For ALL COMMODORE Computer Users

Beginner, Advanced, and Professional

02356

COMMANDER

April 1984

$2.95 U.S.
$3.50 CAN.

INSIDE:

INTRODUCTION TO THE B-SERIES

Enlist your Commodore for
B*A*S*I*C TRAINING

BINGO CALLER

RESCUE

PRODUCT REVIEWS

0 71486 02356 04

C-64 VIC-20 PET VIC-20 C-64

Colin's Column

Reviews, Previews, News and Views

By Colin F. Thompson

Peace and quiet. Isn't it wonderful? The kids are asleep, your routine daily tasks are done and you sit in front of the computer with your mind at ease. Tonight, you say to yourself, I'm going to have some fun. It's just me and the VIC. Perhaps your new modem will take you to some interesting places and allow you to meet some new friends. Maybe the Christmas card list needs some changes, or a few letters could be written on the word processor. You might want to continue that BASIC program you've been writing. What you really do is not terribly important, it's the fun and excitement that counts.

So you sit down at your computer with high expectations, and what usually happens? Before you know it, chances are, the complexity of your task increases many times over, and you're awash in ambiguous technical decisions and unknown terminology. Your only hope is either to stick with the simplest of programs and projects, or to learn more about the inner workings and capabilities of your computer. My choice, naturally, is to pursue the second course, and acquire information through magazines, user's groups, and other sources. To that end, I'd like to take you on a guided tour of the VIC-20's inner memory.

A 64K VIC?

Sure. Every VIC-20 is a 64K microcomputer. The only catch is cost. When you first bought your VIC, it was missing some memory. Commodore left out most of the RAM memory chips so the VIC could be sold at rock bottom prices. If you want your VIC to do something besides play games, some extra memory must be purchased and then plugged into the "expansion port." That's the wide slot in the back of the VIC where game cartridges normally plug in. Extra memory cartridges come in four different sizes: 3K, 8K, 16K and 24K. The last three sizes are in increments of eight. That's important, so let's "expand" on the idea.

RAM AND ROM DEMYSTIFIED

Eight is a magic number in the VIC's language. It's so important to your understanding of how the VIC works that I am going to ask you to get a pencil and paper. If your understanding of computers could be improved on, please follow along, It is a very simple VIC (not 64) exercise, but to get a full measure of understanding, please draw the diagram as I outline it. First draw eight large squares from top to bottom. Draw them well, neatness counts. Under each box, starting with the one on the bottom, number the boxes zero, one, two and on through seven. (For some reason, counting on computers usually starts at Ø instead of 1.) What you are looking at is the basis of the VIC's memory. Hence the term "memory map." Now let's add some details.

Over blocks Ø, 1, 2, and 3 mark "RAM." Blocks 6 and 7 should be marked "ROM." Draw a horizontal line through the center of block 4 and label the bottom half "ROM" and the top half "RAM." That leaves Block 5 unlabeled. It is special, and should be marked "RAM or ROM." Carefully outline the borders of blocks Ø, 4, 6 and 7 so the boxes stand out. These blocks are included with the "bare VIC" you first bought.

The RAM-ROM concept must be mastered before we get any farther with our mapping. Each of the eight blocks represent 8K of memory. Eight blocks times 8K equals 64K of memory. The two kinds of memory used in the VIC are Random Access Memory (RAM) and Read Only Memory (ROM). All computers use a mix of the two types. The VIC is provided with ROM in blocks 4, 6 and 7. ROM is permanent memory. So what's in ROM? A series of machine language programs, written by Commodore, and placed permanently into the ROM chips. The programs allow the computer to function as a computer, rather than say a digital watch or a microwave oven. You could say that the contents of ROM give the VIC its personality.

RAM is a slightly different story. Programs or information stored in RAM may be changed. The purpose of RAM is to provide a storage place for the programs you buy or write. These programs let the VIC do some useful work. ROM retains its programs when the power to the VIC is turned off, but RAM forgets everything.

A STROLL DOWN MEMORY LANE

Starting from the top, we will examine each of the VIC's eight blocks. At the end of the examination, you should remember at least one thing: The VIC has a potential of 64K of memory, arranged in eight blocks of 8K.

Block 7 is top dog in the VIC. It is known as the Kernal ROM. The Kernal actually performs the instructions and operations that are started in other parts of the VIC. All Commodore computers have a Kernal ROM, and the different Kernals are so similar that many programs written for the PET and C-64 will run on the VIC.

Block 6 is the BASIC Interpreter. BASIC is a "high level" language. The grammar (syntax) of BASIC is much closer to English than the VIC's native language, Binary. The BASIC instructions that programs send to the VIC must be interpreted (changed) before they may be executed. This language translation is done in the BASIC interpreter. Each BASIC instruction in the program is examined by Block 6, and then reduced to a series of binary instruction codes. These codes are sent to the Kernal ROM for execution. Block 6 and 7 work closely together, thus forming the heart of the VIC.

Block 5 we have labeled RAM or ROM. This is the block that games and other cartridges plug into. When you plug in a cartridge, the programs stored in the cartridge's ROM enter the VIC through Block 5. When you turn on the VIC, it looks at Block 5 to see if a cartridge is plugged in. If so, the VIC executes the Block 5 program instead of displaying the power-up message. In computer terms, *execute* means to RUN the program, or carry out an instruction, not the gas chamber.

Most of the cartridges that plug into the VIC work from Block 5–but not all. Some of them enter in Block 5 but work from Block 3 also. More on Block 3 later.

Why have we labeled Block 5 RAM of ROM? If necessary, you may place 8K or RAM into Block 5. "Why would I want to do that?" you ask. The answer lies in Blocks 1, 2 and 3, so let's skip Block 4 for the moment and look in on RAMland (not L.A.).

In my two-year-long travel through the VIC-20, I have yet to encounter what I consider a good, useful, businesslike program that will run in the unexpanded VIC. All word processors, spreadsheets, databases and similar programs require at least 8K or expansion RAM to function. Since the VIC has only a paltry 3.5K of RAM when you buy it, the extra memory must be purchased and installed. Remember that expansion RAM memory comes in 8K chunks: 8K, 16K or 24K.

You need extra memory for two reasons. The programs are usually long. A typical business program needs 10K just for the program. Most programs need additional memory to store the data (names, addresses, etc.).

To illustrate this, let's use TOTL.TEXT (TT), the popular wordprocessor. The first consideration when evaluating a new program is "how much memory does it need." That's an easy question. Just read the requirements on the outside of the box. TT clearly states it needs 16K of additional RAM. If you don't already have a 16K RAM cartridge, you must buy one to use TT. 16K cards cost between $50 and $70. Let's assume you buy the 16K card and plug it into the VIC. When you turn the VIC on, the screen will now show that there are 19967 BYTES FREE. The 16K that you plugged in plus the 3.5K the VIC already had results in the number 19967.

Now you can load and run TOTL.TEXT. That takes care of the memory requirements of the program, or does it? Remember the second reason we need extra memory? Data. In this case, data is the letters and the other documents we type into the wordprocessor. With the 19.9K (call it 20K) VIC,TT will let you type in a good long letter, but there are limits to how long the letter can be. For the sake of explanation, let's say you can type in a 5-page letter with a 20K VIC. If you wanted to type a longer letter, you will need more memory. If you bought a 24K RAM card, instead of the 16K card, you could type in a ten-page letter. The extra 8K is used to store more data.

Blocks 1, 2 and 3 are where the "add-on" memory is stored. The 16K card we discussed is made up of two 8K blocks of RAM memory. When the card is plugged in, the chunks of RAM take up residence in Blocks 1 and 2. If you add another 8K chunk, it goes into Block 3. In most cases, the VIC is limited to 28159 bytes of available memory. That includes 24K in Blocks 1, 2, and 3 and the 3.5K built in.

The obvious question now is "If I plugged in an extra 8K into Block 5, will I have a 36K VIC?" Sadly, no. The VIC is arranged so that all the RAM memory used for BASIC programs must be contiguous – in a row, connected. Block 4 gets in the way, so in most cases you are limited to 29K. Before we get too confused, let's return to the map. Draw a large brace enclosing Blocks 1, 2 and 3. Label the point of the brace "BASIC program area for the expanded VIC."

Back to Block 5 – or "why put RAM in a block that BASIC can't use?" Thousands of programmers have spent countless hours figuring out something useful to do with Block 5. They have been quite successful. There is a scheme which lets you load a BASIC program of less than 8K in length into Block 5, and then use Blocks 1, 2 & 3 as storage for data. This nets you an extra 8K of data the program may use (a much longer letter, for example). The scheme requires an 8K RAM card to be plugged into Block 5. Several RAM cards have the ability to locate themselves into Block 5, as well as the normal 1, 2 & 3. Another use for RAM in Block 5 would be to store a machine language program. That works well. Did you ever wonder how Data 20 gets 80 columns on the VIC's screen? They put both ROM and RAM into Block 5 to do the trick.

The uses for RAM in Block 5 are endless, but one more example might give you something to say to your C-64 friends. If you were so inclined, you could buy a plug-in card that expands the VIC's memory to as much as 280K of RAM! The scheme is called "Paged Memory." I have one such device that allows me to use four different 16K "pages" of RAM. Each page may be "flipped" so that it is seen by the VIC in Block 5. Four pages of 16K equals an extra 64K of memory I can use to hold BASIC programs. It's getting confusing, but the ability of the VIC to see memory in a variety of places is one reason why I like the VIC. The C-64 offers a completely different kind of challenge. It's 88K of memory is arranged so that RAM or ROM may be selected at will. The C-64 is not a 'big VIC.' The two computers are only distant relatives, connected by marriage to the BASIC interpreter, the Kernal ROM, and certain input/output ports.

Block 4 is an active, multipurpose block. The bottom half holds 4K of ROM. This ROM makes the characters you see on the screen, hence the name Character Generator. The upper half is RAM, but is not used to hold programs. It keeps track of the colors of the characters and all of the Input/Output (I/0) functions. The I/O functions include sending and receiving data to and from the peripheral equipment attached to the VIC. The VIC commonly uses a disk drive, tape recorder, printer, joysticks and TV screen as peripherals. All information exchanged between the VIC and these devices passes through the I/O RAM section of Block 4.

Block Ø completes our tour through the innards of the VIC. I originally identified this block as one that is included in the bare VIC. Most of it is included, but a 3K chunk is missing. The VIC *can* be expanded in less than 8K chunks. A 3K RAM card is available which will expand the VIC's memory to about 7K total. The 3K card is not very useful, but if you only need a little more RAM, the option is open to you. There is a useful plug-in card that uses the 3K slot. It's called the Superexpander (SE). Commodore released the card just after the VIC was born. The SE adds either of two abilities to the VIC: More BASIC commands or 3K or RAM. You can use one or the other, but not both together. The extra commands let you write graphics and music programs without resorting to POKES, PEEKS or high level math. It's really fun to use, and can be mastered easily.

The rest of Block Ø includes the 3.5K of built-in RAM, screen memory, some "scratchpad" memory the microprocessor uses, and the cassette buffer.

A QUICK TOUR?

Yes, that was the quick tour of the VIC. It's an astounding machine. Remember, the purpose of the tour was to add to your understanding of how the VIC uses commercial programs. Now, when you run into a snag, you should have an idea of where to look to find the problem. For a more detailed explanation of the VIC, pick up a copy of the VIC-20 Programmers Reference Guide. It's about $15 and worth every cent.

RAMAX TO THE RESCUE

I recently added a third VIC system to the technological paradise I call home. (It must be seen to be believed.) Since the new system is used exclusively for word processing, I had to add 16K of RAM and an "expansion" card. The expansion card is a simple printed circuit card that plugs into the Game Slot. This brings the Expansion bus outside the VIC so I can plug in more than one cartridge at once. For my W/P system I need three slots: one for Quick Brown Fox, one for the Data 20 card and one for a RAM card. My other two VICs use Cardco's CardSlot3/s as an expansion card. It works well and has some useful features not found on most cards. This time, however, I wanted to try a different arrangement.

The marketplace is flooded with various brands of RAM cards and Expansion cards. Some are simple and others are quite complex and powerful. After examining the features of many of these devices, I settled on one that fulfilled all my requirements in one neat package. The product is called RAMAX and is made by Apropos Technology. Ramax combines the function of a RAM card and an Expansion card into a single plug-in unit measuring 7½ by 5½ inches. Two expansion ports, facing up, are present at the rear of the unit. Any two cartridges may be plugged in at once. In my case, it's QBF and the Data 20 card. Ramax has four chunks of RAM memory onboard: Three blocks of 8K and a 3K block. A bank of DIP switches allows you to add or subtract memory as you like. For example, you could configure the VIC to have an extra 16K of RAM by turning on the switches marked BLK1 and BLK2. Turn BLK3 and you get 28159 bytes (29K). The Switch marked "RAM" puts 3K of RAM into Block Ø. Remember that the 3K and 8K blocks are not to be used together. It doesn't work like that. By switching on BLK5, Ramax lets any one cartridge that operates from Block 5 be used. Note that only one such cartridge may be used at a time.

Why am I using two cartridges at once? After reading our earlier discussion on Blocks, perhaps you've figured out that

one of the two cartridges doesn't operate out of Block 5. Right. QBF, even though it is a ROM cartridge, operates from Block 3, so there is no conflict with the Data 20 card which uses Block 5.

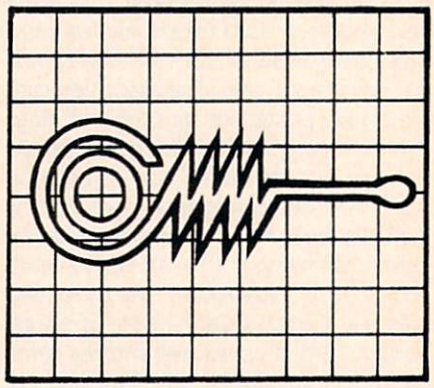
The BLK5 switch on Ramax doesn't put 8K of RAM into Block 5; it just lets any cartridge that uses Block 5 be seen by the VIC. The last of the six switches on Ramax is a reset switch. If you switch it on, then off, the VIC will be reset to the same condition as then power is first turned on. Ramax is protected by a fuse in case a cartridge malfunctions. If this happens, the fuse blows, not the VIC.

In the October issue of *COMMANDER*, I outlined a way to get 31K FREE with the VIC. The scheme called for a 24K card, a 3K card and a Data 20 card. If you plug in these three devices at once and enter POKE 642,4:SYS58232, the result is 31743 BYTES FREE. The trick works well with Ramax, since it provides all the memory needed, plus the slot for the Data 20 card.

I've used the $99 Ramax for 4 months and have encounted no compatibility problems with the horde of cartridges I've tried in it. The user manual is one of the better documents of it's kind. It includes a short BASIC program you may enter that tests the memory. It is covered by a six month limited warranty.

REMEMBER THE C-64?

Apropos also has some goodies for the 64. If your collection of 64 cartridges numbers more than one, you should be looking for an expansion card. Apropos' expansion card will hold up to four cartridges at once. Each of the four slots is activated by a switch. This will save wear and tear on the edge connector of your 64. It also lets you use more than one cartridge at once, and the $49.95 price tag puts it into the "affordable" range. Educational software for the VIC and C-64 is also marketed by Apropos Technology, 350, N. Lantana #801, Camarillo, CA, 93010. (805) 482-3604.



COMMAND POST

By Jim Grubbs

Before long the warmth of spring will return to the air. The trees will begin to fill out with the promise of the hot summer days to come. While there is still just a bit of chill in the air, and before outdoor projects start to dominate the time spent on our hobby, let's finish up the radio teletype project that we started last month.

BIG MOUTH

With last month's program we gained the ability to copy amateur radio and commercial radio teleprinter transmissions (RTTY) using either the VIC-20 or the C-64. Being a good listener is important, but being able to speak can be fun too!

This month, COMMAND POST RTTY TX gives us the other half of the teletype system. Last time we found a Henry Kissinger type who spoke fluent Baudot code and could translate it into ASCII so our computer could print it on the screen. Now we make one more request of our translator. We wish to respond to the messages we are receiving, but we only speak ASCII and our listeners understand only Baudot.

It is as simple as reversing all of the translations we did last time. In the program our transmit translation table is located in lines 440 and 441. L1$ converts the letters, and F1$ converts the numbers and punctuation. All of those "Ds" in F1$ represent space characters and fill in the table for illegal code combinations.

As we did in the receive program, we open the RS-232 channel in line 420 and set the speed to 60 words per minute (approximately 45 baud) in line 430.

COMPATIBILITY THROUGH COMPUTER DATING

COMMAND POST RTTY TX has been designed so that it can be merged directly with last month's RTTY receive program to give us a complete amateur RTTY terminal program. When this program is used in conjunction with last month's receive program, lines 420 and 430 must be deleted. Line 455 should be changed as noted in the REM statements in the program so that when the F1 function key is pressed it will toggle the program from send to receive and back again. Be sure to add line 185 to the receive program as indicated in the REM statements.

LET US BUILD ONE FOR YOU

It is possible to customize this program much as we did with COMMAND POST TX for CW transmitting. You can add preprogrammed messages and assign them to the remaining special function keys. By adding a sub-routine to check for these keys you will be able to send messages with a single key. Refer to the November 1983 COMMAND POST and see if you can implement this feature.

GETTING SHIFTY

The final step in getting on the air is to convert the TTL level computer code coming out of the VIC or C-64 on pin M into shifted audio tones corresponding to mark and space. Our interface will use the same integrated circuit used in many of the commercial interface units. The XR-2206 IC is available from many sources. One is included in the accompanying parts list. The XR-2206 is a very sophisticated waveform generator. For our use, the filters have been set to deliver a fairly pure sine wave signal. We set the two tones by adjusting the variable resistors connected to pins 7 and 8 of the chip. Normally, we will want to set them for 170 hertz shift, so we set the low tone at 2125 hertz and the high tone at 2295. Other combinations can be used, but these are the standards. It is recommended that the frequencies be adjusted using a frequency counter connected to the output of the chip (pin 2). The output from pin M on the user port should be fed to pin 9 on the XR-2206. Wiring is not critical. I built mine on a proto board. After correcting some questionable connections mine worked quite well. It is recommended that the interface be powered from a separate power source. Even a nine volt transistor battery will do to test the unit. A warning once again to those of you who insist on trying to power everything from the VIC or C-64 – DON'T DO IT! The amount of current available from the computer is quite low. You endanger not only your project but your computer as well when you draw too much current from the user port. All it takes is a momentary short – and "poof". For those of you asking about the availability of the user port connector, I have included a source for it as well in the parts list.

LOGGING WITHOUT TREES

One of the bookkeeping functions that our machines can do for us in the ham shack is logging our contacts of QSOs, as hams call them. It's great to have a virtually

instant "memory" afforded by an all electronic data base. In contests, computer logging and duplicate contact checking have become almost a necessity. Numerous articles have appeared over the past few years on performing these functions on many of the popular computers. Many are adaptable to the VIC and the 64.

If you are not big on handling data files and sorting routines, HAM DATA may have just the thing for you.

Chip Lohman, NN4U, is the programmer behind HAM DATA software. It is apparent from the end result that a lot of time and thought have gone into his programs. Let me concentrate on SUPER LOG IV for this discussion.

This is another program that will take some time to get used to. It took me about an hour to get accustomed to the necessary inputs and start to feel comfortable finding my way around the menus. The documentation is fair, but could give some more specific examples.

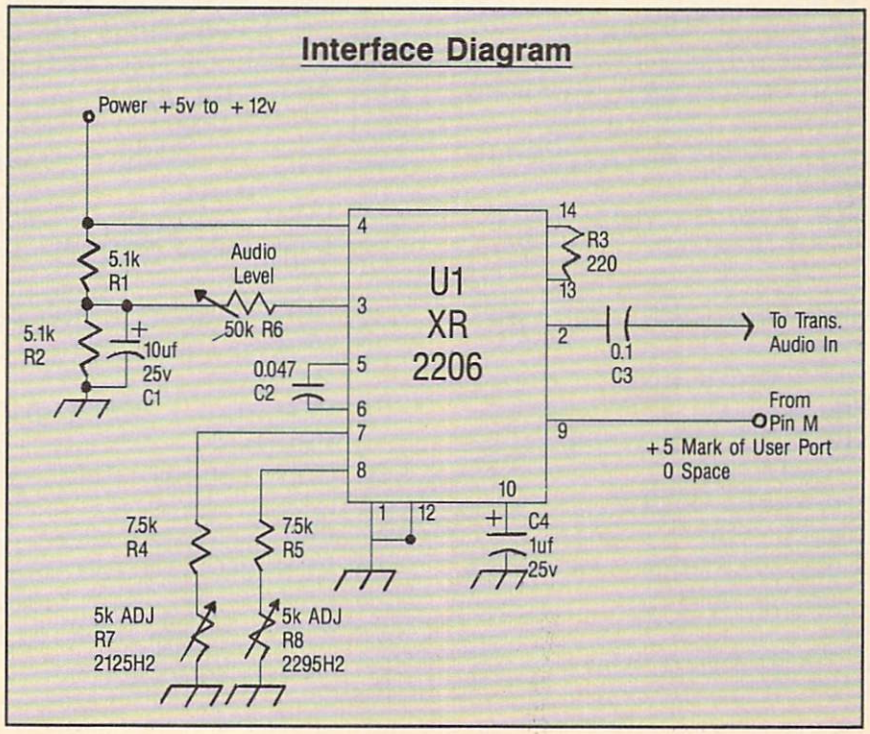
I used the C-64 version, though the program will work on a VIC-20 with a minimum of 8K expansion. The program is available for disk only. With a C-64 and SUPER LOG IV you can have approximately 525 files available at one time. With an 8K VIC that number drops to 88. The amount of memory used by the program itself is kept at a minimum by segmenting the various features into separate programs which are called by the main menu. While test driving the program I found all the loading to be a bit of a nuisance. In regular operation, where you would normally be logging and searching for long periods of time and only later printing the log or creating a summary of your contacts, the re-loading would be minimized. SUPER LOG is very picky about how you enter data–but most data base programs are.

But for these few drawbacks, SUPER LOG IV performs like a champ. It allows you to load your previously saved logging files. You can then view the file, one record at a time, or all of them at once. You can select entries that only show 40 meter contacts, a particular call sign, only people named Jim (or whoever), location, or a date of your choice. The login function is the work-horse where you input all of your QSO information. The change mode allows changes to be made to previously logged entries, but this is one of the auxiliary functions and requires the change routine to be loaded from disk. If you are into hunting states of DXCC countries, SUPER LOG IV will, with your help, keep your states and countries totals up to date.

One feature included is an alarm function. If you select auto-time where you enter the beginning time of the session, you will also be asked if you would like to set the alarm. Presumably, the alarm will remind you of dinner, your favorite TV show, or that your name is mud if your spouse doesn't see you by a certain time. The only problem is that you are only notified that you are "in alarm" when you return to the first prompt while logging. This is no problem during a contest, but if you are just making routine contacts, it could be quite some time before you make your next entry, assuring you a place in the doghouse!

You can print the entire log, or selected portions of it. You can even print the information part of a QSL (confirmation) card. That could be particularly handy if you find yourself on the DX end of pile-ups. Once again, the printing routines are auxiliary programs. The log must be saved and reloaded in order to use the printing functions.

I also took a quick look at CONTEST LOG, a three-in-one program for Sweepstakes, Field Day, and a universal program that will handle most of the other contest formats. These programs do all of your duplicate contact checking for you and even print your log when you are done! Unlike SUPER LOG, all features are available in one program. I can't wait for Field Day when I get to put this one to the



real test. CONTEST LOG is available for the VIC-20 with 8K expansion or the 64 on either disk or cassette. Suggested retail is $17.95 on cassette with a $3.50 additional charge for disk.

SUPER LOG IV retails for $21.95 on disk only for either VIC or the 64. More information is available from HAM DATA, 3331 Bybrook Lane, Woodbridge, Virginia 22192. If you are looking for a truly sophisticated logging program, SUPER LOG IV is highly recommended. For contests, CONTEST LOG is a good buy.

WAITING IN THE WINGS

The post office box usually contains correspondence from a COMMAND POST reader just about every day. Most of your comments so far though have concentrated on either very general information requests or questions specific to the columns that have already appeared. Now I'd like to know what you most would like to see covered in these pages. If you are short on ideas, here are a few currently being considered: creating video graphics for amateur TV, using the 64 as an audio-frequency waveform generator, a machine language CW receive program, transmission of computer programs over the air, repeater control. Several other topics are also being researched but will require a bit more "cooking" before they are ready to serve: SSTV reception, using the analog to digital converters available in the VIC and C-64 for interfacing to such things as antenna rotors, AMTOR reception and transmission, and packet radio. That's not a complete list, but it should get your mind started. Drop me a note and help determine the future contents of COMMAND POST. My address is: Jim Grubbs, K9EI, PO Box 3042, Springfield, Illinois 62708.

PARTS LIST

R1,R2 ----- 5.1 K ohm resistor
R3 --------- 220 ohm resistor
R4,R5 ----- 7.5 K ohm resistor
R6 --------- 50 K ohm single turn trim pot
R7,R8 ----- 5K ohm 10 turn trim pot

C1 --------- 10uf 25v tantalum capacitor
C2 --------- 0.047uf Mylar capacitor
C3 --------- 0.1uf 50volt disc capacitor
C4 --------- 1uf 25volt tantalum capacitor

U1 --------- XR-2206 Exar integrated circuit

12/24 pin user port connector: Sullins 06SUL1224E5 available from:
Priority One Electronics
9161 Deering Avenue
Chatsworth, California 91311

All other parts are avaialable from:
JAMECO Electronics
1355 Shoreway Road
Belmont, California 94002

```
410 PRINT''[clr][rvs-on][sp]COMM
    AND[sp]POST[sp]RTTY[sp]TX[s
    p]''
420 OPEN 2,2,0,CHR$(96+1)
430 POKE 665,211:POKE 666,87
440 L1$=''CYNIAMZTFKOR£LXVWJEPG
    ↑S]UQ''
441 F1$=''MDTIDZQORDDLC£]VWSAJP
    UGFXN↑DDDYD''
450 GET X$:IF X$=''''THEN GOTO
    430

452 IF X$=CHR$(34)THEN X$=CHR$(
    39)
455 IF X$=''''  THEN END
460 PRINT X$;
461 IF X$=CHR$(13) THEN PRINT#2
    ,''H'';:GOTO 450
462 IF X$=CHR$(10)THEN PRINT#2,
    ''B'';:GOTO 450
469 IF X$=CHR$(32)THEN  PRINT#2
    ,''D'';:GOTO 450
470 X=ASC(X$)
471 IF X<33 THEN GOTO 450
472 IF X<65 THEN X=X-32:X$=CHR$
    (91)+MID$(F1$,X,1):PRINT#2,
    X$+CHR$(95);:GOTO 450
478 IF X>95 THEN GOTO 450
480 X=X-64:X$=MID$(L1$,X,1)
490 PRINT#2,X$;
500 GOTO 450
600 REM IF THIS PROGRAM IS USED
     WITH
610 REM COMMAND POST RTTY RX
620 REM THEN DELETE LINES 420 A
    ND 430
630 REM CHANGE LINE 455 TO
640 REM IF X$=''''  THEN GOTO 26
    0
650 REM ADD LINE 185 TO THE RX
    PROGRAM
660 REM 185 IF A$=''''  THEN GOT
    O 410
700 REM COMMAND POST RTTY TX
710 REM BY JIM GRUBBS
720 REM     PO BOX 3042
730 REM     SPRINGFIELD IL 62708
740 REM     (C) 1984
```

B*A*S*I*C Training

By Jonathan Secaur

Last month our hands-on, self-teaching course in Commodore BASIC introduced the PRINT statement and the concept of numeric variables. This lesson focuses on another, important type of variable that is used for "character strings."

Lesson 2

VARIABLES AND MORE VARIABLES

In Lesson 1 you learned that a letter or a group of characters starting with a letter make up a **numeric variable,** a symbol that stands for a number. Most microcomputers only recognize two-character variable names, and the first must be a letter. A, RV, and Z1 are some examples.

1) Put a T by each set of letters or numbers that could be a numeric variable, and an F by each that could not be.

_____ET

_____C3PO

_____K9

_____3D

_____Q

_____29

2) Now you know how to let something stand for a number in BASIC, but we also need something to stand for a *name* or other *words.*

Here's an important definition: anything in quotes is called a **string.** Your name (inside quotes), your address (inside quotes), anything at all inside quotes is a string. That's short for saying it's a string of characters, or a character string.

If we use numeric variables to stand for numbers, you can guess that we use **string variables** to stand for strings. A string variable is written just like a numeric variable, except that string variables always end with a string sign, $. That's the same thing as a dollar sign, but it has nothing to do with money. When you see that symbol, just read the word "string."

Some possible BASIC string variables would be A$, ZZ$, B1$, and even SHOE$ or KITE$. (If you missed the jokes, please read the previous paragraph again!)

To see how it works, type in these lines, pressing [RETURN] after each line:

```
NAME$ = "[type your name here]" (actually put your
  name between the quotes)
PRINT NAME$
```

and the computer will be glad to print out your name.

3) You can save some time and space by typing more than one command on a line if you put a colon (:) between the statements to separate them. Try this one:

```
A$ = "CAR":B$ = "NATION":PRINT A$ + B$
  (remember to press [RETURN])
```

If you ask the computer to PRINT 4 + 3, it will give you the sum, 7. How can it add two words together? As you can see, it just sticks them together with no space in between. Adding two strings together is called **concatenating** them. Next time someone asks you what you've learned about computers, tell them you know how to "concatenate strings," and they will surely be impressed.

You could make the computer leave a space between the two words if you include a space between them somewhere, in quotes. You could say,

A$ = "CAR":B$ = "NATION":PRINT A$ + "" + B$

4) If you use a string variable (a variable name followed by a $), you must set it equal to something in quotes. If you use a numeric variable, you must let it equal a number.

See what happens if you don't, by typing in these lines:

A$ = ANGELFOOD
A = "5"

What does the computer say if you use the wrong kind of variable?

6) By the way, since the computer reads from left to right, the variable name must always be on the left side of the equals sign.

You can say USEDCAR = 1975, but not 1975 = USEDCAR.

See you next time.

Answers to Lesson 2 Questions:

1) T, T, T, F, T, F

4) ?TYPE MISMATCH ERROR That just means that the information you supplied was not the same kind, or of the same type, as the variable used.

BASIC Education: Programming for Learning
Part IV-Getting into Inputs

By Andy Van Duyne

IN SEARCH OF A BETTER MOUSE . . .

All good educational software requires some sort of input from the user. There are a number of alternative input sources: joysticks, graphics pads, paddles, light pens, "mice," and even the voice. Unfortunately, most students are left to do battle with a woefully inferior device–the keyboard. Due to the many frailties of the QWERTY (standard typewriter keyboard) system, this haphazardly arranged device is probably the single greatest impediment to enjoyable and efficient computer use. Even experienced typists are always vulnerable to "typos", and young users often spend much time in the frustrating pursuit of the right key. Commodore has tried to make things easier for VIC-20 and C-64 users by including those large, isolated function keys–but try to explain their numbering to a youngster. (I have found that, to get any real use out of them with kids, it was necessary to cover the keys with some kind of label, transforming them to 1-2-3-4 or A-B-C-D). Anyway, most of us have just the keyboard with which to work, and we must make the best of the situation.

One good thing about the keyboard as an input device is its universality–virtually every computer has one. A programmer trying to write for a number of unknown future users can almost certainly rely on this device as being part of the setup!

MELODIC INPUTS

It is good practice to design your input techniques to minimize common mistakes, and accept only "correct" responses. Let's look at a few techniques by writing a program to give the user practice in "ear training." A short melody (5 notes) will be played, then the user must enter the scale degree or note (nos. 1-8) in the proper sequence, one at a time. The user can also choose to hear the "mystery tune" or the major scale at any time. We want to create an input routine to satisfy the following specifications:

1. The numbers 1-8, standing for pitches, will be accepted as input. [RETURN] must be pressed for the answer to be finally entered. This will allow the user to think about an answer and change it if desired.
2. By pressing "T" alone, the mystery tune will be played. ([RETURN] not required)
3. By pressing "S", the major scale will be played to help the user establish the tonality of the tune. (Again, without [RETURN])
4. By pressing the [DELETE] key, a response may be cleared so that another can be entered.

Let's examine some of our options:

User input from the keyboard can take several forms-numbers, letters, words, single keypresses, control functions (on the C-64), and special characters, such as delete, back arrow or the function keys. Perhaps the weakest (though simplest) input technique is this:

```
INPUT X
```

Of course, you know that this is looking for a number. You can carefully print a screen message to the effect of "ENTER A NUMBER FROM 1-8." You may also know, if you have done this for any period of time, that kids (and even adults) will hit one of the nearby letter keys instead of a number with annoying frequency–"Q" instead of the "1", for example. This results in the all-too-familiar "REDO FROM START" message, and often the scrolling of the text, obliterating your carefully designed and beautifully prepared screen. This is because the response to an INPUT statement is PRINTed on the screen, even before you touch [RETURN].

One solution to this problem is to use a string variable input:

```
INPUT X$
X=VAL(X$)
```

Under these conditions, the errant letter "Q"would not result in the error condition, but would return a value of zero for the variable X. This solves part of the problem, and we can pick up the letters "S" and "T", but still allows for plenty of accidents. What happens, for example, if the student hits the [HOME] key, or [CLR], or those infernal repeating cursor keys. There goes your nice screen! The INPUT statement also cannot read the function keys, should you want them to be used.

There is also the possibility of PEEKing at address 197, which will return a value for the current key being pressed. This would be a bit convoluted, however, especially in handling the RETURN requirement of our first specification.

DO NOT FORGET "GET"

As you may have guessed, the core of the input routine we will develop is the GET statement. Perhaps the most common use of this statement is as follows:

```
100 PRINT "TOUCH A KEY TO CONTINUE"
110 GETA$: IF A$= "" THEN 110
```

The GET statement takes a quick glance at a particular input device. In this example, it looks at the default device, or the keyboard. If, at the time the statement takes the look, there is a value being generated (in this case, a code being returned from a pressed key), that value is assigned to the variable. This method of

securing input is very useful when working with young children, as it requires less keyboard manipulation. Using GET and string variables, we can read ANY key on the keyboard, including the cursors, delete, and function keys. Unlike the INPUT statement, GET does not print its character on the screen—meaning that [HOME], [CLR], and the cursors can be pressed without destroying your carefully planned screen. But this also means that we have to do a little more work in the input routine, so that the user does not end up typing "blind".

Before using a GET statement, you should always include the following:

POKE 198, 0

This clears the keyboard buffer. Often a user becomes either impatient or excited, and will press a key a number of times to enter just one response. When the GET statement looks at the keyboard, it actually checks the bottom address of the keyboad buffer (a section of memory that stores multiple keypresses.) The GET statement will take it instead of the most recent keypress, which will be topmost in the buffer.

THE WHOLE INPUT ROUTINE

An annotated version of the recommended routine is shown in Figure 1.

This routine fits our specifications, and prohibits the entry of any erroneous values. About the only thing a user can do to crash this routine would be to press the RUN/STOP key, which can be defeated elsewhere in the program if desired. While this particular configuration was designed for the accompanying programs, the techniques can be applied to any number of situations. One note—the DELETE function is not really required in this case, as the variable AN$ is always updated to equal the latest valid entry. It is a good idea to include it, though, as it will be required when this routine is expanded to handle answers which are longer than one character.

Remember, the goal of an "educational" program is NOT to confound and frustrate the user with beastly input routines. Given the shortcomings of the keyboard, the program should do as much as possible to help the user to be successful in demonstrating knowledge of the subject matter. It should not be a drill on typing skills.

If you have any comments or suggestions about this series, please send them to me in the care of *COMMANDER* Magazine. 'Til next time . . .

```
200 REM INPUT ROUTINE
201 POKE 198,0 : PRINT D$SP$D$ "WHICH STEP? "AN$
     (clear buffer- clear screen space-place
     message-print current answer value)
202 GET A$ : IF A$ ="" THEN 202
     (check for keypress- loop if none)
203 IF A$ = CHR$(13) THEN 209 (check for RETURN)
204 IF A$ = "T" THEN GOSUB 220 : AN$= "": GOTO 201
     (intercept tune request:clear answer- reloop)
205 IF A$ = "S" THEN GOSUB 230 : AN$= "": GOTO201
     (intercept scale request:clear answer- reloop)
206 IF A$ = CHR$(20) THEN AN$= "" : GOTO 201
     (check for delete key - clear answer- reloop to
          print message)
207 IF A$ < "1" OR A$ >"8" THEN 202
     (check for any other invalid key)
208 AN$ = A$ : GOTO 201
     (accept pressed key as the answer- reloop to get
      either RETURN or DELETE)
209 IF AN$ = "" THEN 201
     (check for null string as answer)
210 RETURN (everything is O.K.- return with this value
      for AN$)
```

Ears – VIC-20 Version

```
10 REM EARS (VIC)

12 X=RND(-TI)

20 PRINT''[clr]''SPC(207)''EAR
   S''SPC(100)''BY[sp]ANDY[sp]
   VAN[sp]DUYNE''

21 FORP=1TO2E3:NEXT:PRINT''[do
   wn, 3 times]TOUCH[sp]A[sp]K
   EY...''

22 GETA$:IFA$=''''THEN22

30 REM SETUP STRINGS

31 D$=''[home][down, 16 times]
   '':SP$=''[sp, 21 times]''

35 E$=''[clr][rvs-on][sp, 9 ti
   mes]EARS[sp, 9 times]''

40 REM SETUP SOUND

42 S=36876:POKES,0:POKES+2,15

43 DIM T%(5),A%(5),P%(8)

44 FORN=1TO8:READP%(N):NEXT

46 DATA201,207,212,215,219,223
   ,226,228

48 GOSUB250

50 REM GET FIRST NOTE

52 PRINT''[clr][down, 3 times]
   SHOULD[sp]THE[sp]FIRST[sp]
   NOTE[sp, 3 times]BE[sp]SCA
   LE[sp]STEP[sp]#1?''

53 PRINT''[down]1.[sp]YES'':PR
   INT''[down]2.[sp]MAYBE'':PR
   INT''[down]3.[sp]NO[down, 3
    times]''

55 INPUTA$:IFA$<''1''ORA$>''3''
   THEN52

56 IFA$=''1''THENT%(1)=1

57 IF(A$=''3'')AND(T%(1)=1)THE
   NGOSUB250:GOTO56

90 PRINTE$

100 REM MAIN LOOP
```

```
102 TR=0:GOSUB230:GOSUB290:GOSU
    B220:GOSUB290

104 FORR=1TO5:GOSUB310:GOSUB300

106 AN$=''''':GOSUB200:TR=TR+1

108 IFVAL(AN$)<>T%(R)THENGOSUB2
    70:GOTO106

110 GOSUB 275:NEXT:GOSUB300:GOS
    UB290

120 REM FINISHED

122 GOSUB320

124 PRINTE$

126 PRINT''[down][sp, 3 times]*
    *[sp]REPORT[sp]CARD**

128 PRINT''[down, 2 times]YOUR[
    sp]TOOK[sp]''TR''TRIES''

130 PRINT''[sp, 2 times]TO[sp]G
    ET[sp]THE[sp]5[sp]NOTES.''

132 PRINT''[down, 2 times]FINAL
    [sp]SCORE:'';

134 SC=INT((5/TR)*100+.5):PRINT
    SC

136 PRINT''[down, 2 times]WANT[
    sp]ANOTHER?[sp](Y/N)

138 GETA$:IFA$=''Y''THENRUN30

140 IFA$=''N''THEN150

142 GOTO138

150 REM DONE

152 PRINT''[clr][down, 2 times]
    THANKS[sp]FOR[sp]PLAYING!''
    :CLR:END

200 REM INPUT ROUTINE

201 POKE198,0:PRINTD$SP$D$''WHI
    CH[sp]STEP[sp]?'' AN$

202 GETA$:IFA$=''''THEN202

203 IFA$=CHR$(13)THEN209

204 IFA$=''T''THENGOSUB220:AN$=
    '''':GOTO201

205 IFA$=''S''THENGOSUB230:AN$=
    '''':GOTO201

206 IFA$=CHR$(20)THENAN$=''''':G
    OTO201

207 IFA$<''1''ORA$>''8''THEN202

208 AN$=A$:GOTO201

209 IFAN$=''''THEN201

210 RETURN

220 REM PLAY TUNE

221 PRINTD$SP$D$''HERE[sp]IS[sp
    ]THE[sp]TUNE...''

222 FORN=1TO5

224 PI=P%(T%(N)):GOSUB260:NEXT

226 RETURN

230 REM PLAY SCALE

232 PRINTD$SP$D$''HERE[sp]IS[sp
    ]THE[sp]SCALE...''

234 FORN=1TO8:PI=P%(N):GOSUB260
    :NEXT

236 RETURN

250 REM GEN TUNE

251 FORN=1TO5:T%(N)=0:NEXT

252 FORN=1TO5

254 Z=INT(RND(1)*5)+1:IFT%(Z)<>
    0THEN254

256 T%(Z)=RND(1)*8+1:NEXT

258 RETURN

260 REM PLAY NOTE

262 POKES,PI

264 FORP=1TO500:NEXT

266 POKES,0:RETURN

270 REM WRONG ANS
```

```
272 PRINTD$SP$D$''NO,[sp]THAT[s
    p]IS[sp]NOT[sp]IT'':GOSUB29
    0:GOSUB220:RETURN

275 REM RIGHT ANS

276 FORQ=150TO240 STEP5:POKES,Q
    :NEXT:POKES,0

277 PRINTD$SP$D$''RIGHT!!!!'':G
    OSUB290:RETURN

290 FORP=1TO2E3:NEXT:RETURN

300 REM UPDATE SO FAR

302 SF$=SF$+''[sp]''+AN$

304 PRINT''[home][down, 4 times
    ]''SP$''[home][down, 4 time
    s]SO[sp]FAR:''SF$

306 RETURN

310 REM NOTE NUMBER

312 PRINT''[home][down, 2 times
    ]''SP$''[home][down, 2 time
    s]NOTE[sp]NUMBER''R

314 RETURN

320 REM FINISHED SOUND

322 FORN=1TO3:FORZ=250TO150STEP
    -4

324 POKES,Z:NEXTZ,N:POKES,0

326 RETURN
```

EARS–C-64 Version

```
 10 REM EARS (64)
 12 X=RND(-TI)
 20 PRINT''[clr]''SPC(217)''EAR
    S''SPC(110)''BY[sp]ANDY[sp]
    VAN[sp]DUYNE''
 21 FORP=1TO2E3:NEXT:PRINT''[do
    wn, 3 times]TOUCH[sp]A[sp]K
    EY...''
 22 GETA$:IFA$=''''THEN22
 30 REM SETUP STRINGS
 31 D$=''[home][down, 19 times]
    '':SP$=''[sp, 39 times]''
 35 E$=''[clr][rvs-on][sp, 17 t
    imes]EARS[sp, 19 times]''
 40 REM SETUP SOUND
 41 DIM T%(5),P%(8)
 42 S=54272:FORN=STO54295:POKEN
    ,0:NEXT:POKEN,15
 43 POKES+2,0:POKES+3,8:POKES+5
    ,15:POKES+6,255
 44 FORN=1TO8:READP%(N):NEXT
 46 DATA4817,5407,6069,6430,721
    7,8101,9094,9634
 48 GOSUB250
 50 REM GET FIRST NOTE
 52 PRINT''[clr][down, 3 times]
    SHOULD[sp]THE[sp]FIRST[sp]
    NOTE[sp]BE[sp]SCALE[sp]STE
    P[sp]#1?''
 53 PRINT''[down]1.[sp]YES'':PR
    INT''[down]2.[sp]MAYBE'':PR
    INT''[down]3.[sp]NO[down, 3
     times]''
 55 INPUTA$:IFA$<''1''ORA$>''3''
    THEN52
 56 IFA$=''1''THENT%(1)=1
 57 IF(A$=''3'')AND(T%(1)=1)THE
    NGOSUB250:GOTO56
 90 PRINTE$
100 REM MAIN LOOP
102 TR=0:GOSUB230:GOSUB290:GOSU
    B220:GOSUB290
104 FORR=1TO5:GOSUB310:GOSUB300
106 AN$='''':GOSUB200:TR=TR+1
108 IFVAL(AN$)<>T%(R)THENGOSUB2
    70:GOTO106
110 GOSUB 275:NEXT:GOSUB300:GOS
    UB290
120 REM FINISHED
122 GOSUB320
124 PRINTE$
126 PRINT''[down][sp, 11 times]
    **[sp]REPORT[sp]CARD**
128 PRINT''[down, 2 times]YOUR[
    sp]TOOK[sp]''TR''TRIES''
130 PRINT''[sp, 2 times]TO[sp]G
    ET[sp]THE[sp]5[sp]NOTES.''
132 PRINT''[down, 2 times]FINAL
    [sp]SCORE:'';
134 SC=INT((5/TR)*100+.5):PRINT
    SC
136 PRINT''[down, 2 times]WANT[
    sp]ANOTHER?[sp](Y/N)
138 GETA$:IFA$=''Y''THENRUN30
140 IFA$=''N''THEN150
142 GOTO138
150 REM DONE
152 PRINT''[clr][down, 2 times]
    THANKS[sp]FOR[sp]PLAYING!''
    :CLR:END
200 REM INPUT ROUTINE
201 POKE198,0:PRINTD$SP$D$''WHI
    CH[sp]STEP[sp]?'' AN$
```

```
202 GETA$:IFA$=''''THEN202

203 IFA$=CHR$(13)THEN209

204 IFA$=''T''THENGOSUB220:AN$=
    '''':GOTO201

205 IFA$=''S''THENGOSUB230:AN$=
    '''':GOTO201

206 IFA$=CHR$(20)THENAN$='''':G
    OTO201

207 IFA$<''1''ORA$>''8''THEN202

208 AN$=A$:GOTO201

209 IFAN$=''''THEN201

210 RETURN

220 REM PLAY TUNE

221 PRINTD$SP$D$''HERE[sp]IS[sp
    ]THE[sp]TUNE...''

222 FORN=1TO5

224 PI=P%(T%(N)):GOSUB260:NEXT

226 RETURN

230 REM PLAY SCALE

232 PRINTD$SP$D$''HERE[sp]IS[sp
    ]THE[sp]SCALE...''

234 FORN=1TO8:PI=P%(N):GOSUB260
    :NEXT

236 RETURN

250 REM GEN TUNE

251 FORN=1TO5:T%(N)=0:NEXT

252 FORN=1TO5

254 Z=INT(RND(1)*5)+1:IFT%(Z)<>
    0THEN254

256 T%(Z)=RND(1)*8+1:NEXT

258 RETURN

260 REM PLAY NOTE

261 P1=INT(PI/256):P2=PI-256*P1

262 POKES,P2:POKES+1,P1:POKES+4
    ,65

264 FORP=1TO500:NEXT

265 POKES,0:POKES+1,0:POKES+4,6
    4

266 RETURN

270 REM WRONG ANS

272 PRINTD$SP$D$''NO,[sp]THAT[s
    p]IS[sp]NOT[sp]IT'':GOSUB29
    0:GOSUB220:RETURN

275 REM RIGHT ANS

276 POKES+4,65:FORQ=30TO80STEP4
    :POKES,0:POKES+1,Q:NEXT

277 POKES+1,0:POKES+4,64

278 PRINTD$SP$D$''RIGHT!!!!'':G
    OSUB290:RETURN

290 FORP=1TO2E3:NEXT:RETURN

300 REM UPDATE SO FAR

302 SF$=SF$+''[sp]''+AN$

304 PRINT''[home][down, 4 times
    ]''SP$''[home][down, 4 time
    s]CORRECT[sp]SO[sp]FAR:''SF
    $

306 RETURN

310 REM NOTE NUMBER

312 PRINT''[home][down, 2 times
    ]''SP$''[home][down, 2 time
    s]NOTE[sp]NUMBER''R

314 RETURN

320 REM FINISHED SOUND

322 FORN=1TO3:POKES+4,65:FORZ=1
    00TO50STEP-2

324 POKES+1,Z:NEXTZ,N:POKES+1,0
    :POKES+4,64

326 RETURN
```

Explorations with Assembly Langauge

DIS/MON

By Eric Giguere

Welcome to the first installment of my new column. Those of you who have been following my previous series ("An Introduction to Assembly Language") should now be ready to start working with more advanced concepts. "Explorations With Assembly Language" assumes you have an elementary knowledge of 6502/6510 assembly language. For those of you who are interested in assembly language but do not know where to start, I suggest you buy a book on the subject or order back issues of *COMMANDER*.

WITH OR INTO?

I had at first considered naming this column "Explorations *Into* Assembly Language" but then decided to replace the "Into" with "With." Why? Because we are not only going to learn more about programming in assembly language, but we are also going to explore the insides of the computer. We are going to look at how assembly language is used to make the computer "tick"–from the BASIC language to the Input/Output (I/O) routines, all from the assembly language point of view.

THE TOOL OF THE TRADE

Before we can start exploring, we need some kind of tool to aid in our explorations. Most of the things we will do require the use of a machine-language monitor and a disassembler (I am assuming you know what these are by now). Some of you may have at least one of these already, as there are several good ones available. In case you do not, I am including here the program DIS/MON, a combination disassembler/monitor (the proper term is really "extended monitor"). Written in BASIC, and hence a trifle slow, it works both on the VIC-20 and the C-64.

Typing in DIS/MON should be fairly straightforward. You can omit all the REM statements as no GOTOs or GOSUBs are made to them. VIC owners will have to plug in their 8K (or larger) RAM expansion cartridge (you do have one, do you not?) and fix-up the PRINT statements at the beginning of the program so that they fit on the 22-column screen. And if you do not feel like typing in the program, you can obtain a copy of it through COMMANDER's "Tapes on Command" offered elsewhere in this issue.

USING DIS/MON

Once you have typed DIS/MON in, enter the magic word RUN and press [RETURN]. On the bottom left-hand side of the title screen, you will see a bracket and a flashing cursor. DIS/MON is now waiting for you to enter one of twelve commands: A, C, D, G, L, M, P, R, S, X, : or Z. To enter a command, type the command letter, a space, and the parameters needed (explained below). Press [RETURN] and DIS/MON will execute the command (providing everything is in order). If you make a typing mistake, you can use the INST/DEL key to delete the preceding character(s) and type the correct sequence. Following are the descriptions and instructions for each command.

(A)SCII

The command A is used to display the ASCII values of specified memory locations. The command A must be followed by two four-digit hexadecimal numbers (all numbers in DIS/MON are in hex), each separated by a blank space. The first digit is the starting address and the second is the ending address. Upon pressing [RETURN], DIS/MON will display all the ASCII values of the bytes from the starting to the ending addresses. For example:

```
]A 033C 033F
```

will display the ASCII values of the bytes $033C-$033F. The bytes are displayed 16 at a time, in reverse text, preceded by a four-digit hex number showing the first byte in the line. Please note that the reversed quote (") character you see at the start of each line is not an actual ASCII value–it is there only to make sure certain ASCII characters will show up on the screen. If you select the P ON option, the ASCII values will be sent to the printer instead of the screen.

Why display ASCII values? A lot of times messages are stored into memory in ASCII form. This command can be used to show us what these messages are without having to resort to cumbersome ASCII conversion charts. For an interesting peek at BASIC type the following on the VIC:

```
]A C000 C100
```

or on the C-64:

```
]A A000 A100
```

You will only see garbage at first, but then it gets more interesting. (Can you figure out what they are? Try switching character sets by pressing the shift and Commodore keys.) When using the ASCII command, holding down a key freezes the listing, while pressing the space bar aborts it completely and returns you to the command mode.

(C)atalog

This is only for those users with disk drives. Pressing C and [RETURN] gives you a catalog or directory–the list of all the programs on the current disk in the drive. Pressing a key will freeze the catalog and pressing the space bar aborts it completely.

(D)isassemble

This is a powerful command: it disassembles assembly language code. What this means is that arcane hex numbers are converted into the more easily understood assembly language instructions that we know and love, such as LDA #$F4 and JMP ($033C). To use it, simply type the starting and ending addresses of the memory locations you wish disassembled, as in:

]D FFD2 FFFF

which will proceed to disassemble all the memory from $FFD2 to $FFFF. At any time during the disassembly, you may freeze the listing by holding a key or abort it by pressing the space bar. If the printer option is on, the listing will go to the printer.

If a byte to be disassembled is a non-existent assembly language instruction it will be displayed on screen as three asterisks–***. This means you have wandered somewhere where there are no instructions. You will see a few occasionally, but these will mostly be random combinations. We will talk some more about this in a later column.

(G)oto

The G command is very simple: it allows you to start executing a machine language program in memory. Simply type G and the starting address. Example:

]G FFD2

would execute the program starting at $FFD2.

(L)oad

L is used to load a machine-language program into memory. Type L, a space, the filename of the program in between apostrophes, and the device number from which you want to load the program. Example:

]L 'EXAMPLE PROG',Ø8

would load EXAMPLE PROG from device 8 (the disk drive). Tape users would use

]L 'EXAMPLE PROG',Ø1

If you do not specify a device number it will be assumed to be the disk, 8.

Notice that the filename is to be enclosed within apostrophes ('). Usually we use quotes (") but these can sometimes affect the screen in funny ways. Rather than go through the hassle of dealing with them, DIS/MON automatically changes SHIFT-2 (") to SHIFT-7 (') each time you press that particular combination. This means that you can type out L "EXAMPLE PROG" and on the screen it will show as L 'EXAMPLE PROG'.

(M)emory Display

The M command is just like the A command except that it displays the true hex values of the bytes and not their ASCII representations. The command

]M Ø33C Ø33F

will display the values of the four bytes from $Ø33C-$Ø33F. As in the ASCII command, hitting a key freezes the display while pressing the space bar aborts it. If the printer option is chosen output is sent to the printer instead of the screen.

(P)rinter ON/OFF

This command is used to direct output to the screen or printer. Type P, a space, and the word 'ON' or 'OFF'. 'ON' turns on the printer–i.e., most commands will send their output to the printer. 'OFF' turns off the printer and directs output to the screen. When DIS/MON is first run, the printer is in the OFF mode. This command is useful for getting printed copies for later study.

(R)egister Display

The R command displays the current values of the three important registers: the accumulator (AC), the X-register (XR) and the Y-register (YR).

(S)ave

S is the opposite of L: it stores machine-language programs in memory. Use the same syntax as L but follow the device number by the starting and ending addresses of the program, separated by a comma. Example:

]S 'EXAMPLE PROG',Ø8,Ø4ØØ,Ø8ØØ

Always add 1 to the actual ending address so that the last byte will be saved. Once again, cassette users may replace Ø8 with Ø1 for the device number.

e(X)it Program

X is used to quit the program and exit to BASIC. Simply type X and press [RETURN]. If you wish to return to DIS/MON intact, type GOTO 2ØØ.

(:)-Change memory

The : (colon) command is used to store new values into memory. Type :, the starting address, a blank space, and the two digit values to be stored. Example:

]:Ø33C 6Ø ØØ FF FF ØØ

would store the five bytes $6Ø,$ØØ, $FF,$FF and $ØØ into memory starting at $Ø33C. If you then did a M Ø33C Ø34Ø you would see these bytes displayed.

There is only one limitation to this command: you cannot store bytes within the DIS/MON program space itself. This is so that you do not crash the computer while playing around.

(Z) send disk command

The Z command has no relation to its purpose, but that is OK. Follow Z by the disk command enclosed in apostrophes and it will automatically be transmitted to the disk drive. Example:

]Z 'S:EXAMPLE PROG'

would send S:EXAMPLE PROG to the drive. If you type 'E' as the disk command DIS/MON will instead read and display the error channel message. Example:

]Z 'E'

HOW IT WORKS

DIS/MON is basically a simple program. If you have typed in my EDIT/ASM and BASICMON programs from previous issues you will see many similarities. In fact, if you have a BASIC extension utility you could load in the EDIT/ASM assembler module and delete all the lines up to 8999. This would leave you the DATA statements from 9ØØØ on and save you quite a bit of typing (they are exactly the same in both EDIT/ASM and DIS/MON). Once DIS/MON recognizes a command it goes to the appropriate routine and returns when finished. The important program sections are

1Ø - 19Ø	Initialization
2ØØ - 24Ø	Fetch command
5ØØ - 56Ø	A routine
1ØØØ - 113Ø	C "
15ØØ - 177Ø	D "
2ØØØ - 2Ø1Ø	- G "
25ØØ - 254Ø	L "
3ØØØ - 3Ø6Ø	M "
35ØØ - 352Ø	P "
4ØØØ - 4Ø2Ø	R "
45ØØ - 457Ø	S "
5ØØØ	X "
55ØØ - 553Ø	: "
6ØØØ - 6Ø2Ø	Z "
7ØØØ - 7Ø1Ø	Error messages
75ØØ - 751Ø	Get disk status
76ØØ - 762Ø	Print disk status
77ØØ - 777Ø	Get filename and device #
8ØØØ - 8Ø8Ø	Input routine
8Ø9Ø - 811Ø	Get two hex values
85ØØ - 852Ø	Hex to decimal converter
853Ø - 858Ø	Decimal to hex converter
9ØØØ - 999Ø	Data for disassembly

You should have fun analyzing this program!

NEXT MONTH

Once you've typed in the program you'll be all set for future columns. Next month we're going to start exploring that useful area of memory called the Kernal. In the meantime have fun trying out your new program.

Any questions or comments should be sent to: ERIC GIGUERE c/o *COMMANDER* Magazine, P. O. Box 98827, Tacoma, WA 98498. Please allow several weeks for a reply, as they must be relayed to me in Canada

Explorations With Assembly Language

DIS/MON

```
10 REM     ************
20 REM     *          *
30 REM     * DIS/MON *
40 REM     *          *
50 REM     ************
60 REM
70 REM COPYRIGHT (C)1983 BY
80 REM     ERIC GIGUERE
90 REM
100 IF PEEK(806)=202 THEN POKE
    53280,6: POKE 53281,1: GOTO
     120
110 POKE 36879,30
115 REM VIC OWNERS WILL HAVE TO
     ADJUST THE FOLLOWING PRINT
     STATEMENTS TO FIT
117 REM THE 22-COLUMN SCREEN. N
    O BIG DEAL.
120 PRINT''[clr][rvs-on][9rn][s
    p, 40 times]'';
130 PRINT''[rvs-on][sp, 17 time
    s]DIS/MON[sp, 16 times]'';
140 PRINT''[rvs-on][sp, 40 time
    s]''
150 PRINT''[down][blu][sp, 3 ti
    mes]COPYRIGHT[sp](C)1983[sp
    ]BY[sp]ERIC[sp]GIGUERE''
160 PRINT''[down, 10 times]'':
    OPEN 4,3
165 :
170 CS$=''ACDGLMPRSX:Z'':HN$=''
    0123456789ABCDEF'':NU$=CHR$
    (0):QU$=CHR$(34)
180 R$=CHR$(13)
190 LM=PEEK(44)*256+PEEK(43)-2:
     HM=PEEK(46)*256+PEEK(45)+1
200 PRINT ''[down][blu]]'';: GO
    SUB 8000: COM$=LEFT$(A$,1)
210 Z=0:FORY=1TO12:IFCOM$=MID$(
    CS$,Y,1)THENZ=Y:Y=1E9
230 NEXT:IFZ=0THENPRINT''[down]
    [9rn]COMMANDS:[sp]A,C,D,G,
    L,M,P,R,S,X,:[sp]OR[sp]Z'':
    GOTO200
240 ON Z GOTO 500,1000,1500,200
    0,2500,3000,3500,4000,4500,
    5000,5500,6000
497 REM
498 REM (A)SCII
499 REM
500 IF A>65535 THEN 7000
510 IF B<A OR B>65535 THEN 7010
520 ZZ=-1: FOR LOOP=A TO B: ZZ=
    ZZ+1
530 IFZZ/16=INT(ZZ/16)THEN DV=L
    O:M=0:GOSUB 8550: PRINT#4:
    PRINT#4,HX$'':[rvs-on]'';QU
    $;
540 DV=PEEK(LOOP): PRINT#4,CHR$
    (DV);
550 WAIT 197,64: GET KEY$: IF K
    EY$=''[sp]'' THEN LOOP=1E9
```

```
560 NEXT LOOP: PRINT#4: GOTO 20
    0

997 REM

998 REM (C)ATALOG

999 REM

1000 PRINT#4 : OPEN 15,8,15,''I''
     : GOSUB 7500: IF E THEN 760
     0

1010 OPEN 1,8,0,''$0'': GOSUB 75
     00: IF E THEN 7600

1020 GET#1,A$,A$

1030 GET#1,A$,A$

1040 IF A$='''' THEN 1130

1050 GET#1,A$,B$

1060 PRINT#4,ASC(A$+NU$)+ASC(B$+
     NU$)*256;

1070 GET#1,A$

1080 IF A$='''' THEN PRINT#4: GO
     TO 1030

1090 PRINT#4,A$;

1100 GET KEY$: IF KEY$=''[sp]''
     THEN 1130

1110 WAIT 197,64

1120 GOTO 1070

1130 PRINT#4: CLOSE 1: CLOSE 15:
      GOTO 200

1497 REM

1498 REM (D)ISASSEMBLE

1499 REM

1500 IF A>65535 THEN 7000

1510 IF B<A OR B>65535 THEN 7010

1520 PRINT#4: PC=A

1530 M=0: DV=PC: GOSUB 8550: PRI
     NT#4,HX$;''[sp]'';: BY=PEEK
     (PC): RESTORE
```

```
1540 READ OP$: IF OP$=''END'' TH
     EN 1770

1550 READ VA$: IF VA$=''-1'' THE
     N 1540

1560 OP=VAL(MID$(VA$,2)): IF BY<
     >OP THEN 1550

1570 PRINT#4,OP$''[sp]'';: AD=VA
     L(LEFT$(VA$,1))

1580 LO=PEEK(PC+1): HI=PEEK(PC+2
     )

1585 IF LEFT$(OP$,1)<>''B'' OR O
     P$=''BIT'' OR OP$=''BRK'' T
     HEN 1610

1590 IF LO>127 THEN LO=-256+LO

1600 PC=PC+2: DV=PC+LO: M=0: GOS
     UB 8550: PRINT#4,''$'';HX$:
      GOTO 1740

1610 DV=LO: GOSUB 8540: LO$=HX$:
      DV=HI: GOSUB 8540: HI$=HX$

1620 ON AD+1 GOTO 1630,1640,1650
     ,1670,1680,1690,1700,1710,1
     720,1730

1630 PRINT#4,''#$'';LO$: PC=PC+2
     : GOTO 1740

1640 PRINT#4,''$'';LO$: PC=PC+2:
      GOTO 1740

1650 IF OP$=''LDX'' OR OP$=''STX
     '' THEN PRINT#4,''$'';LO$;''
     ,Y'': PC=PC+2: GOTO 1740

1660 PRINT#4,''$'';LO$;'',X'': P
     C=PC+2: GOTO 1740

1670 PRINT#4,''($'';LO$;'',X)'':
      PC=PC+2: GOTO 1740

1680 PRINT#4,''($'';LO$;''),Y'':
      PC=PC+2: GOTO 1740

1690 PRINT#4,''$'';HI$;LO$: PC=P
     C+3: GOTO 1740

1700 PRINT#4,''($'';HI$;LO$;'')''
     : PC=PC+3: GOTO 1740

1710 PRINT#4,''$'';HI$;LO$;'',X''
     : PC=PC+3: GOTO 1740

1720 PRINT#4,''$'';HI$;LO$;'',Y''
     : PC=PC+3: GOTO 1740

1730 PRINT#4: PC=PC+1

1740 WAIT 197,64: GET KEY$: IF K
     EY$=''[sp]'' THEN PC=1E9

1750 IF PC>B THEN PRINT#4: GOTO
     200

1760 GOTO 1530

1770 PRINT#4,''###'': PC=PC+1: G
     OTO 1740

1997 REM

1998 REM (G)OTO

1999 REM

2000 IF A>65535 THEN 7000

2010 SYS (A): GOTO 200

2497 REM

2498 REM (L)OAD

2499 REM

2500 GOSUB 7700: IF E THEN PRINT
      ''[down][red]NO[sp]FILENAM
     E'': GOTO 200

2510 P=512: FOR ZZ=1 TO LEN(N$):
      ZY=ASC(MID$(N$,ZZ,1)): POK
     E P,ZY: P=P+1

2520 NEXT: POKE 183,P-512: POKE
     187,0: POKE 188,2: POKE 185
     ,1: POKE 184,127

2530 POKE P,169: POKE P+1,0: POK
     E P+2,32: POKE P+3,213: POK
     E P+4,255

2540 POKE P+5,96: POKE 186,DE: P
     OKE 157,128: PRINT ''[up]''
     ;: SYS P: PRINT: GOTO 200
2997 REM

2998 REM (M)EMORY

2999 REM

3000 IF A>65535 THEN 7000

3010 IF B<A OR B>65535 THEN 7010
```

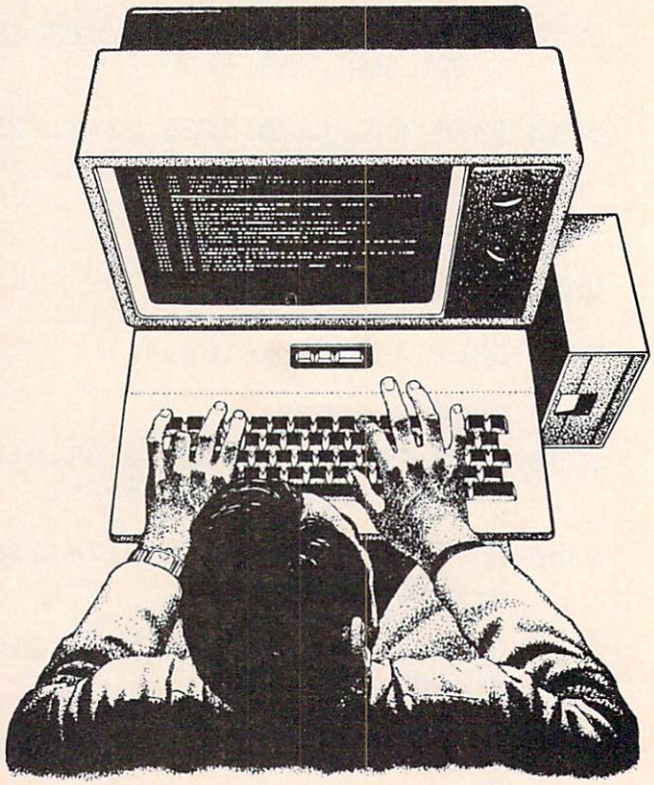
```
3020 ZZ=-1: FOR LOOP=A TO B: ZZ=
     ZZ+1

3030 IF ZZ/8=INT(ZZ/8) THEN DV=L
     OOP: M=0: GOSUB 8550: PRINT
     #4: PRINT#4,HX$''':'';

3040 DV=PEEK(LOOP): GOSUB 8540:
     PRINT#4,HX$''[sp]'';

3050 WAIT 197,64: GET KEY$: IF K
     EY$=''[sp]'' THEN LOOP=1E9

3060 NEXT LOOP: PRINT#4: GOTO 20
     0

3497 REM

3498 REM (P)RINTER ON/OFF

3499 REM

3500 IF MID$(IN$,4,1)=''N'' THEN
      CLOSE4: OPEN 4,4: GOTO 200

3510 IF MID$(IN$,4,1)=''F'' THEN
      CLOSE4: OPEN 4,3: GOTO 200

3520 GOTO 200

3997 REM

3998 REM (R)EGISTER DISPLAY

3999 REM

4000 PRINT#4,R$''[sp, 2 times]AC
     [sp, 2 times]XR[sp, 2 times
     ]YR''

4010 FOR LOOP=780 TO 782: DV=PEE
     K(LOOP): GOSUB 8540: PRINT#
     4,''[sp, 2 times]''HX$;: NE
     XT

4020 PRINT#4,R$: GOTO 200

4497 REM

4498 REM (S)AVE MEMORY

4499 REM

4500 GOSUB 7700: IF E THEN PRINT
      ''[down][red]NO[sp]FILENAM
     E'': GOTO 200

4510 P=512: FOR ZZ=1 TO LEN(N$):
      ZY=ASC(MID$(N$,ZZ,1)): POK
     E P,ZY: P=P+1
```

```
4520 NEXT: POKE 183,P-512: POKE
     187,0: POKE 188,2: POKE 185
     ,0: POKE 184,127

4530 POKE 186,DE: POKE 157,128:
     A%=A/256: POKE 194,A%

4540 POKE 193,A-A%*256: POKE P,1
     69: POKE P+1,193: POKE P+2,
     162

4550 POKE P+4,160: POKE P+6,32:
     POKE P+7,216: POKE P+8,255:
      POKE P+9,96

4560 B%=B/256: POKE P+3,B-B%*256

4570 POKE P+5,B%: PRINT''[up]'';
     : SYS P: PRINT: GOTO 200

4997 REM

4998 REM E(X)IT COMMAND

4999 REM

5000 PRINT''[down][blk]TO[sp]RE-
     ENTER[sp]PROGRAM[sp]TYPE[sp
     ]'GOTO[sp]200'.'':END

5497 REM

5498 REM (:) MEMORY CHANGE COMMA
     ND

5499 REM

5500 IF LEN(IN$)<8 THEN PRINT ''
     [down][red]NO[sp]DATA'':GOT
     O200

5510 V$=MID$(IN$,2,4): GOSUB 850
     0: IF DV<HM AND DV>LM THEN
     200

5520 P=DV: FOR ZZ=7 TO LEN(IN$)
     STEP 3: V$=MID$(IN$,ZZ,2):
     GOSUB 8500

5530 POKE P,DV: P=P+1: NEXT: GOT
     O 200

5997 REM

5998 REM (Z) SEND DISK COMMAND

5999 REM

6000 GOSUB 7700: IF E THEN PRINT
      ''[down][red]NO[sp]COMMAND
     [sp]TO[sp]BE[sp]SENT'': GOT
     O 200

6010 OPEN 15,8,15: IF LEFT$(N$,1
     )=''E'' THEN PRINT: GOSUB 7
     500: GOTO 7600

6020 PRINT#15, N$: CLOSE 15: GOT
     O 200

7000 PRINT ''[down][9rn]FIRST[sp
     ]VALUE[sp]OUT[sp]OF[sp]RANG
     E'': GOTO 200

7010 PRINT ''[down][9rn]SECOND[s
     p]VALUE[sp]OUT[sp]OF[sp]RAN
     GE'': GOTO 200

7497 REM

7498 REM GET DISK STATUS

7499 REM

7500 INPUT#15,E,E$,T,S: IF E<20
     THEN E=0

7510 RETURN

7597 REM
7598 REM PRINT DISK ERROR MESSAG
     E

7599 REM

7600 PRINT ''[blk]DISK[sp]ERROR:
     [down][red]''

7610 PRINT E;''-[sp]'';E$;'':'';
     T;'','';S

7620 CLOSE 1: CLOSE 15: GOTO 200

7697 REM

7698 REM GET FILENAME & DEVICE

7699 REM

7700 E=0: N$='''': QU=0: FOR LOO
     P=1 TO LEN(IN$): ZZ$=MID$(I
     N$,LOOP,1)

7710 IF ZZ$=CHR$(39) THEN QU=1-Q
     U: IF QU=0 THEN EN=LOOP: LO
     OP=1E9: GOTO 7730
```

```
7720 IF QU THEN N$=N$+ZZ$

7730 NEXT LOOP: N$=MID$(N$,2): I
     F LEN(N$)<1 THEN E=1

7740 DE$=MID$(IN$,EN+2): ZZ=VAL(
     DE$): IF ZZ<8 AND ZZ<>1 THE
     N ZZ=8

7750 DE=ZZ: FOR LOOP=1 TO LEN(DE
     $)

7760 IF MID$(DE$,LOOP,1)=''," T
     HEN IN$=''[sp, 2 times]''+M
     ID$(DE$,LOOP+1): LOOP=1E9

7770 NEXT LOOP: GOSUB 8100: RETU
     RN

7997 REM

7998 REM GENERAL PURPOSE INPUT R
     OUTINE

7999 REM

8000 IN$='''':PRINT''[sp][left]''


8010 P=PEEK(209)+PEEK(210)*256+P
     OS(0):CH=PEEK(P):RC=128:TT=
     0

8020 POKEP,CH+RC:TT=TT+1:IFTT>10
     THENTT=0:RC=128-RC:GOTO8020

8030 GETX$:IFX$=''''THEN8020

8040 X=LEN(IN$):IFX$=CHR$(34)THE
     NX$=CHR$(39):GOTO8080

8050 IFX$=CHR$(20)ANDX>0THENIN$=
     LEFT$(IN$,X-1):POKEP,CH:PRI
     NT''[left][sp][left]'';:GOT
     O8010

8060 IFX$=CHR$(13)ANDX>0THENPOKE
     P,CH:PRINT:GOTO8090

8070 IFASC(X$)<32ORASC(X$)>90THE
     N8020

8080 IN$=IN$+X$:POKEP,CH:PRINTX$
     ''[sp][left]'';:GOTO8010

8090 A$=LEFT$(IN$,1)

8100 V$=MID$(IN$,3,4): GOSUB 850
     0: A=DV: V$=MID$(IN$,8,4):
     GOSUB 8500: B=DV

8110 RETURN

8497 REM

8498 REM HEX TO DECIMAL

8499 REM

8500 DV=0: FOR HX=1 TO LEN(V$):
     DV%=ASC(V$+NU$): DV%=DV%-48
     +(DV%>64)*7

8510 V$=MID$(V$,2): DV=16*DV+DV%
     : NEXT: IF DV<0 THEN DV=0

8520 RETURN

8524 REM

8525 REM DECIMAL TO HEX

8526 REM

8530 DV=VAL(V$): REM ENTRY POINT
      1

8540 M=0: IF DV<256 THEN M=2: RE
     M ENTRY POINT 2

8550 IF DV<0 THEN DV=0: REM ENTR
     Y POINT 3

8560 N=DV: N%(1)=N/4096: N=N-N%(
     1)*4096: N%(2)=N/256: N=N-N
     %(2)*256: N%(3)=N/16

8570 N%(4)=N-N%(3)*16:HX$='''':
     FOR HX=1+M TO 4: HX$=HX$+MI
     D$(HN$,N%(HX)+1,1)

8580 NEXT: RETURN

8999 END

9000 DATA ADC,0105,1101,2117,397
     ,4113,5109,7125,8121,-1

9010 DATA AND,041,137,253,333,44
     9,545,761,857,-1

9020 DATA ASL,16,910,514,222,730
     ,-1

9030 DATA BCC,9144,-1

9040 DATA BCS,9176,-1

9050 DATA BEQ,9240,-1

9060 DATA BIT,136,544,-1
```

```
9070 DATA BMI,948,-1

9080 DATA BNE,9208,-1

9090 DATA BPL,916,-1

9100 DATA BRK,90,-1

9110 DATA BVC,980,-1

9120 DATA BVS,9112,-1

9130 DATA CLC,924,-1
9140 DATA CLD,9216,-1

9150 DATA CLI,988,-1

9160 DATA CLV,9184,-1

9170 DATA CMP,3193,1197,0201,520
     5,4209,2213,8217,7221,-1

9180 DATA CPX,0224,1228,5236,-1

9190 DATA CPY,0192,1196,5204,-1

9200 DATA DEC,1198,5206,2214,722
     2,-1

9210 DATA DEX,9202,-1

9220 DATA DEY,9136,-1

9230 DATA EOR,365,169,073,577,48
     1,285,889,793,-1

9240 DATA INC,1230,5238,2246,725
     4,-1

9250 DATA INX,9232,-1

9260 DATA INY,9200,-1

9270 DATA JMP,576,6108,-1

9280 DATA JSR,532,-1
9290 DATA LDA,3161,1165,0169,517
     3,4177,2181,8185,7189,-1

9300 DATA LDX,0162,1166,5174,218
     2,8190,-1

9310 DATA LDY,0160,1164,5172,218
     0,7188,-1

9320 DATA LSR,170,974,578,286,79
     4,-1

9330 DATA NOP,9234,-1
```

```
9340 DATA ORA,31,15,09,513,417,2
     21,825,729,-1

9350 DATA PHA,972,-1

9360 DATA PHP,98,-1

9370 DATA PLA,9104,-1

9380 DATA PLP,940,-1

9390 DATA ROL,138,942,546,254,76
     2,-1

9400 DATA ROR,1102,9106,5110,211
     8,7126,-1

9410 DATA RTI,964,-1

9420 DATA RTS,996,-1

9430 DATA SBC,3225,1229,0233,523
     7,4241,2245,8249,7253,-1

9440 DATA SEC,956,-1

9450 DATA SED,9248,-1

9460 DATA SEI,9120,-1

9470 DATA STA,3129,1133,5141,414
     5,2149,8153,7157,-1

9480 DATA STX,1134,5142,2150,-1

9490 DATA STY,1132,5140,2148,-1

9500 DATA TAX,9170,-1

9510 DATA TAY,9168,-1

9520 DATA TSX,9186,-1

9530 DATA TXA,9138,-1

9540 DATA TXS,9154,-1

9550 DATA TYA,9152,-1

9990 DATA END

9995 :

9996 :

9997 :

9998 *** END OF PROGRAM ***
```

RESCUE!

FOR THE 5K VIC-20
(JOYSTICK REQUIRED)

By Mike Scharland, age 16

In this game, you are an angel of mercy who must save several trapped people on ledges above a sea of sulfuric acid. Use the joystick to move left or right, and press the red fire button to go up. Otherwise gravity will pull you down when you are not sitting on a ledge. Don't touch the skull creature or hit your head on the underside of a ledge.

Your angel has 3 "lives", and a warning tone sounds when time is running out. If you choose a higher difficulty level, your angel will have less time to work.

Because of the VIC's limited meaning, the program must be loaded in two parts. First type in Listing 1, and SAVE it as "RESCUE1". Then type in Listing 2 and SAVE it as "RESCUE2". When you are ready to play, LOAD and RUN "RESCUE1", then LOAD and RUN "RESCUE2". Be sure your memory expansion cartridges are removed.

Hope you enjoy the challenge, as well as the graphics and sound.

Listing 1-"RESCUE1"

```
 5 POKE52,28:POKE56,28:POKE51,P
   EEK(55):CLR

10 CS=256*PEEK(52)+PEEK(51):FO
   RI=CSTOCS+511:POKEI,PEEK(I+
   32768-CS):NEXT

20 FORI=7168TO7215:READJ:POKEI
   ,J:NEXT

21 DATA0,0,0,0,0,108,255,255,8
   0,255,255,4,0,0,0,0

26 DATA36,60,126,255,255,126,6
   0,0,60,60,24,255,24,24,36,1
   02

28 DATA129,219,219,255,60,24,3
   6,102,56,46,127,255,248,255
   ,127,0

100 CLR
```

Listing 2-"RESCUE2"

```
 0 POKE36869,255:SC=0:LIVES=3:G
   OTO10000

 1 RESTORE:PRINT"[clr]":FORE
   E=8164TO8185:POKEEE+30720,5
   :POKEEE,0:NEXT

 2 FORQ=1TO5:READA:READB:FOREE
   =ATOB:POKEEE+30720,2:POKEEE
   ,1:NEXT:NEXT

 3 DATA7806,7811,7883,7887,796
   6,7970,7960,7964,8042,8047

 4 FORQ=1TO5:READA:FOREE=1TO2:
   B=A+INT(RND(1)*4)

 5 IFPEEK(B)=3THENB=B+INT(RND(
   1)*3)-1:GOTO5

 6 POKEB+30720,7:POKEB,3:NEXTE
   E:NEXTQ:POKE36875,0:POKE368
   74,0

 7 DATA7785,7861,7945,7938,802
   0
```

```
  8 X=7725:FORU7=7680TO7701:POK
    EU7+30720,2:POKEU7,1:NEXT

  9 PRINT''[home][blu][down, 7
    times]BBE'':PRINT''[home][d
    own, 10 times]BBE'':PRINT''
    [home][down, 14 times]BBE''

 10 E(1)=7837:E(2)=7903:E(3)=79
    91

 18 TI$=''000000''

 19 IFTI>1300THENPOKE36878,3:PO
    KE36874,200:POKE36875,200:I
    FTI>2000-175*LEVTHEN4999

 20 POKE37154,127:P=PEEK(37152)
    AND128:J0=-(P=0):POKE37154,
    255:P=PEEK(37151)

 21 J2=-((PAND16)=0):FB=-((PAND
    32)=0):IFFBTHEN300

 22 IFPEEK(X+22)<>1THENX=X+22:P
    OKEX-22,32:IFPEEK(X)<>32THE
    N1000

 23 IFPEEK(X+(J0-J2))=3THENJ0=0
    :J2=0

 26 POKEX,32:X=X+(J0-J2):POKEX+
    30720,4:POKEX,4

 40 FORD=1TO3:POKEE(D)-4,32:POK
    EE(D)+30719,6:POKEE(D)-1,2:
    E(D)=E(D)+1

 41 POKEE(D)+30719,1:POKEE(D)-1
    ,5

 42 IFE(3)>8009THEN50

 43 NEXT:GOTO19

 50 FORT=1TO3:FOREE=E(T)-4TOE(T
    ):POKEEE,32:NEXTEE:NEXTT

 51 FORY=1TO3:E(Y)=E(Y)-21:NEXT
    Y

 80 GOTO19

300 POKE36878,13:POKE36877,137:
    POKEX,32:X=X+(J0-J2)-22:POK
    E36877,0:IFPEEK(X)<>32THEN5
    000
```

```
 301 POKEX+30720,4:POKEX,4:GOTO4
     0

1000 IFPEEK(X)=3THEN2000

1001 IFPEEK(X)=0THEN1100

1010 POKE36878,15:FORBN=150TO128
     STEP-1:POKEX+30720,INT(RND(
     1)*8)+1:POKE36874,BN:NEXT

1011 POKE36874,0:POKE36879,6+2*L
     IVES:LIVES=LIVES-1:IFLIVES=
     0THEN4000

1012 X=7725:FORL1=1TO400:NEXT:GO
     TO18

1100 POKE36878,0:POKE36877,222:F
     ORP2=4TO0STEP-.05:POKE36878
     ,P2:NEXT:POKE36877,0:GOTO10
     11

2000 SC=SC+20:PRINT''[home][down
     ][cyn]POINTS:[yel]'';SC

2001 POKE36877,0:POKE36878,13:FO
     RL2=1TO10:POKE36876,INT(RND
     (1)*128)+128

2002 FORK=0TO10:NEXTK:NEXTL2:POK
     E36876,0:POKE36878,0

2003 IFSC=TS+200THENTS=SC:GOTO1

2004 GOTO19

4000 POKE36869,240:PRINT''[down,
      2 times][clr][grn]FINAL[sp
     ]SCORE:[yel]'';SC

4001 CLR:INPUT''[down][pur]ANOTH
     ER[sp]GAME(Y/N)'';A$

4002 IFA$=''Y''THENRUN

4003 END

4999 POKEX,32

5000 R6=0:FORE=X+44TO8185STEP22:
     R6=R6+1:POKEE-22,32:IFPEEK(
     E)=3THENTS=TS-20

5001 POKEE+30720,4:POKEE,4:POKE3
     6878,15:POKE36875,260-R6*8

5002 NEXT:POKEE+30698,5:POKEE-22
     ,0:POKE36875,0:POKE36874,0:
     GOTO1100

10000 POKE36879,8:POKE36869,240:P
      RINT''[clr][blu][down, 2 ti
      mes][sp, 3 times]INPUT[sp]S
      KILL[sp]LEVEL''

10001 PRINT''[down, 2 times][yel]
      [rvs-on]1[rvs-off][sp][grn
      ]SIMPLE''

10002 PRINT''[down, 2 times][yel]
      [rvs-on]2[rvs-off][sp][grn
      ]HARDER''

10003 PRINT''[down, 2 times][yel]
      [rvs-on]3[rvs-off][sp][grn
      ]YOU[sp]GOTTA[sp]BE[sp]FAST
      '':PRINT''[right, 2 times]F
      OR[sp]THIS[sp]ONE!''

10004 PRINT''[down, 2 times][rvs-
      on][yel]4[rvs-off][sp][red]
      INSTRUCTIONS''

10005 PRINT''[down, 2 times][cyn]
      '':INPUTLEV:IFLEV>4ORLEV<1
      THEN10000
```

```
10006 IFLEV<>4THENPOKE36869,255:P
      OKE36879,14:GOTO1

10050 PRINT''[clr]YOU[sp]ARE[sp]A
      N[sp]ANGEL[sp]WITH[sp]THE[s
      p]MISSION[sp]OF[sp]SAVING[s
      p]SEVERAL[sp]TRAPPED[sp]PEO
      PLE'';

10051 PRINT''THAT[sp]ARE[sp]STUCK
      [sp]ON[sp, 5 times]LEDGES[s
      p]ABOVE[sp]A[sp]SEA[sp]OF[s
      p]SULFURIC[sp]ACID.''

10052 PRINT''YOU[sp]MUST[sp]FLY[s
      p]ON[sp]TOP[sp, 3 times]OF[
      sp]THEM[sp]TO[sp]SET[sp]THE
      M[sp, 3 times]FREE[sp]USING
      [sp]THE[sp]RED''

10053 PRINT''BUTTON[sp]ON[sp]THE[
      sp]JOYSTICKTO[sp]FLY[sp]AND
      [sp]THE[sp]LEFT[sp]&[sp]RIG
      HT[sp]CONTROLS[sp]TO''

10060 PRINT''MANUEVER[sp]THE[sp]A
      NGEL.[sp, 3 times]YOU[sp]MU
      ST[sp]BEWARE[sp]OF[sp]THESK
      ULL[sp]CREATURES[sp]WHOSE''

10061 PRINT''TOUCH[sp]IS[sp]FATAL
      [sp]TO[sp]YOU.YOU[sp]ALSO[s
      p]CANNOT[sp]LAND[sp, 2 time
      s]IN[sp]THE[sp]SULFURIC[sp]
      ACID''

10063 PRINT''OR[sp]HIT[sp]YOUR[sp
      ]HEAD[sp]ON[sp]1[sp]OF[sp]T
      HE[sp]LEDGES.''

10065 PRINT''[down][sp, 4 times]<
      HIT[sp]ANY[sp]KEY>''
```