "#" symbol). Now for the columns three and four, examine the BASIC addresses $A01B and $A023:

```
A01B  2651  A06E  .... &Q --- BNE PA06E
A023  2649  A06E  A01B &I --- BNE PA06E
```

Column three has the address referenced by this instruction (if any). (Normally, in assembler output, the label PA06E would have a symbolic name and the address reference would not be obvious.) To explain column four I must say that the 64-column output produces a listing which contains all addresses explicitly referenced together with the address of the highest reference. (This list is sorted numerically.) Column four of the highest address contains the address of the next higher location, and so on. The last reference has a "...." in column four. This is shown at address $A01B.

The narrow format.

If the 32 column display field is used, this output will take up two lines -- this is much less readable. One alternative is to suffer. The other is to omit some of the fields. Five other variations are available, and the user may switch between them "on-the-fly".

Also while the program is running, the user may vary the output speed on the TV display from unreadably fast, to fast to single step. These are conviently selected by the keyboard.

Micro Works Disassembler

My problem with DISASM.

The only problem I had with DISASM concerns the output routine. The ease with which I was able to fix the problem is indicative of the thought that went into DISASM design. Micro Works claims that DISASM is compatable with any printer that will run with BASIC. This is correct -- they use the BASIC printer output routine! My printer, (TI Silenttype) however, is not quite compatable -- it doesn't linefeed automatically after a carriage return. Radio Shack has told me how to patch BASIC, but DISASM requires a different patch. Here are the key ideas in the solution: First DISASM is written in position independent code; located anywhere in memory; it will execute. Not like any of the Radio Shack ROM Paks which are intentionally only a little bit position-dependent. (This is something like being a little bit pregnant.) Second, DISASM make only one direct call to the BASIC printer driver. This is in the beginning of the program where it is easy to find and modify. Therefore a patch in only one place will solve the problem. The solution is shown in Listing 2. DISASM has been relocated to $4020 and the patch resides at locations from $4000 to $401F. The patch also sets The baud rate to 300 Baud. Not that the patch is also position independent.

In Summary.

In summary, I believe that a serious user of the Color Computer will find the Micro Works disassembler to be a useful and efficient tool to have around -- from disassembling BASIC to maintaining documentation of his own programs.

Listing 1. An excerpt from a BASIC Disassembly.

```
                                    NAM   DISASM
                                    ORG   $0000
                               XC608 EQU   $C608
                               XC60E EQU   $C60E
                               XFFC9 EQU   $FFC9
                               XFFFF EQU   $FFFF
                               V0000 RMB   1
                               V0001 RMB   1
A00    A1C1      A1C1 A000   !A      AA000 FDB   PA1C1
002    A282      A282 A002   ".            FDB   PA282
004    A77C      A77C A004   '|            FDB   PA77C
006    A70B      A70B A006   '.            FDB   PA70B
008    A7F4      A7F4 A008   't            FDB   PA7F4
00A    A9DE      A9DE A00A   )^            FDB   PA9DE
00C    A7D8      A7D8 A00C   'X            FDB   PA7D8
```

Listing 2. Patch to print CR with LF.

```
001 0600                X4024   EQU $4024
002 0600                XA2BF   EQU $A2BF
003 0600                V0000   RMB $0096
004 0696                V0096   RMB $3F6E
005 4604                        ORG $4000
006 4000 86B4           P4004   LDA #$B4        SET 300 BAUD
007 4002 9796                   STA <V0096
008 4004 201E                   BRA X4024       JUMP TO DISASM
009 4006 810D           CRLF    CMPA #$0D       CHECK FOR CR
010 4008 2605                   BNE P4013       NO? PRINT & RTS
011 400A BDA2BF                  JSR XA2BF       ELSE PRINT CR
012 400D 860A                   LDA #$0A        THEN PRINT LF
013 400F 7EA2BF         P4013   JMP XA2BF       GO TO BASIC
014 4012                        END   *         DRIVER
```

SIGMON

by Kenneth Kalish
85 Cooper St.
Pringle, PA 18704

If you're interested in writing some graphics subroutines or some short utilities, or if you'd just like to learn about 6809 assembly language, then the most economical answer is available from Datasoft, Inc. under the name of "Sigmon". It is billed as a combination monitor, mini-assembler, and debugger. Sigmon comes with a price tag of only $29.95, and Datasoft doesn't take all year to get it out to you, either. Everything goes in at once, and takes up about 6K or RAM.

Sigmon works, and it works well; but as with everything, there's good news and then there's bad news. The most serious drawback lies in the fact that you don't get a full fledged Editor/Assembler. You do get a mini-assembler, as advertised. That means that there is no source code, therefore no editing and no labels. With Sigmon, you assemble one line at a time, directly to memory. Having no labeling or editing capability can be quite a serious handicap when writing very long programs or routines with complex internal interactions. For example, if you want to insert an instruction in the middle of your assembled code, you can use the block MOVE command to make room for it, but you'll end up throwing off all of your relative addressing. You'll then have to go back and correct each one by hand. It also makes it pretty rough when you're branching ahead in memory to a spot you haven't written yet, You'll usually have to write a guess or a dummy branch, then come back later and fix it with the correct destination (or you'll find yourself structuring your programs to include mostly backward branches). It can all get to be a little aggravating, and a little confusing.

However, since you are programming closer to the ground, so to speak, you are liable to end up learning a lot more about some of the finer details of the language than you would otherwise. You also don't have to wait for assembly runs, which is natural enough since there is no source code to assemble. This is especially helpful when you only want to make one or two quick changes to a program. In addition, you get instant notice of assembly errors; and since there is no source code storage space, you'll have more RAM to work with.

The usual procedure would be to assemble your program to memory, and next use the disassembler to check the code that you use put in. (The disassembler also come in handy in exploring Color Basic or game pak ROMs.) You can then debug the program using break points or the built in single stepper. Sigmon's stepper works by determining the expected address after execution of the current displayed line, and then placing a temporary SWI instruction at the address of that expected destination. It therefore cannot be used through ROM areas.

The single stepper does work beautifully, displaying the register contents as well as the current address, memory contents and disassembled nmemonic for each instruction. However, as a single stepper, it does suffer from a serious malady. That is, after the twentieth time through a simple loop, you begin to feel more like a telegraph operator than a computer programmer, what with having to press a key for each line executed.

Fortunately, a short patch can be applied which provides a good solution. Load Sigmon into it's normal residence and say; "Physician, heal thyself". If that doesn't work, then assemble the following code at address $0FE7 (4071 dec.):

```
$0FE7 LDA 341
$0FEA ORA #8
$0FEC STA 341
$0FEF BRA $2528    Back to Sigmon
```

Then, connect the patch by assembling at $1A11 (6673 dec.);

```
$1A11 BSR $0FE7    To patch
```

and save it to tape with WRITE "SIG",$0FE7,$27F7,$0FF2
That's it. The patch provides a repeat key function for the up arrow key, by suppressing the action of the keyboard buffer for that particular key. (The keyboard buffer is the same one which is used by the keyboard scanning routine in Color Basic. The subroutine is in turn called by Sigmon.) You can now step along by holding down

the up arrow key; and you can vary the step rate by using Sigmon's own "SPEED" command (which imposes a variable delay whenever a carriage return is printed, and is intended to control listing speed).

So there you have it. Sigmon is a no-frills but very functional approach to assembly language programming, with the tradeoffs lying mainly in the advantages of cost and unity, versus a marked lack of ease and convenience in certain programming tasks. You ultimately could write anything with it, such as your own version of Space Invaders, but it isn't very well suited for writing long or complex programs. Still, the final judgement has to be that there is an awful lot of good stuff packed in there for only $29.95.

continued from page 8

```
468 PRINT@202,"DARTH VADER'S"
469 PRINT@234,"TIE FIGHTERS GOT
YOU!!";END
470 GOTO 400
500 SOUND 89,3:SOUND 147,3:SOUND
 147,3
510 FOR D=1 TO 100:NEXT D
520 SOUND 133,3:SOUND 125,3:SOUN
D 108,3
530 SOUND 175,3
540 FOR D=1 TO 100:NEXT D
550 SOUND 147,3:FOR D=1 TO 100:N
EXT D
560 SOUND 147,3:RETURN
800 CLS(0):SET(H,V,0)
805 SOUND 230,1:SOUND 220,1:SOUN
D 200,1:SOUND 180,1
810 FOR D=1 TO 40:NEXT D:CLS(0)
820 SET(H+2,V,0):SET(H-2,V,0):SE
T(H-1,V-1,0)
830 SET(H+1,V-1,0):SET(H-1,V+1,0
):SET(H+1,V+1,0)
840 FOR D=1 TO 40:NEXT D:CLS(0)
850 SET(H-4,V,0):SET(H+4,V,0):SE
T(H-2,V-2,0)
860 SET(H,V-3,0):SET(H+2,V-2,0):
SET(H-2,V+2,0)
870 SET(H,V+3,0):SET(H+2,V+2,0)
880 FOR D=1 TO 40:NEXT D:CLS(0)
890 RETURN
2000 CLS:PRINT@170,"GOOD SHOOTIN
G JEDI!!"
2010 PRINT@200,"THE FORCE WAS WI
TH YOU!
2020 END
```

METEOR

```
10 CLS:INPUT"INSTRUCTIONS";A$:IF
LEFT$(A$,1)="Y"THEN CLS:GOTO 210

20 PO=0:G=0:S$=CHR$(134)+CHR$(13
7):E$=CHR$(139)
25 L=1024
30 PRINT@G," ";:PRINT@RND(31)+48
0,"*":PRINT@0,PO;
35 R=JOYSTK(0):H=R/2
50 IF PEEK(L+H)=106 THEN 100
60 PRINT@H,S$;
70 IF PEEK(65280)=126 THEN 79 EL
SE IF PEEK(65280)=254 THEN 79 EL
SE 90
79 SOUND235,1
80 FOR V=1 TO 16:IF PEEK(1056+H+
(32*V))=106 THEN 130 ELSE IF V<1
4 THEN POKE1024+H+(32*V),139:IF
V<14 THEN POKE1024+H+(32*V),143:
NEXT V:IF V<16 THEN 80 ELSE 30
90 PO=PO+1:GOTO 30
100 PLAY"L255;V31;O1ADCFBAGED;V1
6;ACEGAD;V4EABCAEDB"
110 DK=DK+1:IFDK=3THEN170ELSE30
120 FOR I=1 TO 100:NEXT I:GOTO 1
70
130 PO=PO+100:SOUND50,1:POKE1056
+H+(32*V),255:SOUND50,1:POKE1056
+H+(32*V),143:GOTO 30
170 CLS:IF PO>HP THEN 260 ELSE P
RINT@32,HP$":";HP;:PRINT@64,"YOU
 EARNED "PO" POINTS ON THAT MISS
ION";
180 PRINT@224,;:INPUT"   DO YOU
WANT  TO  TRY ANOTHER MISSION";Y
$:IF LEFT$(Y$,1)="N"THEN END ELS
E 20
210 FOR I=1 TO 20:PRINT@RND(31)+
480,"*":NEXT:PRINTTAB(15);"BY HO
LLIS HOLCOMB                MOOR
E,OK."
220 PRINT:PRINT"YOUR MISSION IS
TO BLAST A PATH THROUGH A METEOR
 STORM.YOU STEERWITH THE RIGHT J
OYSTK.YOU GET   100 POINTS FOR E
ACH METEOR YOU  SHOOT.SHOOT WITH
 RIGHT JOYSTK   BUTTON."
230 INPUT"PRESS ENTER TO BEGIN";
A$:GOTO 20
260 HP=PO:PRINT@160,CHR$(128);"Y
OU GOT HIGH SCORE OF"HP:INPUT"EN
TER YOUR NAME";HP$:CLS:GOTO 180
```

6809 Machine Code
by Bill Sias

This issue we're again going to move away from writing any machine language and look instead at the disassembler written by Larry Ashmun and marketed by Soft Sector Marketing. SSM has donated this program for you to type in or, for the cover dates of this issue, you may purchase it directly from them for $9.95 (regular price is $14.95). The listing is long so I'll quit here and let you get right into it.

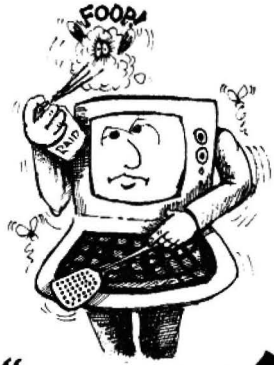
```
1 '        DISASSEMBLER 6809
2 '        COPYRIGHT (C) 1981
3 '   SOFT SECTOR MARKETING INC.
4 '
5 '       WRITTEN BY L. ASHMUN
6 '
7 '            VERSION  1.4
9 CLS
10 GOSUB9000
20 GOSUB9300
48 GOTO1100
49 ' special dec. to hex.
50 D5=INT(D4/16):D6=D4-D5*16
55 D$=H$(D5)+H$(D6)
60 RETURN
99 ' 1 byte dec. to hex.
100 D=PEEK(A):A=A+1
105 DH=INT(D/16):DL=D-DH*16
110 D$=H$(DH)+H$(DL):OP$=OP$+D$
120 IFD)32AND D(91THENDA$=DA$+CH
    R$(D)
130 RETURN
149 ' 2 byte dec. to hex.
150 IFAD(0ORAD)65535THENAD$="err
    or":RETURN
155 A1=INT(AD/4096):AI=AD-A1*4096
160 A2=INT(AI/256):AI=AI-A2*256
170 A3=INT(AI/16):A4=AI-A3*16
180 AD$=H$(A1)+H$(A2)+H$(A3)+H$(
    A4)
190 RETURN
199 ' immediate mode
200 PR$="#":FORZ=1TOBT
210 GOSUB100:PR$=PR$+D$:NEXTZ
220 RETURN
249 ' direct mode
250 GOSUB100:PR$=D$
260 RETURN
299 ' extended mode
300 FORZ=1TO2
310 GOSUB100:PR$=PR$+D$:NEXTZ
320 RETURN
349 ' 1 byte relative mode
350 GOSUB100
360 IFD)127THEND=D-256
370 AD=D+A:GOSUB150:PR$=AD$
380 RETURN
399 ' 2 byte relative mode
400 GOSUB100:D1=D:GOSUB100:D2=D
410 D=D1*256+D2
420 IFD)32767THEND=D-65536
430 AD=D+A:GOSUB150:PR$=AD$
440 RETURN
499 ' indexed modes
500 R$="":R1$="":GOSUB100:XZ=D A
    ND16
505 ON(D AND96)/32GOTO515,520,525
510 R$="X":GOTO530
515 R$="Y":GOTO530
520 R$="U":GOTO530
525 R$="S"
530 IF(D AND128))0THEN545
535 D4=D AND15:IFXZ)0THENR1$="-"
    :D4=ABS(D4-16)
540 GOSUB50:PR$=R1$+D$+","+R$:RE
    TURN
545 PB=D AND15
550 ONPB GOTO560,565,570,575,580
    ,585,590,595,600,590,605,610,615
    ,590,620
555 R$=","+R$+"+":IFXZ)0THEN590E
    LSE655
560 R$=","+R$+"++":GOTO655
565 R$=",-"+R$:IFXZ)0THEN590ELSE
    655
570 R$=",--"+R$:GOTO655
575 R$=","+R$:GOTO655
580 R$="B,"+R$:GOTO655
585 R$="A,"+R$:GOTO655
590 PR$="bad pbyte":RETURN
595 R$=","+R$:GOTO625
600 R$=","+R$:GOTO635
605 R$="D,"+R$:GOTO655
610 R$=",PC":GOTO625
615 R$=",PC:GOTO635
620 GOSUB300:R$=PR$:GOTO655
625 GOSUB100:D4=D:IFD)127THEND4=
    ABS(D-256):R1$="-"
630 GOSUB50:GOTO650
635 GOSUB100:DD=D:GOSUB100:AD=DD
    *256+D
640 IFAD)32767THENAD=ABS(AD-6553
    6):R1$="-"
645 GOSUB155:D$=AD$
650 R$=R1$+D$+R$
655 IFXZ)0THENPR$="("+R$+")" ELS
    EPR$=R$
660 RETURN
799 ' push/pull group
800 GOSUB100
805 IFD=0THENPR$="bad pbyte":RET
    URN
810 IF(D AND128))0THENPR$=",PC"
```

```
820 IF(D AND64))0THENPR$=PR$+",S
   /U"
830 IF(D AND32))0THENPR$=PR$+",Y"
835 IF(D AND16))0THENPR$=PR$+",X"
840 IF(D AND8))0THENPR$=PR$+",DP"
845 IF(D AND4))0THENPR$=PR$+",B"
850 IF(D AND2))0THENPR$=PR$+",A"
855 IF(D AND1))0THENPR$=PR$+",CC"
870 PR$=RIGHT$(PR$,LEN(PR$)-1)
875 IFPP=1THENPR$=PR$+T$ELSEPR$=
   T$+PR$
880 RETURN
899 ' transfer/exchange group
900 GOSUB100:X=0:J=(D AND240)/16
910 ONJ+1GOTO925,930,935,940,945
   ,950,920,920,955,960,965,970
920 PR$="bad pbyte":RETURN
925 PR$=PR$+"D":GOTO975
930 PR$=PR$+"X":GOTO975
935 PR$=PR$+"Y":GOTO975
940 PR$=PR$+"U":GOTO975
945 PR$=PR$+"S":GOTO975
950 PR$=PR$+"PC":GOTO975
955 PR$=PR$+"A":GOTO975
960 PR$=PR$+"B":GOTO975
965 PR$=PR$+"CC":GOTO975
970 PR$=PR$+"DP"
975 IFX=1THENRETURN
980 X=1:J=D AND15:PR$=PR$+",":GO
   TO910
999 ' hex. to dec.
1000 X=LEN(A$):X1=X+1:A=0
1010 IFX(1ORX)4THENRETURN
1020 FORZ=1TOX
1030 N=ASC(MID$(A$,X1-Z,1))
1040 IFN)47ANDN(58THENN=N-48:GOT
   O1070
1050 IFN)64ANDN(71THENN=N-55:GOT
   O1070
1060 N=0
1070 ONZ GOTO 1075,1080,1085,1090
1075 P=1:GOTO1095
1080 P=16:GOTO1095
1085 P=256:GOTO1095
1090 P=4096
1095 A=A+N*P:NEXTZ
1096 RETURN
1098 DATA 0,1,2,3,4,5,6,7,8,9,A,
   B,C,D,E,F
1099 ' program control
1100 CLEAR200:CLS
1105 T$="*":B=0
1110 DIMH$(15)
1115 DIMS(20):S=0
1120 FORZ=0TO15:READH$(Z):NEXTZ
1130 PRINT:INPUT"DISASSEMBLY STA
   RT ";A$
1131 IFA$="Z"THEN8000
1132 IFA$="P"THENPRINT:B=ABS(B-1
   ):IFB=1THENPRINT"printer on":GOT
   O1130 ELSEPRINT"printer off":GOT
   O1130
1135 IFA$=""THENA1=0:GOTO1140
1137 GOSUB1000:A1=A
1140 INPUT"END ADDRESS ";A$
1145 IFA$=""THENA$="FFFF"
1150 GOSUB1000:AE=A:IFAE(A1 THEN
   1140
1155 A=A1
1190 IFB=1THENPRINT#-2," "
1200 AP=A:FORZX=1TO16
1205 IFA=)AE ORA)65535THENZX=16:
   NEXTZX:GOTO1130
1210 OP$="":MN$="":PR$="":DA$=""
1215 L=0:X=0:AD=A
1220 GOSUB150:A$=AD$:GOSUB100:OP
   =D
1230 TP=(OP AND192)/64
1240 ONTP+1GOSUB2000,3000,4000,4
   000
1245 IFB=1THENPRINT#-2,A$;TAB(5)
   OP$;TAB(20)MN$;TAB(26);PR$;TAB(4
   5)DA$
1250 PRINT:PRINTA$;TAB(5)OP$;TAB
   (16)MN$;TAB(22);PR$;
1255 K$=INKEY$
1260 IFK$="X"THENZX=16:NEXTZX:GO
   TO1130
1262 IFK$="P"THENZX=16:NEXTZX:A$
   =K$:GOTO1132
1265 IFK$="Z"THENZX=16:NEXTZX:GO
   TO8000
1266 IFB=1THEN1278
1267 IFK$="R"THENZX=16:NEXTZX:GO
   TO1350
1268 IFK$="S"THENZX=16:NEXTZX:GO
   TO1300
1278 NEXTZX
1279 IFB=1THEN1200
1280 K$=INKEY$:IFK$=""THEN1280
1282 IFK$="Z"THEN8000
1283 IFK$="P"THENA$=K$:GOTO1132
1285 IFK$="X"THEN1130
1286 IFK$="S"THEN1300
1287 IFK$="R"THEN1350
1289 IFK$()CHR$(13)THEN1280
1290 PRINT:GOTO1200
1300 IFS=20THENPRINT:PRINT"stack
   full":GOTO1130
1310 S=S+1:S(S)=AP
1320 PRINT:PRINT"subroutine";S;"
   address";:INPUTA$
1325 IFA$=""THENA=0:GOTO1200
1330 GOSUB1000:GOTO1200
1350 IFS)0THENA=S(S):PRINT:PRINT
   "return";S:S=S-1:GOTO1200
1355 PRINT:PRINT"stack empty":GO
   TO1130
2000 IFOP(16 THEN3000
2005 IFOP)31 THEN2100
2010 ONOP-16 GOTO2020,2025,2030,
   2035,2035,2040,2045,2035,2050,20
   55,2035,2060,2065,2070,2075
2015 GOTO6000
2020 GOTO7000
2025 MN$="NOP":RETURN
```

```
2030 MN$="SYNC":RETURN
2035 MN$="bad opcode":RETURN
2040 MN$="LBRA":GOTO400
2045 MN$="LBSR":GOTO400
2050 MN$="DAA":RETURN
2055 MN$="ORCC":BT=1:GOTO200
2060 MN$="ANDCC":BT=1:GOTO200
2065 MN$="SEX":RETURN
2070 MN$="EXG":GOTO900
2075 MN$="TFR":GOTO900
2100 IFOP)47THEN2200
2105 ONOP-32GOTO2115,2120,2125,2
   130,2135,2140,2145,2150,2155,216
  0,2165,2170,2175,2180,2185
2110 MN$="BRA":GOTO2190
2115 MN$="BRN":GOTO2190
2120 MN$="BHI":GOTO2190
2125 MN$="BLS":GOTO2190
2130 MN$="BCC":GOTO2190
2135 MN$="BCS":GOTO2190
2140 MN$="BNE":GOTO2190
2145 MN$="BEQ":GOTO2190
2150 MN$="BVC":GOTO2190
2155 MN$="BVS":GOTO2190
2160 MN$="BPL":GOTO2190
2165 MN$="BMI":GOTO2190
2170 MN$="BGE":GOTO2190
2175 MN$="BLT":GOTO2190
2180 MN$="BGT":GOTO2190
2185 MN$="BLE"
2190 IFL=1THENMN$="L"+MN$:GOTO400
2195 GOTO350
2200 ONOP-48GOTO2215,2220,2225,2
   230,2235,2240,2245,2250,2255,226
  0,2265,2270,2275,2250,2280
2210 MN$="LEAX":GOTO500
2215 MN$="LEAY":GOTO500
2220 MN$="LEAS":GOTO500
2225 MN$="LEAU":GOTO500
2230 PP=0:MN$="PSHS":GOTO800
2235 PP=1:MN$="PULS":GOTO800
2240 PP=0:MN$="PSHU":GOTO800
2245 PP=1:MN$="PULU":GOTO800
2250 MN$="bad opcode":RETURN
2255 MN$="RTS":RETURN
2260 MN$="ABX":RETURN
2265 MN$="RTI":RETURN
2270 MN$="CWAI":BT=1:GOTO200
2275 MN$="MUL":RETURN
2280 MN$="SWI":RETURN
3000 MD=(OP AND48)/16:X=OP AND15
3005 ONX GOTO3020,3020,3030,3040
   ,3020,3050,3060,3070,3080,3090,3
  020,3100,3110,3120,3130
3010 MN$="NEG":GOTO3140
3020 MN$="bad opcode":RETURN
3030 MN$="COM":GOTO3140
3040 MN$="LSR":GOTO3140
3050 MN$="ROR":GOTO3140
3060 MN$="ASR":GOTO3140
3070 MN$="ASL":GOTO3140
3080 MN$="ROL":GOTO3140
3090 MN$="DEC":GOTO3140
3100 MN$="INC":GOTO3140
3110 MN$="TST":GOTO3140
3120 MN$="JMP":IFOP=78OROP=94THE
   N3020ELSE3140
3130 MN$="CLR"
3140 IFOP(16THEN250
3150 ONMD+1GOTO3160,3170,500,300
3160 MN$=MN$+"A":RETURN
3170 MN$=MN$+"B":RETURN
4000 X=OP AND15:BT=1
4005 ONX GOTO4020,4030,4040,4050
   ,4060,4070,4080,4090,4100,4110,4
  120,4130,4140,4150,4160
4010 MN$="SUB":GOTO4180
4020 MN$="CMP":GOTO4180
4030 MN$="SBC":GOTO4180
4040 BT=2:MN$="SUBD":IFOP)191THE
   NMN$="ADDD":GOTO4190 ELSE4190
4050 MN$="AND":GOTO4180
4060 MN$="BIT":GOTO4180
4070 MN$="LD":GOTO4180
4080 MN$="ST":IFOP=135OROP=199TH
   EN4200ELSE4180
4090 MN$="EOR":GOTO4180
4100 MN$="ADC":GOTO4180
4110 MN$="OR":GOTO4180
4120 MN$="ADD":GOTO4180
4130 BT=2:MN$="CMPX":IFOP)191THE
   NMN$="LDD":GOTO4190 ELSE4190
4140 IFOP=205THEN4200 ELSEIFOP=1
   41THENMN$="BSR":GOTO4190
4145 MN$="JSR":IFOP)191THENMN$="
   STD":GOTO4190 ELSE4190
4150 MN$="LD":BT=2:GOTO4170
4160 MN$="ST":IFOP=143OROP=207TH
   EN4200
4170 IFOP(192THENX$="X"ELSEX$="U"
4175 GOTO4185
4180 IFOP(192THENX$="A"ELSEX$="B"
4185 MN$=MN$+X$
4190 MD=(OP AND48)/16:IFOP=141 T
   HEN MD=4
4195 ONMD+1GOTO200,250,500,300,3
   50
4200 MN$="bad opcode":RETURN
6000 GOSUB100:OP=D
6010 IFOP(33THEN6190
6020 IFD(48THENL=1:GOTO2100
6030 IFD=63THENMN$="SWI2":RETURN
6040 IFD)191THEN6140
6050 IFD(131THEN6190
6060 D1=D AND15:D2=(D AND240)/16
   -7
6070 IFD1=3THENMN$="CMPD":GOTO61
   20
6080 IFD1=12THENMN$="CMPY":GOTO6
   120
6090 IFD1=14THENMN$="LDY":GOTO61
   20
6100 IFD1=15THENMN$="STY":GOTO61
   20
6110 GOTO6190
```

```
6120 BT=2:OND2 GOTO200,250,500,3
   00
6130 GOTO6190
6140 D1=D AND15:D2=(D AND240)/16
   -11
6150 IFD1=14THENMN$="LDS":GOTO61
   80
6160 IFD1=15THENMN$="STS":GOTO61
   80
6170 GOTO6190
6180 BT=2:OND2 GOTO200,250,500,3
   00
6190 MN$="bad opcode":RETURN
7000 GOSUB100
7010 IFD(63THEN6190
7020 IFD=63THENMN$="SWI3":RETURN
7030 IFD(131THEN6190
7040 D1=D AND15:D2=(D AND240)/16
   -7
7050 IFD1=3THENMN$="CMPU":GOTO70
   80
7060 IFD1=12THENMN$="CMPS":GOTO7
   080
7070 GOTO6190
7080 BT=2:OND2 GOTO200,250,500,3
   00
7090 GOTO6190
8000 PRINT:INPUT"zap display sta
   rt ";A$
8005 IFA$="P"THENPRINT:B=ABS(B-1
   ):IFB=1THENPRINT"printer on":GOT
   08000 ELSEPRINT"printer off":GOT
   08000
8006 IFA$="R"THEN1130
8015 IFA$=""THENA=AP:GOTO8030
8020 GOSUB1000
8030 IFA(0THENA=0
8035 IFB=1AND A)65535THENA=0
8040 IFA)65440THENA=65440
8042 AB=A
8045 CLS:IFB=1THENPRINT#-2," "
8050 FORZL=1TO16
8055 AD=A
8060 GOSUB150
8065 IFB=1THENPRINT#-2,AD$;"    ";
8070 PRINT:PRINTAD$;"    ";
8075 OP$="":DA$="":DB$=""
8080 FORZG=1TO3
8090 D$="":FORZB=1TO2
8100 GOSUB100
8110 PRINTD$;
8115 IFB=1THENPRINT#-2,D$;
8120 IFD)31 ANDD(91THENDB$=DB$+C
   HR$(D)ELSEDB$=DB$+"."
8130 NEXTZB:PRINT"   ";
8135 IFB=1THENPRINT#-2,"   ";
8140 NEXTZG:PRINTDB$;
8141 IFB=1THENPRINT#-2,DB$
8142 K$=INKEY$:IFK$="X"ORK$="R"O
   RK$="P"THENZL=16:NEXTZL:GOTO8180

8150 NEXTZL
8155 IFB=1THENK$=INKEY$:GOTO8180
8160 K$=INKEY$:IFK$=""THEN8160
8165 IFK$="-"ORK$="="THENA=AB-96
   :GOTO8030
8170 IFK$=";"ORK$="+"THEN8030
8175 IFK$="A"THENA=AB:GOTO8030
8180 IFK$="X"THEN8000
8190 IFK$="R"THENA=AP:GOTO1190
8192 IFK$="P"THENA$=K$:GOTO8005
8195 IFB=1THEN8030
8200 GOTO8160
9000 PRINT@135,"DISASSEMBLER 680
   9"
9010 PRINT:PRINTTAB(7)"COPYRIGHT
    (C) 1981"
9020 PRINTTAB(3)"SOFT SECTOR MAR
   KETING INC."
9030 PRINT:PRINTTAB(6)"WRITTEN B
   Y L. ASHMUN"
9050 PRINT:PRINTTAB(7)"INSTRUCTI
   ONS ? Y/N"
9060 K$=INKEY$
9070 IFK$="N"THENRETURN
9080 IFK$="Y"THEN9100
9090 GOTO9060
9100 CLS
9110 PRINTTAB(7)"COMMANDS AVAILA
   BLE"
9115 PRINTTAB(7)"(DISASSEMBLY MO
   DE)"
9120 PRINT:PRINT" X = NEW START
   ADDRESS      **"
9130 PRINT" S = GOTO SUBROUTINE
          *"
9140 PRINT" R = RETURN FROM SUBR
   OUTINE *"
9150 PRINT" Z = GOTO ZAP DISPLAY
    MODE"
9160 PRINT" P = PRINTER ON/OFF S
   WITCH"
9162 PRINT" ENTER = NEXT PAGE
         *"
9165 PRINT:PRINT" * NOT ACTIVE W
   ITH PRINTER on."
9167 PRINT"** X WILL NOT CANCEL
   PRINTER"
9170 PRINT"      OUTPUT."
9180 PRINT:PRINTTAB(10)"PRESS EN
   TER"
9185 K$=INKEY$:IFK$()CHR$(13)THE
   N9185
9190 CLS
9200 PRINTTAB(7)"COMMANDS AVAILA
   BLE"
9210 PRINTTAB(7)"(zap display mo
   de)"
9220 PRINT:PRINT" X = NEW START
   ADDRESS  **"
9230 PRINT" + = INCREMENT ONE PA
   GE  *"
9240 PRINT" - = DECREMENT ONE PA
   GE  *"
9250 PRINT" A = REDISPLAY SAME P
   AGE *"
```

```
9260 PRINT" R = RETURN TO DISASS
   EMBLY MODE"
9270 PRINT" P = PRINTER ON/OFF S
   WITCH"
9275 PRINT:PRINT" * NOT ACTIVE W
   ITH PRINTER on."
9280 PRINT"** X WILL NOT CANCEL
   PRINTER"
9285 PRINT"      OUTPUT."
9290 PRINT:PRINTTAB(10)"PRESS EN
   TER";
9295 K$=INKEY$:IFK$()CHR$(13)THE
   N9295
9299 RETURN
9300 CLS
9310 PRINT"  PRINTER BAUD RATE S
   ELECTION"
9320 B=PEEK(150)
9330 IFB=190THENB$="300 BAUD"
9340 IFB=87THENB$="600 BAUD"
9350 IFB=41THENB$="1200 BAUD"
9351 IFB=18THENB$="2400 BAUD"
9380 PRINT:PRINT"   CURRENT RATE
   - ";B$
9390 PRINT:PRINT
9400 PRINT" 1 = 300 BAUD"
9410 PRINT" 2 = 600 BAUD"
9420 PRINT" 3 = 1200 BAUD"
9422 PRINT" 4 = 2400 BAUD"
9430 PRINT:PRINT"   PRESS ENTER
   FOR NO CHANGE"
9450 PRINT:PRINT:INPUT"    WHICH
   BAUD RATE ";B$
9455 POKE155,80
9460 IFB$=""THENRETURN
9470 IFB$="1"THENPOKE150,190
9480 IFB$="2"THENPOKE150,87
9490 IFB$="3"THENPOKE150,41
9492 IFB$="4"THENPOKE150,18
9495 RETURN
```



"BUGS!"

A Simple Screen Pointer Manipulation Program

by Mark Rothstein

Are you like me? I own an Extended Basic Color Computer, and I like to know what's going on inside. Sometimes I'd like to do peculiar things that Basic won't let me do; other times I'd just like to know. Well here's a short program that will start you on your way.

The program takes a value you enter, from 0 to 31, and uses that to move the video display area that is controlled by the Synchronous Address Multiplexer (SAM). Each unit represents a 512 byte offset. A "0" entry points you at zero; a "1" points the screen to location 512, etc.

This way you can see what's going on in Basic's RAM area, watch your program variables change, or just see how BASIC stores your program. Try running the program with a zero offset and see what happens when you press "Shift 0". Location $11A (the $ represents hexadecimal) will change from a blank space to an "@" in reverse video. Also, if you press any key beside the up arrow, you will see characters changing in locations $152 - 159. This is where Basic figures out what key you've pressed.

Still interested? Try replacing line 230 with:

230 CSAVE"TEST"

or

230 RENUM

Can you see which locations are changing now? Watch out, though. RENUM and BREAK don't return to the program. They exit back to Basic with the screen pointer exactly where you left it. This could be the middle of nowhere! To get out of this, press the RESET button, or hit:

BREAK <ENTER>
RUN <ENTER>
2 <ENTER>

This resets the pointer back to where Basic expects it -- $400. You'll have to enter the keys without seeing the visual feedback on the screen, though.

Other interesting areas are cassette file names at locations $1D2 and $1DA; also the screen buffer at $2E1. Incidentally the screen buffer stores only the latest 90 keyboard entries. Or hook up the joysticks, and read the joystick value in the program loop:

add 225 J=JOYSTK(0)
and change 230 IF INKEY$="" THEN 225.

Now as you change the position of the joysticks note that locations $15A thru $15D also change.

This is only the start! Although Radio Shack doesn't provide you with listings of what goes on inside, the Color Computer is certainly powerful enough to help you figure it out.

```
10 'THIS IS A PROGRAM THAT ALLOWS
   YOU TO PEEK INTO RAM
20 'BY MOVING THE VIDEO DISPLAY
   AREA IN THE SAM
30 '
40 'BY MARK ROTHSTEIN
50 '  3123 WALNUT AVE.
60 '  OWINGS MILLS, MD 21117
70 '
80 'DIRECTIONS:
90 'WHEN THE COMPUTER ASKS FOR AN
   OFFSET, ENTER A VALUE FROM 0 TO 31
100 'THIS MOVES THE VIDEO DISPLAY
   IN UNITS OF 1/2 K BYTES
110 '
120 'AFTER YOU HIT <ENTER>, THE
   COMPUTER WILL MOVE THE DISPLAY
   POINTER.
130 'TO TRY A NEW AREA OF MEMORY,
   JUST PRESS THE UP ARROW.
140 'THE COMPUTER WILL REPLACE THE
   POINTER SO YOU CAN SEE WHAT YOU ARE
   DOING.
150 '
160 'GET THE INPUT OFFSET VALUE.
170 INPUT"OFFSET";O
180 '
190 'GO TO THE SUBROUTINE TO DO ALL
   THE HARD WORK
200 GOSUB 300
210 '
220 'THEN WAIT FOR AN UP ARROW INPUT
230 IF INKEY$<>"up arrow" THEN 230
240 '
250 'NOW RESTORE THE DISPLAY (USING
   THE SAME SUBROUTINE!)
260 O=2: GOSUB 300
270 GOTO 170
```

Continued on Page 47

CSAVE INSURANCE

by Jorge Mir
12851 W. Balboa Dr.
New Berlin, WI 53151

Have you ever CSAVEd a program after many hours of debugging, endless revisions and tiresome tests only to subsequently find out that you had a defective tape? or for whatever reason the program cannot be CLOADed again? If it happened once, that's once too many times!

Here is a simple way to assure yourself of a good copy (or several good copies) before you turn your machine off. When you are ready to CSAVE the program, type the following and hit <ENTER>:

PRINT PEEK(25) PEEK(26)

The screen will then show two numbers, such as:

30 1

This is the decimal address where your Basic program starts. Jot it down so you don't forget it. Next, type the following and hit <ENTER>:

PRINT PEEK(27) PEEK(28)

The screen will again show two numbers. This is the decimal address for the start of simple variables. Jot those numbers down so you don't forget them either. CSAVE your program as many times as you wish. Now, type the following and hit <ENTER>.

POKE 25, PEEK(27): POKE 26, PEEK(28)

This moves the Basic program beyond the end of the original program so that your original program is not erased from memory when loading a new program.

Now, CLOAD the copies you made (one at a time, of course) and check each copy out by RUNning it or LISTing it. If everything as OK, then go ahead and turn your machine off. The copies you made are OK. However, if you are not able to CLOAD the new copies, or when LISTing or RUNning them you find there are problems, then don't worry! You still have the original copy in memory!

To get back to the original copy of your program just type the following (the '#' indicates the first code you jotted down as discussed above):

POKE 25,#: POKE 26,#

This resets the start of Basic back to the beginning of the original program. You can now RUN or LIST the original program. You can also CSAVE it again! Then, you can follow the same routine over again until you know you have a good copy or copies on tape.

By the way if you want to return to the second program, You can type the following (using the second set of numbers you jotted down) and hit <ENTER>:

POKE 25,#: POKE 26,#

That's all there is to it!

continued from page 19

```
);CHR$(207);CHR$(207);CHR$(154);

170 PRINT
180 PRINT@128+C,CHR$(151);CHR$(1
55);
190 IF PEEK(344)=247THEN C=C+1
200 IF PEEK(343)=247THEN C=C-1
210 IF C>29THENC=29
220 IF C<1THENC=1
230 R9=R8:R8=R7:R7=R6:R6=R5:R5=R
4:R4=R3:R3=R2:R2=R1:R1=R
240 IF C<=R9 THEN C=C+2:GOSUB 31
0
250 IF C>=R9+6 THEN C=C-2:GOSUB
310
260 PRINT@160+C,CHR$(226);CHR$(2
25);
270 IF CR>4 THEN 110
280 PT=PT+1
290 NEXT Y
300 GOTO110
310 SOUND 150,5
320 PRINT@160+C,CHR$(130);CHR$(1
29);
330 PRINT@10,"CRASH!!!!!";
340 PRINT@ 42,"POINTS ="PT;
360 TT=TT+PT
370 PT=0
380 CRASH=CRASH+1
390 IF CRASH>4 THEN 420
400 FOR T=1 TO 1000:NEXT
410 RETURN
420 PRINT@ 100,"TOTAL POINTS ="T
T;
430 PRINT@450,"PRESS <ENTER> FOR
 A NEW GAME  ";
440 INPUT X
450 PRINT@416,""
460 TT=0
470 CR=0
480 RUN90
```

DISKS!
by Bill Sias

Last issue we had ads about two new disk controllers for the Color Computer. Since that went to press Tandy has announced their's and rumors have mentioned three others. In this article I will attempt to answer some questions people have asked me about disks, their operation and use. Most of this discussion will relate to how to use a disk and DOS and in future articles we'll get into it deeper as in how it performs it's operations and the necessary hardware.

What is a disk?

Physically a disk is similar to a soft 45 rpm record inside a perminent jacket. Although it is very flexable one of the quickest way to damage it is to bend it. It has three holes in it. The large one is the opening for the read / write head. You can see the disk through this hole and by touching the disk through it you can totally destroy the data on the disk. The next is the center hole this is used by the disk drive for positioning and to allow the disk to spin within it's jacket. The last hole is very small and is used by the disk drive to view a small hole on the disk itself so that it is constantly indexed to the correct position. On "soft sectored" disks their is one of these index holes, on "hard sectored" disks their are ten holes. Soft sectored disks are used by Radio Shack and Exatron and Hard sectored are used by Tallgrass Technology. In use there is very little functional difference between the two types of disks.

What do I gain by owning a "Disk"?

Well, the first thing you gain is rapid loading and saving of programs. Unscientifically speaking, about 1000 times faster than cassette. The second thing you gain is the ability to store data efficiently. If you have tried cassette data files at all you know the gross inefficency of that medium. With a disk you can have immediate access to somewhere between 86 and 200K of information with a cassette you can handle only what your memory allows. Another thing that you gain is tidyness. Personally I store one program per cassette, that saves me the hassle of having to SKIP over any other files before I come to the program I want (no, I don't own stock in a cassette manufacturing firm). With my disks I have one with all of my games, another with all of my utilities, etc. I can load any of them faster than you can find the right cassette.

What disadvantages does a Disk have?

Well the first and most obvious is cost. But even that is relative, when you consider the fact that a single disk holds much more than a cassette and can cost less than one of Radio Shack's certified cassettes, which in the long run will save you money (unless you are one of those folks that fill a C60 cassette completely). The other disadvantage is that, in my opinion, you will have to have 32K with any of the systems to make it practical.

What does it cost?

Again that depends on the system that you buy. Let's look at the three major systems available now. This comparision assumes that you don't want to modify your computer or void your warrentee in any way. It also assumes that you now have Extended Basic and 16K of RAM. All prices are taken from either last issue of CCN or Radio Shack's Catalog.

DISKS!

Item	Radio Shack	Exatron	Tallgrass
32K RAM	$149.95	NA	$149.95
Controller	$599.95	$298.00	$ 99.95
1 Drive	NA	$329.95	$329.95
DOS	NA	$ 29.95	$ 69.95
Total	$749.90	$657.90	$649.80

What about compatability?

Judging by past performance, Radio Shack will use their own DOS exclusively. Both Exatron and Tallgrass have introduced their own DOS. Compatability between the three of them will depend primarily on the manufacturers themselves and other outside vendors. Exatron is already working on a Radio Shack compatable version for their CCI so that will be compatable, with Tallgrass' hard sectored disks it will be difficult at best to develope a DOS compatable with Radio Shack's. Steve Odneal and I are working on making FLEX[tm] available for the Color Computer using Exatron's CCI. From past experience I feel safe in saying that someone, somewhere will probably create a method of interchange between them.

Can you compare the three systems in use?

Not really. The only experience I have with any of them is the Exatron CCI. I can say, however, that I like it very much. One of the advantages it has is that the DOS resides in RAM that is located just above the memory that the program paks usually use, which makes it easily modifiable. For example, I have been used to using CPM, TRS-DOS and NEWDOS 80, therefore I found the command CAT very hard to get used to (most systems use DIR), so I changed it to DIR. The other thing I liked was the fact that it has a command to load Model I BASIC programs into the CC, which allowed me to use a lot of software I had put away. I also have added a new command that I call CAT (I thought you didn't like that?) that reads the directorys of all of my disks and puts them into a disk file, this allows me to keep a record of all my programs and where they are located on one disk. Granted not everyone is going to modify the DOS so that "advantage" may have limited appeal except when you consider that without the RAM at locations $C000 and up developing a system that will allow you to use the FLEX[tm] operating system from TSC would have been difficult if not impossible. This alone will open up more software for the Color Computer than Radio Shack could ever develope and allows us a method of interchange with our "big" brothers like SWTPC, GIMIX and Smoke Signal Broadcasting. Another feature it has is the ability to "back-up" ROM paks. At first I was upset about this feature as I thought it was just another tool for software pirates, but I recently used it to put Radio Shack's Personal Finance program on disk for a friend and patched it so it would write the file to the disk instead of the cassette.

Which one will CCN be supporting?

All of them. Right now we have only the Exatron CCI and as such it is the only one we directly support right now, and since it will be compatable with Radio Shack's we will be better able to support it by using the Exatron system. We will, however, accept articles about any or all of them and as we purchase them will make a solid commitment to each. We have not made support of any of them magazine policy.

DISKS!

You keep using the term "DOS", what does it mean?

DOS, pronounced Doss, stands for Disk Operating System. As time goes on we'll have more articles about all of these systems and more detail about DOSs in general and specifically. But for now you probably have more information than you wanted anyway. If you have any experience with any of these please write and let others know about it.

Tell me more about Exatron's CCI.

Exatron's DOS is quite interesting, first of all it really doesn't have a "DOS level" and a "BASIC level", you are operating from both all of the time. For example, you can use any DOS command as a statement from your BASIC program. Machine language programmers will find the extra 16K, located above the BASIC ROMs very handy. I talked with one fellow who is developing a word processor using the Exatron system and he likes it because the word processor is located in the high bank of 16K which allows a 32K text buffer. Because the DOS only requires 8K of RAM that leaves another 8K that is protected from BASIC for you to add your own commands to the system, I use that area for my SORT command and have used it to draw graphic screens before displaying them in my machine language programs. In fact this month's letters column includes a program that puts a screen print routine in that memory space.

You sound like you think the Exatron CCI is the best.

First you have to keep in mind my situation. Because of the magazine I have to stay compatable with as many of you as possible without buying all of the systems that will ever become available, with the CCI I can do just that. In some cases it may require that I personally write the software to be compatable but that's alot cheaper than owning all of the controllers. Second, I write alot of long machine language programs and I need as much memory as I can get, with the CCI I have the full 48K at my disposal. In addition, being able to "instantly" load and save my source code files adds some insurance to never losing them, with cassette I save the files at the end of the session, now I save frequently.

A Simple Screen Pointer

```
280 '
290 'SUBROUTINE
300 A0=&HFFC6
310 '
320 FOR I=1 TO 7
330 A=INT(O/2)
340 B=O-2*A
350 O=A
360 POKE B+A0,0
370 A0=A0+2
380 NEXT I
390 RETURN
400 END
```


COLOR COMPUTER DISK SYSTEM

SPECIAL LIMITED OFFER

CO RESIDENT EDITOR/ASSEMBLER

TEXTPRO
TEXT EDITOR/PROCESSOR

SYSTEM MONITOR

DATAPROS
DATA COMMUNICATIONS PACKAGE

5566 Ricochet Avenue
Las Vegas, Nevada 89118

CER-COMP
(702) 452-0632

All Orders Shipped From Stock
Add $1.00 Postage – MC/Visa Add 3%

Kid's Page

```
 1 'LIGHTNIN
 2 'LIGHTNING 1.0
 3 'by Billy Sills (age 8)
 4 'July 26, 1981
 10 CLS2
 20 PRINT"POW BANG"
 30 CLS4
 40 CLS8
 50 GOTO 20
```

I saw your ad for the Kid's Page and thought I would send in a program. I have a TRS 80 Color Computer. I an 11 years old. I have enclosed a program that my brother helped me on and a picture.

```
1 REM JUNK LETTER
5 REM BY NELL RUX
10 PRINT"  THIS PROGRAM IS JUNK"
20 PRINT: PRINT: PRINT: PRINT: PRINT
:PRINT
23 PRINT"      AND SO ARE YOU.."
24 PRINT: PRINT: PRINT
25 PRINT"           SINCERELY,"
27 PRINT"         YOUR COMPUTER"
30 T=RND(255)
40 X=RND(8)
50 CLS(X)
60 SOUND T,2
70 GOTO 10
```

Dear Sirs;
I 12 years of age and very interested in computers. I have a Radio Shack TRS-80 Color Computer which gets a lot of use. I have a question. How does a joystick work???
Sincerely,
Adam Rux

Adam,
The answer to your question would make a very long article depending on the detail the author went into. Anyway, before you can understand how it works you have to know what it is. The joystick itself contains two variable resistors connected to the handle. As you move the handle you cause the resistance to change. The Color Computer has a circuit inside called an "Analog to Digital Converter". The ADC'S job is to measure this resistance and give the Color Computer a number that is proportional to the amount of resistance it found in each of the variable resistors. The entire process is more complex than this the basic princepal is true. I hope this makes it a bit (bad pun) clearer. I have a question. Do you REALLY eat goldfish? YUCCHHH!!!
Bill

```
 1 REM GOLD FISH
 5 REM BY ADAM RUX
10 PRINT"THIS IS THE GOLDFISH GAME"
20 PRINT" "
30 PRINT"WOULD YOU LIKE
INSTRUCTIONS?"
40 INPUT D$
50 IF LEFT$(D$,1)="N" THEN 70
60 PRINT"THE OBJECT OF THE GAME IS
TO EAT AS MANY GOLDFISH YOU CAN
WITHOUT BERPING."
70 A=RND(100)
75 PRINT
80 PRINT"HERE'S THE GOLDFISH..."
81 FOR K=1 TO 2000
82 NEXT K
85 PRINT "**** ********* ********
******"
90 PRINT"<<<<< <<<<<CHOW
BROTHER>>>>> >>>>>"
91 PRINT "**** ********** *******
******"
92 FOR T=1 TO 500
95 NEXT T
96 FOR I=1 TO 15
100 PRINT" I "
110 NEXT I
120 IF A<50 GOTO 140
130 IF A>50 GOTO 150
140 PRINT" I ******* SLOOSSHHH
********** THAT ONE SLID DOWN YOUR
THROAT LIKE MILK !!!!!!!!!!!!"
145 GOTO 70
150 PRINT" I ******* BERRRP
************ YOU NOW FACE THE
CONSEQUENCES OF INDIGESTION !!!!!!!!!
THE ONLY REMEDY IS PEPTO-BISMOL."
160 PRINT" DO YOU WANT TO EAT SOME
MORE"
170 INPUT B$
180 IF LEFT$(B$,1)="Y" THEN 70
185 PRINT"    GOOD BY"
190 END
```

TAPETYPE

Have you ever wondered just what is on that unlabeled tape in the bottom of your tape basket? Or where a machine language cassette loads? Or why your favorite program tape gets an I/O error?

This program tells you just what is on a tape, record by record. Each record that is encountered is dumped verbatim to the screen, while the program decodes all the information as to record type, file type, load addresses, checksum errors and so forth.

The listing of the program which is given below is interesting as an example of Position Independent Code (P.I.C.), in-line parameters and stack based variables.

This program may be typed into your Editor Assembler (it was written on the SDS80C from the Micro Works). If you don't feel like typing, it is available in object on cassette from the Micro Works for $14.95.

```
0001 0600                          NAM TAPETYPE
                                   *
                                   *  ANDREW E. PHELPS
                                   *  THE MICRO WORKS
                                   *  21 SEPTEMBER 1981

                                   *  ROM ENTRY POINTS -
0002 A701                          SREAD  EQU $A701        SYNC AND READ
0003 A70B                          READ   EQU $A70B        PLAIN READ
0004 A77C                          SYNC   EQU $A77C        READ $55'S

                                   *  BLOCK READ PARAMETERS -
0005 007C                          BTYPE  EQU $7C          BLOCK TYPE
0006 007D                          BLEN   EQU $7D          BLOCK LENGTH
0007 007E                          BADDR  EQU $7E          BLOCK ADDRESS

0008 0400                          SCREEN EQU $400
0009 0500                          BUFFER EQU SCREEN+$100
                                   *  VARIABLES (ON STACK) -
0010 0000                          FILLEN EQU 0            FILE LENGTH
0011 0002                          SFLAG  EQU 2            SYNC FLAG
0012 0003                          EOFLAG EQU 3            FOUND END-OF-FILE
0013 0004                          STAD   EQU 4            START ADDRESS
0014 0006                          VLEN   EQU 6            TOTAL STACK BYTES


0015 0600 327A                     START  LEAS -VLEN,S     VAR SPACE
0016 0602 1F43                            TFR S,U          VARIABLE POINTER
0017 0604 17017D                          LBSR INITSC      SET UP SCREEN
0018 0607 CC0000                          LDD #0
0019 060A EDC4                            STD FILLEN,U     CLEAR LEN
0020 060C ED44                            STD STAD,U       CLEAR S.A.
0021 060E 6F42                            CLR SFLAG,U      SYNC FIRST
0022 0610 6F43                            CLR EOFLAG,U     NOT END FILE

                                   *  MAIN LOOP -
                                   *  READ A RECORD AND CALL THE
                                   *  APPROPRIATE PROCESSOR.
0023 0612 8E0500                   RECORD LDX #BUFFER
0024 0615 9F7E                            STX BADDR        BLOCK TO SCREEN
0025 0617 B60400                          LDA SCREEN
0026 061A 8A40                            ORA #$40
0027 061C B70400                          STA SCREEN       FLASH CORNER
0028 061F 6D42                            TST SFLAG,U
0029 0621 2705                            BEQ A@
```

```
0030 0623 BDA70B            JSR READ          NON-SYNC READ
0031 0626 2003              BRA B@
0032 0628 BDA701      A@    JSR SREAD         SYNC READ
0033 062B             B@
0034 062B 2712              BEQ C@            CHECKSUM ERROR?
0035 062D 108E042F          LDY #SCREEN+ES
0036 0631 170190            LBSR MSG
0037 0634 2A4552524F        FCC /*ERROR*/
0038 063B 8680              LDA #$80          FILL BLACK
0039 063D 2010              BRA R@
0040 063F 108E042F    C@    LDY #SCREEN+ES
0041 0643 17017E            LBSR MSG
0042 0646 4F4B202020        FCC /OK     /
0043 064D 8660              LDA #$60          FILL GREEN
0044 064F 170149      R@    LBSR FILL         FILL SCREEN

0045 0652 6D43              TST EOFLAG,U      NEW FILE?
0046 0654 270A              BEQ S@            SKIP IF NOT
0047 0656 108E046C          LDY #SCREEN+FN
0048 065A 170182            LBSR CLR4         BLANK OUT NAME
0049 065D 17017F            LBSR CLR4
0050 0660 6F43        S@    CLR EOFLAG,U

0051 0662 108E044E          LDY #SCREEN+RT
0052 0666 967C              LDA BTYPE         BLOCK TYPE
0053 0668 10270017          LBEQ HEADER       0=NEW FILE
0054 066C 4C                INC A             WAS IT -1?
0055 066D 102700DC          LBEQ EOFILE       END OF FILE
0056 0671 8102              CMPA #2           WAS IT +1?
0057 0673 102700BF          LBEQ DATABL       DATA BLOCK
0058 0677 17014A            LBSR MSG
0059 067A 494C4C4547        FCC /ILLEGAL/
0060 0681 208F              BRA RECORD

                      *  HEADER BLOCK -
                      *  DISPLAY FILE NAME & INFO
                      *  AND SELECT SYNC / NO SYNC
0061 0683 17013E      HEADER LBSR MSG
0062 0686 4845414445        FCC /HEADER /
0063 068D 108E04DC          LDY #SCREEN+EA
0064 0691 17014B            LBSR CLR4         CLEAR END ADDR
0065 0694 108E046C          LDY #SCREEN+FN
0066 0698 8E0500            LDX #BUFFER
0067 069B 170135            LBSR COPY         DISPLAY NAME
0068 069E 108E048C          LDY #SCREEN+FT
0069 06A2 B60508            LDA BUFFER+8      FILE TYPE
0070 06A5 2712              BEQ BASFIL        BASIC?
0071 06A7 4A                DEC A
0072 06A8 271B              BEQ DATFIL        DATA FILE?
0073 06AA 4A                DEC A
0074 06AB 2724              BEQ MACFIL        MACHINE LANG.?
0075 06AD 170114            LBSR MSG
0076 06B0 494C4C4547        FCC /ILLEGAL/
0077 06B7 203A              BRA CLRIT

0078 06B9 170108      BASFIL LBSR MSG
0079 06BC 4241534943        FCC /BASIC  /
0080 06C3 202E              BRA CLRIT
0081 06C5 1700FC      DATFIL LBSR MSG
0082 06C8 4441544120        FCC /DATA   /
```

```
0083 06CF 2022              BRA CLRIT
0084 06D1 1700F0      MACFIL LBSR MSG
0085 06D4 4D41434849        FCC /MACHINE/
0086 06DB 108E04B3          LDY #SCREEN+SA
0087 06DF FC050B            LDD BUFFER+11
0088 06E2 1700C4            LBSR HEXOUT
0089 06E5 108E04CF          LDY #SCREEN+LA
0090 06E9 FC050D            LDD BUFFER+13
0091 06EC ED44              STD STAD,U      SAVE LOAD AD
0092 06EE 1700B8            LBSR HEXOUT
0093 06F1 2013              BRA NEXT

0094 06F3 108E04B3    CLRIT LDY #SCREEN+SA
0095 06F7 1700E5            LBSR CLR4
0096 06FA 108E04CF          LDY #SCREEN+LA
0097 06FE 1700DE            LBSR CLR4
0098 0701 CC0000            LDD #0
0099 0704 ED44              STD STAD,U      NO LOAD AD

0100 0706 CC0000      NEXT  LDD #0
0101 0709 EDC4              STD FILLEN,U    CLEAR LENGTH
0102 070B 8D6F              BSR WLEN        DISPLAY LENGTH

0103 070D 108E0494          LDY #SCREEN+FT+8
0104 0711 B6050A            LDA BUFFER+10   GAP TYPE
0105 0714 270C              BEQ L@          SKIP IF NO GAPS
0106 0716 1700AB            LBSR MSG
0107 0719 4153434949        FCC /ASCII  /
0108 0720 2011              BRA M@
0109 0722 17009F      L@    LBSR MSG
0110 0725 42494E4152        FCC /BINARY /
0111 072C 8601              LDA #1
0112 072E A742              STA SFLAG,U     FLAG NO SYNCS
0113 0730 BDA77C            JSR SYNC        DO FIRST SYNC
0114 0733 16FEDC      M@    LBRA RECORD

                    *  DATA BLOCK -
                    *  DISPLAY AND COUNT FILE SIZE
0115 0736 17008B    DATABL LBSR MSG
0116 0739 4441544120        FCC /DATA   /
0117 0740 ECC4              LDD FILLEN,U
0118 0742 DB7D              ADDB BLEN       ADD ON FILE LEN
0119 0744 8900              ADCA #0
0120 0746 EDC4              STD FILLEN,U
0121 0748 8D32              BSR WLEN        DISPLA FILE LEN
0122 074A 16FEC5            LBRA RECORD

                    *  END OF FILE BLOCK -
                    *  DISPLAY FINAL SIZE
                    *  (NAME AND LENGTH ARE ZEROED
                    *  IN CASE NEXT BLOCK IS NOT
                    *  A HEADER.)
0123 074D 8D75      EOFILE BSR MSG
0124 074F 454E444649        FCC /ENDFILE/
0125 0756 6F42              CLR SFLAG,U     BACK TO SYNCS
0126 0758 8601              LDA #1
0127 075A A743              STA EOFLAG,U    CLEAR NAME
0128 075C 8D1E              BSR WLEN        DISPLAY LENGTH
0129 075E 108E04FC          LDY #SCREEN+FL+14
0130 0762 ECC4              LDD FILLEN,U
```

```
0131 0764 8D43              BSR HEXOUT      DISPLAY AGAIN
0132 0766 ECC4              LDD FILLEN,U
0133 0768 E344              ADDD STAD,U     START ADDR
0134 076A 830001            SUBD #1         "THRU"- NOT "TO"
0135 076D 108E04DC          LDY #SCREEN+EA
0136 0771 170035            LBSR HEXOUT     DISPLAY END
0137 0774 CC0000            LDD #0
0138 0777 EDC4              STD FILLEN,U    ZERO LENGTH
0139 0779 16FE96            LBRA RECORD

                        *  DISPLAY CURRENT LENGTH -
0140 077C 108E04EE      WLEN    LDY #SCREEN+FL
0141 0780 ECC4                  LDD FILLEN,U
0142 0782 2025                  BRA HEXOUT

                        *  SET UP INITIAL SCREEN -
0143 0784 8E0400        INITSC LDX #SCREEN
0144 0787 318D005F             LEAY BANNER,PCR
0145 078B A6A4          F@      LDA 0,Y         END OF PATTERN?
0146 078D 2702                  BEQ G@
0147 078F E6A0                  LDB ,Y+
0148 0791 CA40          G@      ORB #$40        NON-INVERT
0149 0793 E780                  STB ,X+
0150 0795 8C0600                CMPX #SCREEN+$200
0151 0798 25F1                  BLO F@
0152 079A 39                    RTS
                        *  FILL UNUSED BUFFER SPACE -
                        *  CALLED WITH COLOR IN "A".
0153 079B 8E0500        FILL    LDX #BUFFER
0154 079E D67D                  LDB BLEN        BLOCK LENGTH
0155 07A0 3A                    ABX
0156 07A1 A780          D@      STA ,X+
0157 07A3 8C0600                CMPX #BUFFER+$100
0158 07A6 26F9                  BNE D@
0159 07A8 39                    RTS

                        *  DISPLAY NUMBER IN HEX -
                        *  NUMBER IN "D", SCREEN ADDRESS
                        *  IN "Y"
0160 07A9 8D02          HEXOUT BSR HEXBYT
0161 07AB 1F98                  TFR B,A
0162 07AD 3402          HEXBYT PSHS A
0163 07AF 44                    LSR A
0164 07B0 44                    LSR A
0165 07B1 44                    LSR A
0166 07B2 44                    LSR A
0167 07B3 8D02                  BSR HEXNYB
0168 07B5 3502                  PULS A
0169 07B7 840F          HEXNYB ANDA #$0F
0170 07B9 8109                  CMPA #9
0171 07BB 2302                  BLS Q@
0172 07BD 8BC7                  ADDA #7-$40
0173 07BF 8B70          Q@      ADDA #$70
0174 07C1 A7A0                  STA ,Y+
0175 07C3 39                    RTS

                        *  DISPLAY 7-CHARACTER MESSAGE
                        *  (IN-LINE PARAMETER)
                        *  SCREEN ADDRESS IN "Y".
```

```
0176 07C4 3510          MSG     PULS X
0177 07C6 C607                  LDB #7
0178 07C8 A680          E@      LDA ,X+
0179 07CA 8A40                  ORA #$40
0180 07CC A7A0                  STA ,Y+
0181 07CE 5A                    DEC B
0182 07CF 26F7                  BNE E@
0183 07D1 6E84                  JMP 0,X

                        *  COPY 8 BYTES FROM X TO Y -
0184 07D3 C608          COPY    LDB #8
0185 07D5 A680          O@      LDA ,X+
0186 07D7 8A40                  ORA #$40
0187 07D9 A7A0                  STA ,Y+
0188 07DB 5A                    DEC B
0189 07DC 26F7                  BNE O@
0190 07DE 39                    RTS

                        *  CLEAR FOUR SPACES ON SCREEN
0191 07DF 8660          CLR4    LDA #$60
0192 07E1 A7A0                  STA ,Y+
0193 07E3 A7A0                  STA ,Y+
0194 07E5 A7A0                  STA ,Y+
0195 07E7 A7A0                  STA ,Y+
0196 07E9 39                    RTS
0197 07EA 2A54415045    BANNER  FCC /*TAPETYPE       /
0198 07FA 2054484520            FCC / THE MICRO WORKS/
0199 080A 4552524F52            FCC /ERROR STATUS =  /
0200 002F               ES      EQU $2F
0201 081A 2020202020            FCC /                /
0202 082A 5245434F52            FCC /RECORD TYPE =   /
0203 004E               RT      EQU $4E
0204 083A 2020202020            FCC /                /
0205 084A 46494C4520            FCC /FILE NAME =     /
0206 006C               FN      EQU $6C
0207 085A 2020202020            FCC /                /
0208 086A 46494C4520            FCC /FILE TYPE =     /
0209 008C               FT      EQU $8C
0210 087A 2020202020            FCC /                /
0211 088A 5452414E53            FCC /TRANSFER ADDRESS/
0212 089A 203D202020            FCC / =              /
0213 00B3               SA      EQU $B3
0214 08AA 4C4F414420            FCC /LOAD ADDRESS =  /
0215 00CF               LA      EQU $CF
0216 08BA 2020202020            FCC /       THRU     /
0217 00DC               EA      EQU $DC
0218 08CA 46494C4520            FCC /FILE LENGTH =   /
0219 00EE               FL      EQU $EE
0220 08DA 00                    FCB 0

0221 08DB                       END START
```

```
BADDR   007E  BANNER  07EA  BASFIL  06B9  BLEN    007D
BTYPE   007C  BUFFER  0500  CLR4    07DF  CLRIT   06F3
COPY    07D3  DATABL  0736  DATFIL  06C5  EA      00DC
EOFILE  074D  EOFLAG  0003  ES      002F  FILL    079B
FILLEN  0000  FL      00EE  FN      006C  FT      008C
HEADER  0683  HEXBYT  07AD  HEXNYB  07B7  HEXOUT  07A9
INITSC  0784  LA      00CF  MACFIL  06D1  MSG     07C4
NEXT    0706  READ    A70B  RECORD  0612  RT      004E
SA      00B3  SCREEN  0400  SFLAG   0002  SREAD   A701
STAD    0004  START   0600  SYNC    A77C  VLEN    0006
WLEN    077C
```

Color Computer News Magna-zine Service.

This New Device Will Give You A Three Weeks Vacation!!!

Well actually, the "vacation" is from the tedium of hand typing the programs published in **Color Computer News**. Even if you are a fairly good typist (i.e. you use *more* than two fingers, and you *don't* have to look at the keyboard!) it would take you about *twelve hours* to type in most of the programs in an average **Color Computer News** issue — and *then* you have to de-bug the programs on top of that! Save your "finger energy" for scratching your head while you think great thoughts and leave the program typing to the **CCN Magna-zine Service**. We guarantee that our monthly program tapes will save even the fastest typist many hours of frustration!! Relief for your tired fingers is just a **CLOAD** away!

Each month, CCN Magna-zine subscribers receive a top quality digital cassette which contains about a half dozen programs from their favorite CC-80 magazine, **Color Computer News**. American and Canadian subscriptions are available for just $42.00 (plus $6.00 first class postage) for a full 12 issues and can start with any issue number you specify. Single issues are also available for the low price of just $6.00 each plus $1.00 postage. Subscription postage to all other countries is $15.00 per year (sent via AO Air Mail). Overseas single issue postage is $2.00 per tape. (Florida residents add $.30 sales tax for single tape purchases *only*.)

The **CCN Magna-zine Service** is staffed by people who are highly qualified in cassette tape mastering and production and who use only top quality, custom loaded, all American made digital cassettes. Each tape is *fully* guaranteed for one full year against *any and* ***all*** hazards — up to and including the tape being crushed by a falling meteor!! Just return the original tape (or at least the piece with our label on it!) along with $1.00 for return postage, and that issue will be instantly replaced — no questions asked! Who else offers you such a guarantee???

To start your own subscription to the **CCN Magna-zine**, just fill out the coupon (a photo copy or a plain piece of paper with the proper information is just fine!) and mail it to: **CCN Magna-zine Service**, Box 68, Safety Harbor, Florida 33572. Include your check (personal checks are OK) or money order and be sure to indicate which **Color Computer News** issue you want your subscription to begin with if it is anything other than the next as yet unpublished issue number.

You already *know* about the high quality programming articles that have set **Color Computer News** apart from all other computer magazines, therefore, you *also* know what to expect from our cassette tape version!!! So, don't delay any longer — send in for your own subscription today! Spend your time *computing*, ***NOT*** *typing!!!*

YES! *Sign me up for a one year's subscription to the CCN Magna-zine! Enclosed is my check/money order for the full amount (including postage) of $48.00 (domestic and Canada) or $57.00 (overseas).*

NAME _____

STREET ADDRESS _____ *APT. #* _____

CITY _____ *STATE* _____ *ZIP* _____

Begin with issue number _____ ***instead of the next regular issue.***

CCN Magna-zine Service ~ Box 68 ~ Safety Harbor, FL 33572

I'd like to dedicate this book Color Computer News The Best of 1981 to all of the subscribers that believed in our project in the first half of 1981 and also to the advertisers that "came aboard" in our first four issues. To thank these advertisers we have placed a small copy of their current ad in this book.

While going through the old issues I was reminded of my philosophy for this medium. My cause was to find a method for Color Computer owners to exchange ideas and techinques with each other and to help one another with the problems that occur in the pursuit of their hobby. The fact that the medium of exchange is paper is inconsequential. I think we've (you and I) have done our job well.

The only sign of life is growth and I hope you'll see in these pages that CCN is very much "Alive and Well".