ENGINEERING NOTES
on
Radio Shack Color Computers

May 1984
Vol. 1 No. 4

$1.95

DYNACOM

COLOR

NEWS

* ASCII PART II
* INTERRUPTS

* COMPUTER THEORY
* BASIC PROGRAMMING
* OPERATING HINTS
* MACHINE LANGUAGE PROGRAMMING
* QUESTIONS & ANSWERS

DYNAMIC COLOR NEWS is published monthly by DYNAMIC ELECTRONICS, INC., P.O. Box 896, Hartselle, AL 35640, phone (205) 773-2758. Bill Chapple, President; Alene Chapple, Sec. & Treas.; John Pearson, Ph. D. Consultant; Bob Morgan, Ph. D., Consultant.

 Subscriptions are $15/yr for U.S.A. & Canada, $30 other foreign.

The purpose of this newsletter is to provide instruction on Basic & Machine Language programming, Computer theory, operating techniques, computer expansion, plus provide answers to questions from our subscribers.

The submission of questions, operating hints, and solutions to problems to be published in this newsletter are encouraged. All submissions become the property of Dynamic Electronics the material is used. We reserve the right to edit all material used and not to use material which we determine is unsuited for publication.

All paid subscribers are entitled to discounts of 10% on hardware. and 20% on software manufactured or produced by Dynamic Electronics Inc. plus "specials" mentioned in the newsletter. To receive these discounts use your DCN number which is at the right of your name on the address label. DCN subscribers may obtain a personalized reply to specific computer problems or advice on purchasing equipment. The charge for this service is $10.

DYNAMIC COLOR NEWS

May 1984

Editor and Publisher
Bill Chapple

Secretary
Belinda Parker

CONTENTS

EDITOR'S COMMENTS

A recent problem with my house prompted me to look at another application for microcomputers. A couple of years ago I had my house insulated using the Tennessee Valley Authority's (TVA) home insulation program. My electricity is supplied by TVA and this program is supposed to reduce power consumption saving TVA the expense of building additional power generating plants. The program works like this. A TVA representative inspects the house and makes recommendations as to what needs to be done. A TVA certified insulation firm then is selected by the home owner to do the work at an agreed price. As the work is progressing and after it is finished TVA inspects the job to make sure it complies with their specifications. The cost is added to the monthly electric bill over a period of a few years.

The job was completed and the benefits of the added insulation were immediately realized. The additional monthly cost for the TVA loan is obviously offset by the reduction in my electric bill. The house is totally electric. About six months ago I begin to notice a problem. The shingles began curling upwards and breaking off. After further investigation I noticed that the insulation which was blown into the attic was so thick that the air flow was greatly reduced. I will need a new roof soon, but with a 12 year old house this is not too painful a thought. You may ask where do computers fit into this? The answer is for monitoring the heat in the attic. As mentioned in an earlier newsletter it would really be nice if the computer could be made to control and monitor devices around a home or business.

How convenient it would be if the computer could monitor my own telephone calls. It could give me the date, time of day, and length of each call made. Also it could control an attic fan for my air flow problem, be a smart telephone answerer, a fancy clock, a family message center, and a burglar alarm. These are just a few of the obvious applications. To do these tasks requires a hardware interface. The computer can handle these tasks on an interrupt basis. The computer can be interrupted several times a second and attend to these secondary tasks and return to its previous task with no loss of data. The writing of the programs to do these tasks is fairly easy. As stated before I think that these applications are very promising for computers. The question is "are consumers willing to pay for these features". There must be a reasonable assurance that consumers want these devices before expensive development is undertaken.

Because of the numerous questions asked us about our 96KX and 128K memory expanders we are including editorials on both in this issue. We are in the process of putting together a new catalog of our products. We want it to be in a form similar to our newsletter and will show sketches to give you an idea of how the product looks. When it is completed a copy will be mailed to each subscriber.

We had to redesign the modules for our 128K expanders. The SAM chip controls the refreshing of the memory chips. Some initial circuitry placed under this chip killed the memory refresh if it was not properly working rendering the computer useless. After we moved this circuitry to the module under the PIA chip the problem disappeared since even if the PIA chip failled the computer will still operate. Our 128K modules work as well as could be expected. Read the editorial if you are interested in this.

We are continuing with the discussion of ASCII. There seems to be

quite an interest in bulletin boards and we are looking into starting one. ASCII is used with word processors, printers, modems, and terminals. If you are interested in these subjects then you should read our ASCII section.

For our machine language section we are covering interrupts. These can be used with basic as well machine language programs and will allow you to stop what you are doing and run another program. For example you can print to a printer the information that is on the screen or exchange the information on the screen with other information. These are just examples. The list can go on and on. There is a 60 hertz interrupt that you can patch into and modify the basic or keyboard routines. These can also look for levels on a port to do the monitoring functions discussed earlier. Our interrupt switch is an extra feature you can add. It forces the computer to run a machine language program when it is pushed. The program can be anything you would like from a hard reset to a complex program. It can be located anywhere in memory. The command for jump extended is put in location 265 and the vectors for the program are in locations 266 and 267.

ASCII PART II

ASCII & BASIC

Last month we introduced ASCII and showed how it is constructed for serial data transmission. For storing information or characters in memory we have to convert the characters to numbers since computers only work with numbers. Therefore ASCII is a standard by which each character of the alphabet plus some printer operations are assigned numerical values. For earlier computers it was necessary to enter the ASCII values of characters you wanted printed. Basic has operations that convert characters to their ASCII value and ASCII values to characters. We want to cover these now.

INKEY$

This command forces the computer to recognize characters from the keyboard. Since the characters are strings as indicted by the "$" sign, they can be printed on the screen. A basic program to print characters on the screen follows:

```
10 A$=INKEY$
15 IF A$="" THEN 10
20 ?A$;
25 GO TO 10
```

Statement 10 defines A$ to be the key pressed. Statement 15 says that if no key is pressed then go back to 10. Statement 20 causes the character to be printed.

ASC (STRING$)

Now to convert the character to a value we use the ASC (A$) basic command. A = ASC (A$) will give the ASCII value of the character. Now we can store A in memory using POKE M, A. Notice that we have gone from a character to its ASCII value. We need another command to convert an ASCII value to a string.

CHR$ COMMAND

If A is the ASCII value of a character stored in memory then we can recover the character and print it on the screen with the following:

```
 5 INPUT "MEMORY LOCATION"; M
10 A = PEEK (M)
15 A$ = CHR$ (A)
20 ? A$;
25 M = M + 1: GO TO 10
```

The above program will display the characters stored in memory.

The ASCII Value of CHARACTERS

The following program will display the ASCII value of any keyboard character. We suggest that you type in the program and run it if you are not familiar with the ASCII values of the characters.

```
10 ' THIS DISPLAYS THE ASCII VALUES OF KEYBOARD CHARACTERS
15 A$=INKEY$ : IF A$="" THEN 15
20 A = ASC (A$) : ?A ; A$
30 GO TO 15
```

Try the program out for upper case and lower case letters, numerics, and punctuation symbols so you can get a good feel for ASCII values.

ASCII and PRINTERS

If you use a printer then you will need ASCII. Some of the ASCII codes are used for printer control. Examples are

ASCII value	Printer function
10	Line feed (rolls paper down for the next line)
12	page feed (go to the top of the next page)
13	carriage return (moves print head back to column 1)
27	ESCAPE (used to set printer options)

The instruction manual that came with your printer will tell you what printer codes are required for the various options. These codes can be entered from basic by using the following:

```
?#-2, CHR$ (X)  <ENTER>
```

X is the ASCII value in decimal.

SAVING PROGRAMS IN ASCII

You have probably read in the instructions that came with your computer that basic programs can be saved in ASCII. Why would you want to do this? What is the difference in the structure of an ASCII basic program and a normal basic program? If you will recall from our discussion of vectors, a basic statement is preceeded by a 0 plus vectors in the next 4 bytes. The next byte contains the value representing the basic command. On the other hand, ASCII means that

the program is saved in text form. It takes more space for an ASCII basic program than for a normal basic program. The reason is that every command has to be written out for ASCII. In basic each command occupies 1 or 2 bytes while in ASCII each character occupies 1 byte. For example GO SUB requires 6 bytes and RETURN requires 5 bytes for ASCII. For basic the same commands require 2 and 1 bytes respectively. To save a program in ASCII just put a ,A after the closing quotations.

EXAMPLE: (C)SAVE "TEST", A

EXCHANGING PROGRAMS BETWEEN COMPUTERS

When programs are saved in ASCII format then they can be loaded into any other computer that uses Microsoft Basic. These can even be computers that use different types of microprocessors. Since the ASCII file is just a text file then it is independent of machine language codes. As the ASCII program is loaded into the computer, the basic tokens are inserted and the beginning vectors are computed and added. Each statement in the ASCII program will end with a CHR$ (13) which is the code for "RETURN" or "ENTER". The computer looks for this code so it will know when an end of statement has occurred. After each statement, basic needs a 0 plus the next statement vector followed by the statement number vector. Rather than get into the mechanics of how this is done let's just say that the computer converts the ASCII characters into basic vectors and command tokens.

WORD PROCESSORS - BASIC PROGRAMS

Since an ASCII basic program is a text file why can't word processors be used to generate and edit basic programs? The answer is "they can and will do a nice job". Maybe we will devote an issue to using word processors for creating and editing basic programs if there is enough interest in this area. We used our word processor extensively for creating and editing basic programs.

MACHINE LANGUAGE PROGRAMMING

INTERRUPTS

The use of interrupts does not seem to be well known. An interrupt causes the computer to stop what it was doing, save all pertinent information about its task, take on a new task, and return to the first task without loosing any information. We now have an interrupt switch which you can add to your computer for $9.95. We use the nonmasked interrupt (NMI) because it is the highest priority interrupt and cannot be masked off (disabled) by software.

Let's look at the state of the microprocessor when the interrupt occurs. Let's assume that some kind of program is in progress. The microprocessor doesn't recognize basic, the only language it knows is machine language. Basic is a collection of machine language subroutines that are designed to make it easy for the user. When the interrupt appears the microprocessor finishes the instruction it was doing. Next all of the registers are stored on the stack by using the "PUSH" instruction. Remember a stack is just a designated memory area. Information is stored on the stack with "PUSH" and removed from the stack with "PULL". An interrupt automatically stores the entire machine state. The return from interrupt (RTI) instruction "PULLS"

the previous machine state from the stack allowing the microprocessor to resume its first task.

ENABLING the INTERRUPT

Memory locations 265, 266, and 267 are reserved for the nonmasked interrupt by basic. The top few bytes in the memory map are reserved for the microprocessor's reset and interrupt vectors. A chart showing the interrupts and their vectors follows:

Interrupt	Description	Location	Vector
RESET		65535 (FFFE)	40999
Non-Maskable Interrupt (NMI)		65533 (FFFC)	265
Software Interrupt (SWI)		65531 (FFFA)	262
Interrupt Request (IRQ)		65529 (FFFF)	268
Fast Interrupt Request (FIRQ)		65527 (FFF6)	271
Software Interrupt 2 (SW12)		65525 (FFF4)	259
Software Interrupt 3 (SW13)		65523 (FFF2)	256

Notice that the reset vector points to 40999 which is the value you can use to reset from the keyboard by EXEC 40999. How do we use the nonmasked interrupt? As soon is the interrupt is activated by pressing a switch the computer begins to run a machine language program. It goes to location 265 for its instruction. Since there are only 3 bytes available we can only jump to some other location to continue with the interrupt program. Let's say that we want the interrupt to provide a hard reset. This involves storing a 0 in location 113, and then going to memory location 40999 for the reset. Remember a hard reset requires a 0 in memory location 113. Let's write a program using basic.

```
10 POKE 113, 0
20 EXEC 40999
```

Now for the machine language program we need to pick a memory location in which to write the program. Let's use location 480 for the start of the interrupt program. So in location 265 we need a JUMP EXT to 480 instruction. At 480 we will write the equivalent of the above program.

```
265  JMP E   to 480

480  CLR D  113     (Put a 0 into location 113)
482  JMP E 40999    (Go to the reset subroutine)
485  RTI            (Return from interrupt)
```

Notice we used a "D" after CLR. This is the "DIRECT" addressing mode. It only takes 1 byte and its location is referenced to the direct page counter. Most of the time the direct page couter is a 0 so for memory locations less than 256 we can use the direct mode. This eliminates a zero being one of our values. This will be important later when we show you how to carry machine programs with basic programs in remark statements. The values of the assembled program are as follows with the characters in parenthesis being the HEX equivalent of the decimal values

```
265 (109)    126  (7E)   Code for Jump extended
```

266 (10A)	1	(01)	MSB of Jump Location
267 (10B)	224	(E0)	LSB of Jump Location
480 (1E0)	127	(7F)	Code for Clear Direct
481 (1E1)	113	(71)	Location 113
482 (1E2)	126	(7E)	Code for Jump extended
483 (1E3)	160	(A0)	MSB of 40999
484 (1E4)	39	(27)	LSB of 40999
485 (1E5)	59	(3B)	Code for Return from Interrupt

MSB and LSB are the most and least significant bytes of the location vector (see our February newsletter for an explanation of vectors).

Notice that the value in location 265 is 126. So to set up an interrupt vector poke a 126 in 265 and the MSB and LSB values of the machine language starting location should be poked (stored) in locations 266 and 267. If you have an assembler you might want to try this problem and see if you get the same values as we did. The return from interrupt routine is not needed since the computer goes through its normal power up routine when the interrupt is enabled. Also our values in 265-267 are erased by the power up routine and it is necessary to re-enter them after the reset. Our 96KX software has this hard reset feature built in.

In case you don't know what we mean by a hard reset do the following:

```
POKE 25, 200: NEW
```

You will receive an error message. Now enter ?MEM and you will get an out of memory error. Now push the reset button on the rear of the computer and again ? MEM. The computer is locked up. However if you have an interrupt switch you can force the computer to reset with the above program.

The nonmasked interrupt can be used to run any machine language program. Just put the JUMP E to M values in locations 265 to 267. As an exercise see if you can use the nonmasked interrupt to do the display switching we presented last month.

For those of you that want to install an interrupt switch we are including this in a special hardware section. You can make your own or purchase a switch with eyelets from us with a cassette of programs you can run by pressing the interrupt switch for $9.95.

INSTALLING AN INTERRUPT SWITCH

As much as possible we want to show you how to add hardware features to your computer. However we want to warn you that you can damage your computer with one mistake. We have blown microprocessor chips, PIA chips, SAM chips and memory chips with simple mistakes. The installing of the interrupt switch is fairly easy and considering the fact that it adds a means of starting any machine language program it is a worth while addition.

You will need a normally open push button switch, some small wire, an optional 40 pin socket, and a hole in the computer case to mount the switch. The switch will need to be wired across the nonmasked interrupt input and ground. For the microprocessor these are pins 1 and 2. If you have a 40 pin socket then you can wire pins 1 and 2 to the switch and plug the microprocessor into the socket and the socket with the microprocessor into the microprocessor's socket. If you

don't have a 40 pin socket then you can solder directly to the microprocessor's pins although we don't recommend this. We use eyelets that fit over pins 1 and 2 on the microprocessor. However you can't buy eyelets at parts stores so you will have to get an extra socket or solder to the microprocessor or the output connector.

ASSEMBLY STEPS

1. Collect the required parts - A push button swith, about 18" of 2 conductor wire, and an optional 40 pin socket.

2. Locate a place to mount the switch. We mount ours under the front lip on the right hand side.

3. Measure the wire to go to the microprocessor. Leave about 2 inches extra and cut off the excess wire. Strip about 1/2" from each end of the wires.

4. Remove the microprocessor amd solder one end of the wires to pins 1 and 2 of the microprocessor or socket.

5. Drill a hole and mount the switch to the computer. Solder the other ends of the wires to the two tabs on the switch.

6. Plug in the microprocessor. If a socket is used then plug the microprocessor into the socket and the socket assembly into the microprocessor's socket.

The installation is complete. You can arm the reset button by entering values into locations 265 - 267 as discussed previously.

128K MEMORY EXPANDERS

PRODUCT INFORMATION EDITORIAL

What can you expect from our 128K memory expanders? If I were considering purchasing one then I would want to know what it would do for me. Let's look at the operation of the computer first. The computer's memory is partitioned so that the lower 32K bytes are random access memories (RAM) and the upper 32K bytes are read only memories (ROM). This is the normal power up mode and is called memory map type 0. If you have a 32K computer then you can fully use the 32K of memory. However if you have a 64K computer then you are not much better off than with the 32K arrangement. Some word processing, basic, and machine language programs partition the memory map so that all of the memory is RAM. This is called memory map type 1. They transfer the information in the ROMs to the RAM. However this is a duplication and wastes 16K bytes for the basic and extended basic ROMs. So you have from 8K to 16K more of memory you can use with this method.

We designed our 96KX expander so that both 32K memory banks can be used while retaining the upper 32K of ROM. Word processors, file manager programs, and basic programs can be modified not to configure the memory as all RAM. Patches need to be made to allow text or data to be stored in the other 32K of memory which can be accessed by software programs or our 96KX modules or cartridges.

Now back to the 128K and larger sytems. Since most software is

designed for a 64K or smaller computer, then it is not directly compatible with larger systems. There are chips available for managing memories up to a couple of million bytes. We had to decide on which approach we were going to take. Do we want the additional memory to only be used for storage or do we want the added feature of being able to write programs in any of it? The first option would be similar to a disk drive. We could load and save programs to the extra memory. However this takes additional software similar to the disk drive controller.

The latter approach is the one we decided upon. Since the computer contains software for initilization of memories in the basic ROM, we can use this for initializing the second bank. When the power switch is turned on the computer goes through a hard reset routine and conditions the memory for basic operation. With a 128K memory system configured as two 64K memory banks, only one bank will be initialized when the computer is turned on. We include a 3 position switch with our memory assemblies so that in the "LEFT" position bank 1 is selected and in the "RIGHT" position bank 2 is selected. In the "CENTER" position the banks can be switched by software. After the first bank is initialized we need to move the switch so the second bank can be initialized. We will need to push the reset button after the second bank is selected to force it to be initialized for basic.

Now we can run two independent programs in either bank. We can't run both at the same time because both use the same microprocessor, SAM chip, and video display chips. As long as we are doing the same thing in both banks we can switch back and forth at will with no problems. Let's assume that we are running basic programs in both banks. Suppose we have a finance program in bank one and a telephone program in bank 2. Now suppose it is necessary to stop and make a telephone call. All that is needed is to press the "BREAK" key and then switch to the telephone program. After we enter "RUN" we are in the telephone program. Everything we were doing with the finance program is saved in the other bank including all variables and vectors. After we have finished with the telephone program we can return to the finance program and continue with no loss of data.

Well what about software control of the two banks? We have arranged it so that you can go under software control from one bank to the other. Also you can transfer variables or data from one bank to the other. To switch banks involves two memory pokes. If both banks are in the "RUN" mode then the banks will switch and the program will continu in the second bank. A variable can be transferred by poking a value into a memory location that is common to both banks and peeking this location after the bank switch. Of course all of these pokes are included in the instructions.

What about word processors that use graphic displays? You need to load the software into each bank. If both programs are doing the same thing then you can manually switch banks. For instance suppose we are writing this article in one bank and we want to stop and edit a brochure or write a letter. We can switch banks and use the second bank without distrubing our text in the first bank.

Now can word processors be in one bank and basic programs in the other bank? The answer is "yes" if you can configure the SAM chip and VDG chips to be the same in both banks. This is automatically done if basic or the same word processor is run in both banks.

You may wonder what is included in our 128K expansion kits? Because we did not have to write additional software since we depend on basic for our additional bank's initialization, we were able to produce a compact control circuit in two modules. These modules mount

under a PIA chip and the SAM chip. Connected to the modules is the 3 position switch which can be mounted in a 1/4" hole under the front lip, plus a 64K bank of memory chips with sockets mounted to them. If you have a 64K computer then your chips fit in these sockets in our ME-128-64. For the other upgrades we supply all of the memory chips.

Does the upgrade cause additional heat and is it hard to install? Anytime you add more chips you will create more heat. However the control circuit does not require much power and the unselected bank is automatically placed in a power down state where it draws about 1/10 of its regular power. The whole assembly draws less power than a piggy backed 32K memory upgrade. The installation is easy. Remove the PIA and SAM chips plus the memory chips and install our assembly in the vacant sockets. Next plug the PIA and SAM chips into the sockets on top of the modules. Then mount the switch at any convenient location.

In summary the 128K expanders mount inside the computer, support all software, do not require any soldering or trace cutting for installation, allow banks to be switched by software or manually, and allow variables and program control to be transferred from one bank to the other. When used with our 96KX-M it has the same computing power as 4-32K computers.

64K MEMORY SOLUTION

96KX PRODUCT INFORMATION EDITORIAL

You have just purchased a 64K computer and can't use all of the memory. When you ?MEM you get a value between 22000 and 30000. It would seem logical to have at least 50000 for a 64K machine. To find out what is happening you ask your Radio Shack dealer but he does not give you a satisfactory answer. Next you turn to the magazines and find an article here and there telling you how to use 8K more of the hidden memory which is your other 32K.

The purpose of this article is to explain the problem and show how we solved it. The engineers who designed the color computers chose a powerful memory controller chip the MC6883 which is known as the SAM chip. SAM stands for synchronous address multiplexer. The 16K - 4116 and the 64K - 4164 chips are called dynamic RAMS. RAM stands for Random Access Memory. Dynamic RAMS require refresh signals at minimum specified time intervals to keep from loosing the information stored in them. The SAM chip provides these signals plus the clock signal for the microprocessor plus it selects which memory devices are to be turned on. When you turn the computer on then the Basic ROM is activated by the SAM chip.

There are two modes of operation for the SAM chip. The first is called memory map type 0. In this mode the memories are arranged by the SAM chip so that the upper 32K of the memory map is occupied by ROM and the lower 32K is occupied by RAM. Now for 64K computers this means that there are two banks of 32K RAM in the lower memory. This is the normal mode of operation for Color Computers. The two banks are called memory page 0 and memory page 1. Don't confuse this with graphics pages since we are talking about 32K memory pages. Now the only thing common to both pages is the ROM memory area. So software is needed in the ROM area to access both memory pages.

The Basic ROM initializes the first page for operation when the computer is first turned on. However it does nothing to the second memory page so Basic is not initiliazed for memory page 1. We solved

this problem by putting a ROM in the upper 8K of the ROM memory. There are 4-8K memory blocks that can be used for ROMS. The first is for the Extended Basic ROM, the second is for the Basic ROM, the third is for the expansion or cartridge port and the fourth is generally not used. By being able to fully use all 64K of RAM plus having 32K of ROM, the Color Computer becomes a 96K computer. So we called our ROM Software the 96KX.

In order to put an operating system in page 1 it seemed logical to transfer all of page 0 to page 1. Since page 0 automatically initializes for Basic, when we transfer each byte to the corresponding byte in page 1 then we should have Basic also in page 1. This is one feature of our 96KX. To switch to page 1, it is necessary to poke any value into location 65493. Basic works fine except the screen does not work. So we decided to see if we could exchange the bytes in page 0 with those in page 1. If this works then we would be able to always operate in page 0 where we could see our results on the screen. So we wrote a machine language program to do this exchange. This was incorporated into the 96KX software.

Sometimes it is advantageous to have more memory for data storage. It would be nice if we could transfer files from one 32K memory page and store them in the other 32K memory page. So to take care of this need we wrote subroutines to pass blocks of data from page 0 to page 1 and from page 1 to page 0. This we incorporated into our software.

Now that we have both 32K pages at our disposal, we noticed that we had quite a bit of ROM memory left. So we decided to incorporate routines to aid in the everyday operation of the computer. We added a memory scan routine that displays the values stored in memory in decimal, hex, vector, or ASCII formats. We can also change any value stored in memory, change any one statement number, enter data in mixed decimal or hex, convert decimal to hex and hex to decimal. These saved us from having to load a utility program when we need to develop or fix programs.

One thing that has always been a hassle is determining the beginning, ending and execute address of machine language programs. So we included this in the software. Sometimes our computer would hang up and we couldn't reset it with the reset button. So we added an interrupt switch and software to allow a hard reset when the computer is interrupted by our button. This works even when the computer displays an out of memory error. For example POKE 25, 200: NEW. You will get an error message. Then ?MEM and you will get an out of memory error. Previously we had to turn the computer off before we could get it to run again. The interrupt switch allows a hard reset and restores the computer even when an out of memory error occurs.

After developing the 96KX software we put it in a cartridge with an output connector and an interrupt switch. This worked as expected with disk drive and accessories. The software proved so useful that we looked at putting it inside so it would always be available. We developed a couple of modules that mount under the microprocessor chip and the Extended Basic ROM. This is really nice as the software is now a part of our computer. We mounted a push button switch under the front lip of the computer to provide the hard reset feature.

In summary I believe the 96KX is the best product we have developed. It allows us to use all 64K of RAM plus retain the Basic, Extended Basic, and 96KX ROMS. It is a permanent part of our computer and a real asset.

UTILITY PROGRAM

This handy program can be used if your main program bombs out. For example suppose you are running a finance program and for some unknown reason a hard reset occurs. When you list you get a screen full of odd looking characters that doesn't at all resemble your program. You have spent quite a bit of time on this program and don't want to have to start over. If you could just salvage part of it then all of the work would not have been in vain.

When the hard reset occurred then zeros were put into a large part of the program. Basic considers a string of zeros to mean the end of program. What you will have to do is to search through memory and find a place where the program has not been altered. Then you can place values in memory locations 25 -26 so that you can save the good part of the program.

The program we are going to give will allow you to peek into memory and find the good parts of the program. This may not be too easy a task but it is better than loosing your work. The utility program will need to be loaded into a memory location that you do not normally use for programs. ? PEEK (25) will give the most significant byte (MSB) of the basic starting vector and ? PEEK (27) will give the MSB of the ending vector. If the value in 25 is 30 or 38 then the utility program can start in a lower location such as 20. If you have a 32K or larger computer then it can start at 90. Remember the starting vector is 256 * PEEK (25) + PEEK (26). The value in location 26 is usually 1.

Let's assume our utility program is going at 20, 1. Each basic statement has to be preceeded by a 0 so we need to POKE 256 * 20, 0. Now enter "NEW" and when you get an OK you can enter the following program.

```
10' THIS UTILITY PROGRAM AIDS IN PATCHING
   PROGRAMS
15 INPUT "ENTER STARTING MEMORY LOCATION OR THE
   MSB OF THE VECTOR; M
20 IF M < 255 THEN M=256 * M + 1
25 A = PEEK (M) : A$= CHR$ (A)
30 V = 256 * A + PEEK (M + 1)
35 ? M ; A; A$; V
40 M = M + 1
45 X$ = INKEY$: IF X$ ="M" THEN 15
50 IF X$ = "C" THEN 60 ELSE 25
55 if X$ = "V" THEN 80
60 INPUT "LOCATION TO CHANGE"; L
65 INPUT "ENTER DECIMAL VALUE"; X: M=M-10
70 POKE L, X : GO TO 25
80 ?"THIS DETERMINES THE MS AND LS OF A VECTOR
85 INPUT" ENTER VECTOR VALUE ";VV
90 MS = INT (VV/256): LS = VV - 256 * MS
95 ?"V= "VV;"MS =" MS; "LS="LS
100 ?"PRESS A KEY TO CONTINUE
110 Y$= INKEY$: IF Y$="" THEN 110
120 GO TO 15
```

Statements 15 - 35 gets the value from the memory and from the next

location and displays the memory location, value stored, the ASCII character the value represents and the vector stored in location M. You can look at these values and see how your original program is constructed. In fact it is a good idea to use this program on a good basic program or on itself so you can get a feel of how a basic program is constructed.

All locations are not vectors. You should recognize the statement numbers in the vector column when the program advances to a good part of your previous program. After finding the first part of your program that is good make a note of its location. Then scan the rest of your program looking for a string of zeros. These you will want to replace with 32 which is the ASCII value of a space. Press the "C" key if you want to change a value. After you have patched the program you can put the vectors in locations 25 - 28 and save the program on a cassette or disk.

Since each basic statement is preceeded by a 0 it is easy to spot the beginning of a basic statement. The next location will give the vector to the next basic statement, and the third location after the 0 will be the basic statement number which is in vector format. Become familiar with the program before you need it and you will have it when it is necessary to patch a program. The utility program can be saved on disk or tape and loaded when you need it. Our 96KX software has the equivalent of this program included with it as one of the features.

OPERATING HINTS

If you have an operating hint you would like to share then send it to us and we will print it here.

DELETING CHARACTERS

When editing basic statements using the extended basic editor and you need to delete several characters, don't bother to count the number of characters just press the "D" key to delete one character at a time while observing the characters on the screen.

MOVE THE CURSOR

The cursor's vector is in memory locations 136 and 137. You can move the cursor up N lines by subtracting 32 * N from the value stored in 136 and 137. You can write a basic or machine language program to do this.

QUESTIONS AND ANSWERS

If you have a question you would like to have answered send it to us and we will answer it here.

Question: Are your 128K memory expanders compatible with my existing software? I don't want to have to buy more software.
Answer: Yes. We designed them to be completely compatible with all software. See our editorial in this issue.

Question: I have a friend who has a Model-4 from Radio Shack. Will his programs work on my color computer?
Answer: It is doubtful that you can use the programs easily. However if they are programs that do not use any special ROM calls, require printing to specified memory locations, or obtain information from designated memory locations, it may be possible to use them. You need a terminal program so the Model-4 programs can be downloaded to you in ASCII basic format. The Model-4 uses a different microprocessor but still uses Microsoft basic so programs can be transferred by ASCII. See our editoral on ASCII in this issue.

Question: My computer works for a while but after a few minutes the screen does funny things. What could be the problem?
Answer: One of your chips could be going bad. The suspects are the Microprocessor chip (6809E), the PIA chip (6821), and the SAM Chip (6883). The cheapest of these is the 6821 and there are two of them. The microprocessor is next and the most expensive is the SAM chip. The microprocessor is a good candidate. It is a good idea to have a spare set of chips, at least one of each of the 3 mentioned. Replace them one at a time until the problem disappears.

NEW ADVANCED COLOR COMPUTERS

These are the most advanced color computers you can buy. The 96KX-M allows you to use all of the memory in 64K and 128K computers plus its powerful utilities are available for program development. It is installed so you always have this feature. We are the first to advertise 128K memory expanders and are offering these installed. With the 96KX-M and 128K memory you have the equivalent computing power of 4-32K extended basic computers. Also included is our multiprogram manager software which will allow you to load up to 5 programs in each 32K memory bank giving a total of 10 menue selected programs you can use in 64K computers and 20 for 128K computers about as much as a disk drive. With the 128K computers the banks can be hardware selected with the front mounted toggle switch or software selected.

NEW 64K COCO 2

EB 64K CC2	$299.00
96KX-M	59.95
VR-2	19.95
MPM SOFTWARE	14.95
LABOR	10.00

TOTAL VALUE	403.85
DEI PRICE	325.00

NEW 128K COCO 2

EB 128K CC2	$299.00
96KX-M	59.95
VR-2	19.95
MPM SOFTWARE	14.95
ME-128-64	199.00
LABOR	10.00

TOTAL VALUE	602.85
DEI PRICE	495.00

Computers are shipped by UPS. Add $5 for shipping. DCN Subscribers take 10% off of these prices.

****
* Please sign me up for one year for the DYNAMIC COLOR NEWS SERVICE. I understand I will receive a monthly news letter, Discounts on DYNAMIC ELECTRONIC INC. Computer products plus the Individual Reply to my Computer problems for a special of $10 each. Also I understand that there will be no charge for letters printed with answers in the Newsletter. Cost $15 USA & Canada, $30 foreign.
*
* Name _____ Mail payment to
* Address _____ Dynamic Electronics Inc
* City _____ P. O. Box 896
* State & Zip _____ Hartselle, AL 35640
* Enclosed is a check ___
* charge to VISA ___ MC ___ Number _____Exp._____
*
****

DYNAMIC ELECTRONICS INC.
P. O. Box 896 (205) 773-2758
Hartselle, AL 35640