

---

**COMMODORE**®

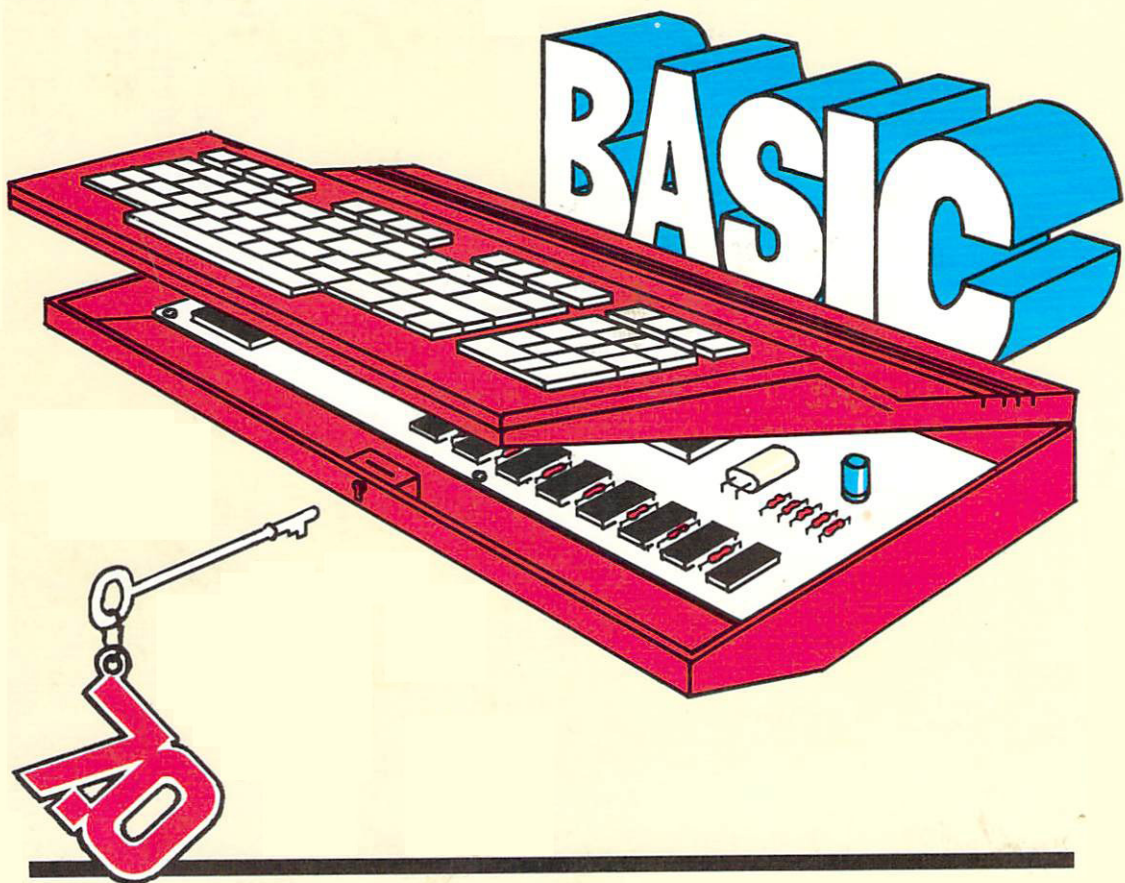
---

**BASIC 7.0**

**128**™

---

**INTERNALS**



---

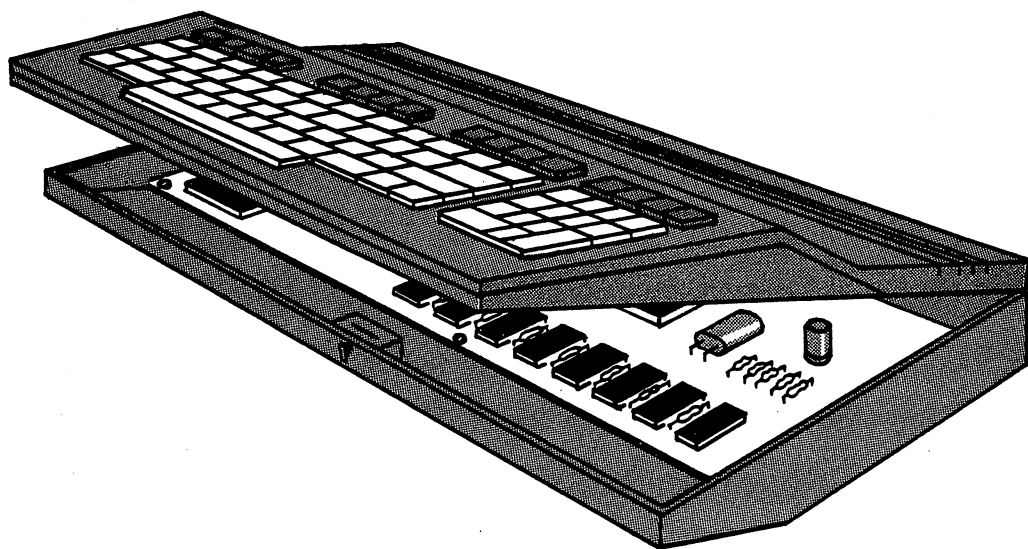
you can count on  
**Abacus** 



# C-128

# BASIC 7.0

# INTERNALS



Dennis Jarvis  
and  
Jim D. Springer

**Abacus** 

First Printing, January 1987  
Printed in U.S.A.  
Copyright © 1986

Copyright © 1987

Dennis Jarvis  
James Springer  
Abacus Software, Inc.  
P.O. Box 7219  
Grand Rapids, MI 49510

This book is copyrighted. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of Abacus Software, Inc.

Commodore 128, Commodore 64, BASIC 7.0, 1571, 1541 are trademarks or registered trademarks of Commodore Int., Ltd.

ISBN 0-916439-71-2



## Table of Contents

	Acknowledgements	v
Chapter 1	<b>How the BASIC Interpreter works</b>	1
1.1	The Pseudo stack in the C-128	5
1.1.1	How GOSUB/RETURN uses the Pseudo Stack	6
1.1.2	How DO/LOOP uses the Pseudo Stack	8
1.1.3	How FOR/NEXT uses the Pseudo Stack	9
1.2	Program Storage in the C-128	12
1.2.1	Resurrecting a BASIC Program	15
1.2.2	Recovering from a System "Crash"	16
1.3	Variable Storage Format in the C-128	17
1.3.1	Simple Variables	19
1.3.1.1	Floating Point Variables	20
1.3.1.2	Integer Variables	23
1.3.1.3	String Variables	24
1.3.2	Complex Variables	26
1.3.2.1	The Array Descriptor	27
1.3.2.2	Floating Point Arrays	30
1.3.2.3	Integer Arrays	31
1.3.2.4	String Arrays	31
1.3.3	Function Variables	32
1.3.4	Variable Pointers	33
1.3.5	A Variable Dump Program	36
1.3.5.1	How To Use The Variable Dump Program	36
1.3.5.2	The Variable Dump Generator Program	38
1.4	Memory Expansion for the C-128	40
1.4.1	RAM Expansion	41
1.4.1.1	The RAM Expansion Unit's Controller	41
1.4.1.1.1	DMA STatus register (DMAST)	43
1.4.1.1.2	DMA CoMmanD register (DMACMD)	43
1.4.1.1.3	DMA ADdress Low/High register	44
1.4.1.1.4	DMA LOw/HIgh register	44
1.4.1.1.5	DMA BaNK register	45
1.4.1.1.6	DMA Data Address Low/High register	46
1.4.1.1.7	DMA Interrupt Mask Register	46

1.4.1.1.8	DMA Address Control Register	47
1.4.1.2	The Software Nuts and Bolts of DMA	47
1.4.1.3	Graphics Demo Program	53
1.4.2	ROM Expansion	54
1.4.2.1	ROM Software Requirements	54
1.4.2.2	Auto-starting ROMs	56
1.4.2.3	Non-Auto-Starting ROMs	56
<b>Chapter 2</b>	<b>Wedging into BASIC</b>	<b>57</b>
2.1	The CHRGET/CHRGOT Routine	59
2.2	Wedging into the CHRGET/CHRGOT routine	61
<b>Chapter 3</b>	<b>Graphics</b>	<b>69</b>
3.1	Managing Memory	71
3.1.1	Moving Screen and Character Memory	73
3.1.2	Color Memory	76
3.1.3	Moving the Bitmap Screen	77
3.2	Hi-res graphics from machine language	79
<b>Chapter 4</b>	<b>The BASIC 7.0 ROM Listing</b>	<b>81</b>
	<b>Appendices</b>	<b>595</b>
Appendix A:	Cross reference from C-64 to C-128	597
Appendix B:	BASIC reserved word table with tokens and command addresses	603
Appendix C:	Hexadecimal / Decimal conversion table	607
Appendix D:	MMU Values	619
	<b>Index</b>	<b>627</b>

## Acknowledgements

It took the hard work and dedication of numerous people to make this final product a reality. It took long days and sleepless nights, and lots of coffee, sodas, and video games (for breaks) to complete the mountain of work and bring together all the bits and pieces into an organized manuscript. The number of painstaking hours spent commenting the BASIC ROMS alone staggers the imagination.

Each person involved in the book gave their all just to see the book to its end. Therefore, we the authors would like to thank, first and foremost, our families and friends for their undying support when things just did not seem to be going right.

We'd like to especially thank the following individuals for helping to enter and edit the pages upon pages of text and information that make up the book (listed in alphabetical order):

Linda L. Jarvis	—	Word Processor
Starla M. Jarvis	—	Word Processor
Kim B. Strandberg	—	Technical support
Michael L. Zinicola	—	Technical Writer and Editor
Patricia E. Zinicola	—	Word Processor

In addition to these people, we would like to also recognize those who indirectly contributed to the book.

Much thanks: to the parents of Dennis, Mr. and Mrs. LeRoy E. Jarvis, without whose help this book would not have been created, and to Jim Butterfield for his numerous articles and technical notes published in the form of magazines and books. To Glen Bredon and Southwestern Data Systems for the creation of MERLIN 64, which was instrumental in the commenting of the ROMS and developing the machine language programs. To Timeworks Inc. for creating WordWriter 128, the wordprocessor used for entering and editing the text and ROM comments.

To these people, our deepest appreciation and thanks. Without you, the book could never have been written in the four months that were allotted for its creation.

One other product that was used in the commenting of these routines was the MONITOR EXTENDER, which is available from J. Ingenito, RD 7 Woodmont Rd., Hopewell Jct, NY 12533. This program extends the MONITOR's capability with extra commands such as WALK, STACK, etc. Without it, many of the routines that switch banks during execution would have been much more difficult to comment.

Dennis Jarvis  
James Springer  
Kissimmee, Florida 1986

## **Chapter One**

# **How the BASIC Interpreter works**



## How the BASIC Interpreter Works

The BASIC interpreter is one of the most important parts of a computer. The interpreter is responsible for the following tasks, among others:

- 1) Allows the user to enter data through the keyboard either as a DIRECT mode command or stored as a BASIC program line.
- 2) If the data entered is a DIRECT mode command statement, the BASIC interpreter calls the proper routines to tokenize the command statement and execute the command statement.
- 3) If the data entered is a BASIC program line, the BASIC interpreter calls the routines necessary to tokenize the program line and store it in memory.
- 4) If the program stored in memory is to be RUN, the BASIC interpreter calls the necessary routines to execute the individual command statements in the BASIC program.

Now that we generally understand what the BASIC interpreter is supposed to do, let's take a closer look at how it works.

Upon powerup or reset, the 8502 microprocessor in the C-128 starts executing the RESET routine stored in locations \$FFFC and \$FFFD. This address is called the RESET Vector. It normally holds the address of the RESET routine at \$FF3D. However, like any other vector, the address stored here can be changed to point to any other address that the user wishes. In this case, it applies only to the reset button.

The RESET routine takes care of initializing the C-128 and finishes by calling the routine that displays the startup message to the screen. Once this has been done, control is turned over to the BASIC interpreter loop.

Let's mention another routine before we continue. It's the IRQ (Interrupt ReQuest) routine. Its address is stored in the IRQ Vector at locations \$FFFE and \$FFFF. The address that is normally stored here when the computer is initialized is \$FF17, which eventually jumps through an indirect vector located at \$0314. This vector can point to several things, such as the tape I/O routines, but normally this vector points to \$FA65—the normal entry point for the Interrupt ReQuest (IRQ) routine.



Every 1/60th of a second, the IRQ routine scans the keyboard to see if a key has been pressed. If a key is pressed, the ASCII value which corresponds to that key is stored in the keyboard buffer to await removal by BASIC.

Once the key value has been stored by the IRQ routine in the buffer, the BASIC interpreter can later retrieve the ASCII value and store it into the INPUT buffer. The process continues until the ASCII value for the <RETURN> key (or <SHIFT><RETURN>) is detected.

When the value for the <RETURN> key is detected, the BASIC interpreter checks to see if the first character in the statement is a number. If it's a number, the BASIC interpreter calls the necessary routines to tokenize and store the statement in memory as a BASIC program line. However, if the first character is not a number, then the BASIC interpreter calls the necessary routines to tokenize the statement and then execute it as a DIRECT mode statement.

For instance, if you type the statement PRINT without a preceding line number and then press the <RETURN> key, the statement is immediately tokenized and executed. This will PRINT a blank line to the screen. However, if you type the same PRINT statement preceded by a number and press the <RETURN> key, the statement is tokenized and stored in memory as a BASIC program line.

When the BASIC RUN command is used to execute the BASIC program stored in memory, the BASIC interpreter is responsible for calling the different routines that search for the command tokens and execute the commands that are specified.

The BASIC interpreter must keep track of many values and addresses in order to RUN a program correctly. It uses what is known as a stack to keep track of the various parameters and addresses. This can be thought of as a stack of papers piled on top of each other, each sheet containing a single value. Because the stack is such an important concept in the BASIC interpreter's operation, we'll go into much greater detail on the stack's operation in the next section.

## 1.1 The Pseudo Stack in the C-128

The term *stack* is used to describe an area of memory set aside as a work area for storing the parameters necessary for the operation of a routine. In the C-128, there are two such stacks: the *processor stack*, located at \$0100 - \$01FF, and the *pseudo stack*, located at \$0800 - \$09FF.

The processor stack is used by the 8502 when executing machine language instructions such as PHA, which pushes (saves) the value in the accumulator onto the processor stack, and PLA, which pulls (retrieves) the topmost value from the stack and places it into the accumulator. Each time an operation is performed using the stack, an internal register in the 8502 called the Stack Pointer is automatically incremented or decremented so that it points to the next available stack location to be used. For example, when a PHA is executed, the stack pointer is decremented. When a PLA is executed, the stack pointer is incremented. If the stack pointer should get too full when the processor stack is used, then a 'FORMULA TOO COMPLEX' error is displayed.

The second stack is the pseudo stack, which is used by the BASIC routines GOSUB, FOR/NEXT, and DO/LOOP. Each of these routines stores a different set of parameters that it needs to accomplish its respective task. The stack pointer for the pseudo stack is located in memory at \$7D and \$7E.

However, unlike the processor's stack pointer, the pseudo stack pointer is not automatically updated when information is stored or retrieved from it. The individual routines that use the pseudo stack are responsible for updating the stack pointer.

For example, if a routine needs to store a parameter on the pseudo stack, it will store the value to the pseudo stack and then decrement the pseudo stack pointer to point to the next available memory location on the stack. If a routine needs to get a parameter from the pseudo stack, the pseudo stack pointer is incremented and the value that it points to is retrieved. If the pseudo stack pointer should get too full when the processor stack is used, then an 'OUT OF MEMORY' or 'FORMULA TOO COMPLEX' error is displayed, depending on when the stack overflow occurred.

Now let's discuss how the GOSUB/RETURN, FOR/NEXT, and DO/LOOP command routines use the pseudo stack to accomplish their individual jobs.

The machine language routines for these commands can be found in the BASIC ROMs at the following locations:

<u>Routine</u>	<u>Location</u>
GOSUB/RETURN	\$59CF and \$5262
FOR/NEXT	\$4DF9 and \$57F4
DO/LOOP	\$5FE0 and \$608A

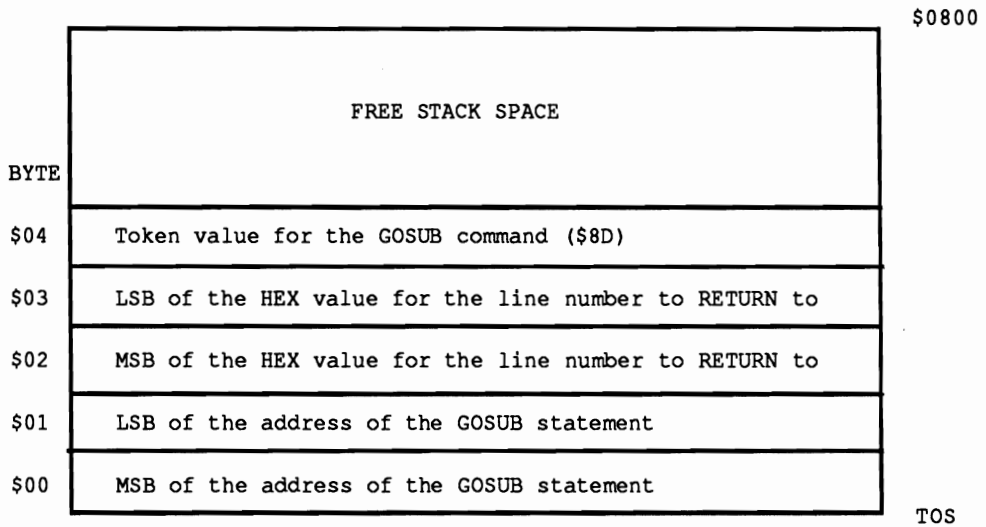
### 1.1.1 How GOSUB/RETURN Uses the Pseudo Stack

The GOSUB/RETURN command combination is used when you want to perform a subroutine that is identified by another line number in a BASIC program, and then return to your original line number in the same program. Here is an example of the GOSUB/RETURN command combination:

```
10 PRINT "HELLO"  
20 GOSUB 50  
30 PRINT "EVERYBODY"  
40 END  
50 PRINT "THERE"  
60 RETURN
```

When the program is running and the BASIC interpreter comes across the token for GOSUB, it will call the GOSUB routine at \$59CF .

The GOSUB routine at \$59CF stores five bytes of information on the pseudo stack that are to be used by the RETURN command routine at \$5262. These five bytes of information are stored in the following order, starting at the memory location pointed to by the Top Of Stack pointer (TOS) and working downward towards the bottom of the stack at \$0800:



TOS = Top Of Stack pointer (Pseudo stack pointer)

**Note:** The address that is stored in Bytes \$00 and \$01 points to the first byte after the token for the GOSUB command.

After the information has been stored on the stack, the program continues execution at the line number specified after the GOSUB command, and continues until the token for the RETURN command (\$8E) is encountered.

When a RETURN command is encountered, the five bytes of information that were stored on the pseudo stack by the GOSUB routine are used by the RETURN routine. This routine searches for the first occurrence of the GOSUB token and then retrieves the information that was stored there, but in the reverse order. This information is then used to return the program execution to the place where the GOSUB command was used to continue the program at the next statement or line number.

### 1.1.2 How DO/LOOP uses the Pseudo Stack

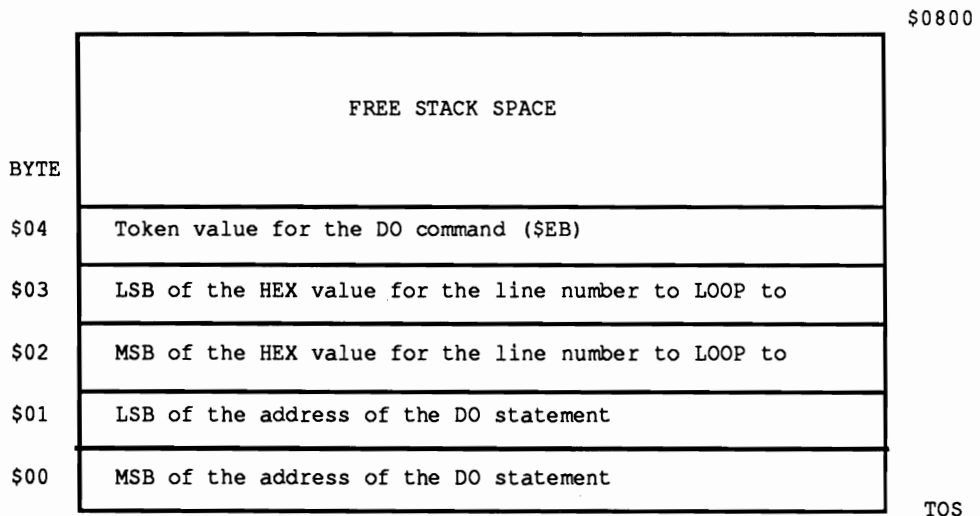
The DO/LOOP command combination is used when you want to execute a certain area of the BASIC program over and over. An example of the DO/LOOP command follows:

```

10 DO
20 PRINT "HELLO THERE EVERYBODY"
30 LOOP
    
```

When the program is running and the BASIC interpreter comes across the token for DO, it calls the DO routine at \$5FE0. The DO routine stores five bytes of information on the pseudo stack that are to be used by the LOOP command routine at \$608A.

These five bytes of information are stored on the pseudo stack starting at the memory location pointed to by the Top Of Stack pointer (TOS), and working downward towards the bottom of the stack at \$0800:



TOS = Top Of Stack pointer (Pseudo stack pointer)

Note: The address that is stored in Bytes \$00 and \$01 points to the first byte after the token for the DO command.

After the information has been stored on the stack, the program continues executing with the next line number until the token for the LOOP command (\$EC) is encountered.

When a LOOP command is encountered, the five bytes of information that were stored on the pseudo stack by the DO routine at \$5FE0 are used by the LOOP routine at \$608A. This routine searches for the first occurrence of the DO token (\$EB) and then retrieves the information that was stored there, but in the reverse order. This information is then used to loop the program execution to the place where the DO command is located.

### 1.1.3 How FOR/NEXT uses the Pseudo Stack

The FOR/NEXT command combination is used when you want to execute a certain area of the BASIC program a specific number of times. This command combination can also be used as a delay loop. An example of FOR/NEXT:

```
10 FOR X=1 TO 10
20 PRINT "HELLO THERE EVERYBODY"
30 NEXT
```

When the program is running and the BASIC interpreter comes across the token for FOR, it will call the FOR routine at \$5DF9. The FOR routine stores eighteen bytes of information on the pseudo stack.

These eighteen bytes of information are stored on the pseudo stack in the following order starting at the memory location that is pointed to by the Top Of Stack pointer (TOS) and working downward towards the bottom of the stack at \$0800:

\$0800

FREE STACK SPACE	
BYTE	
\$11	Token value for the FOR command (\$81)
\$10	LSB of the address of the variable name's descriptor
\$0F	MSB of the address of the variable name's descriptor
\$0E	EXponent of the floating point STEP value
\$0D	M1 of the floating point STEP value
\$0C	M2 of the floating point STEP value
\$0B	M3 of the floating point STEP value
\$0A	M4 of the floating point STEP value
\$09	Sign of the STEP value (\$01 = Positive, \$FF = Negative)
\$08	EXponent of the floating point variable limit value
\$07	M1 of the floating point variable limit value
\$06	M2 of the floating point variable limit value
\$05	M3 of the floating point variable limit value
\$04	M4 of the floating point variable limit value
\$03	LSB of the line number that the FOR statement is on
\$02	MSB of the line number that the FOR statement is on
\$01	MSB of the address of the next statement in the program
\$00	LSB of the address of the next statement in the program

TOS

TOS = Top Of Stack pointer (Pseudo stack pointer)



After the information has been stored on the stack, the program continues executing with the next line number until the token for the NEXT command (\$82) is encountered.

When a NEXT command is encountered, the eighteen bytes of information that were stored on the pseudo stack by the FOR routine are used by the NEXT routine. This routine searches for the first occurrence of the FOR token (\$81) and then retrieves the information that was stored there, but in the reverse order. This information is then used to determine whether the FOR/NEXT loop is finished or whether it needs to be executed again.

## 1.2 Program Storage in the C-128

The format of a program that has been stored in C-128 memory is very similar to the format stored by the other home computers that Commodore produces. However, the C-128 does tokenize some of the BASIC commands and functions differently.

In the C-128, the added commands and functions make it necessary to use dual token commands and functions. Dual token commands and functions consist of a two byte token, with the first byte value being either a \$CE or \$FE. An example of a dual token command is the command PLAY. This command when tokenized and stored in memory consists of the two bytes \$FE and \$04. This is indeed different than a single token command like PRINT, which is tokenized as a one byte value of \$99 (See Appendix B).

The start of BASIC text usually begins at location \$1C01 in RAM bank 0. This is the starting address that will be used for the examples discussed in this text.

For the following discussion, it is easier to explain the storage of a BASIC program if an example program is used. Consequently, turn your computer on and type in the following short BASIC program:

```
10 PRINT"HELLO"
20 END
```

After you have entered the program, enter the MONITOR and type M 01C00 01C17. This displays a memory dump of the area of memory in RAM bank 0 where the BASIC program was stored. Your screen will look like this:

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----
>01C00: 00 0E 1C 0A 00 99 22 48 45 4C 4C 4F 22 00 14 1C
>01C10: 14 00 80 00 00 00 -- -- -- -- -- -- -- --
```

The first thing that can be found stored at \$1C00 is the single zero byte which indicates the end of a BASIC line. This memory location must always be set to zero when storing a BASIC program, or else a 'SYNTAX' error is issued when you try to RUN the program.

Next, stored in locations \$1C01 and \$1C02 are the LSB (Least Significant Bit) and MSB (Most Significant Bit), respectively, of the line link address that points to the line link of the next program line. In our example, locations \$1C01 and \$1C02 point to location \$1C0E.

Following these two bytes at \$1C03 and \$1C04 are the LSB and MSB value, respectively, of the program line number entered with the BASIC program line. In our example, the line number value is \$000A or 10.

Following the program line number is the tokenized BASIC program line that represents the original BASIC program line. Each keyword is converted to a one or two byte token and then stored in memory. However, the text that is to be printed, which was enclosed in quotes when the program line was typed in, is only converted to its HEX equivalent and stored in memory.

The first program line is represented by the following partial memory dump:

```

          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D
          -----
>1C00: 00 0E 1C 0A 00 99 22 48 45 4C 4C 4F 22 00
    
```

The byte following the MSB of the program line number at location \$1C05 is the tokenized value for the command PRINT (\$99). The seven bytes that follow the token for PRINT from location \$1C06 to \$1C0C are the ASCII values for the quotes and letters of the text to be printed, "HELLO". A single zero byte is stored at location \$1C0D to indicate the end of the program line.

Now let's see how the last program line looks in memory:

```

          0E 0F 10 11 12 13 14 15
          -----
>1C0D: 14 1C 14 00 80 00 00 00
    
```

As you can see, the next BASIC program line starts at locations \$1C0E and 1C0F. These are the LSB and MSB of the line link address that points to the next line link address of the next program line. In this case, the address is \$1C14.

The next two bytes, locations \$1C10 and \$1C11, are the LSB and MSB, respectively, of the second program line's line number. Here the line number is \$0014, or decimal 20.

Next, the token for the END command is stored at \$1C12 (\$80). Since the END command is the only thing in this BASIC program line, the end of line marker (\$00) is stored at location \$1C13.

This sequence of bytes continues until the last program line is reached. Since line 20 is the last program line, two consecutive zero bytes are stored at locations \$1C14 and \$1C15. This constitutes the end of BASIC program marker.

The address of the byte after the end of BASIC program marker is stored in locations \$1210 and \$1211. The address stored in these locations is used by the various routines such as the LOAD, SAVE, and RENUMBER routines so that the end of the BASIC program can be determined. These locations are updated each time a program line is inserted or deleted.

Let's take a moment to illustrate with a simple diagram how the line links, etc., are used:

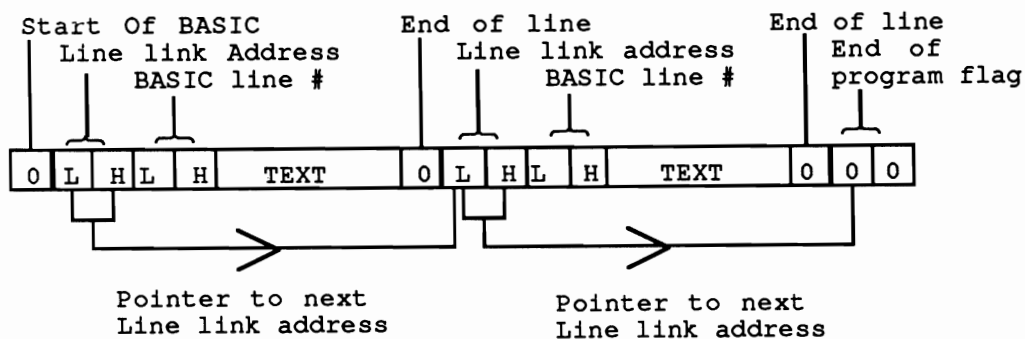


Figure 1.2

## 1.2.1 Resurrecting a BASIC Program

Now that we have seen how a BASIC program line is stored in memory, let's see what happens when we use the command `NEW` and some ways to `unNEW` a program that has been accidentally "erased" from memory.

When you type in the command `NEW` and press `<RETURN>`, two things happen. First, two zero bytes are stored at the beginning of the BASIC text memory. Normally these locations are `$1C01` and `$1C02`. Consequently, a memory dump of the area of memory where the BASIC program was stored would look like this:

```

      00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
      -----
>01C00: 00 00 00 0A 00 99 22 48 45 4C 4C 4F 22 00 15 1C
>01C10: 14 00 80 00 00 00 -- -- -- -- -- -- -- --

```

Next, the end of BASIC pointer at `$1210` and `$1211` is updated to point to one byte past the last zero byte that was just written. So the pointer is updated to `$1C04`. The address that locations `$1210` and `$1211` would now point to would be location `$1C04`.

That is really all that happens when you use the `NEW` command. The previous BASIC program is not erased. Therefore, if the first line link address can be restored and the program lines linked together, the BASIC program can be `unNEW`ed.

There are many ways of accomplishing this, but two methods are very simple to use. Due to some software quirks in the BASIC 7.0 ROMs, they are available to everyone. First, type in the following BASIC program:

```

10 PRINT "HELLO"
20 END

```

Now type `NEW` and press `<RETURN>`. Type `LIST` and press `<RETURN>` to verify that the program is really gone—or so we think. Now type in the following statement in the `DIRECT` mode and press `<RETURN>`:

```
POKE DEC("1C01"),1:RENUMBER
```

Now type `LIST` and press `<RETURN>` to verify that the BASIC program has really returned. "Amazing!" you say. Well, now try the next method.

Since the program has returned, make it disappear again with the `NEW` command. Again type `LIST` and press `<RETURN>` to verify that it is indeed gone. Now type in the following statement in the `DIRECT` mode and press `<RETURN>`:

```
POKE DEC("1C01"),1:DELETE 0
```

Now type `LIST` and press `<RETURN>` and again the `BASIC` program has been returned. How the `unNEW` is accomplished is hidden in the ROMs and can be found in the ROM listing under the routines for the `RENUMBER` and `DELETE` commands.

## 1.2.2 Recovering from a System "Crash"

Occasionally the C-128 will hang up, such as is the case when a `SYS` to a wrong address is executed and puts the microprocessor into an infinite loop. When this happens, there are four ways to recover from the crash:

- 1) Turn the computer off and then on.
- 2) Press the reset button and use the `unNEW` examples in Section 1.2.1.
- 3) Press the `<RUN/STOP><RESTORE>` key combination.

The fourth way you might recover from a system crash is outlined below:

- 1) Depress and hold down the `<RUN/STOP>` key.
- 2) While still holding down the `<RUN/STOP>` key, press the `RESET` button on the right side of the 128 and release it.
- 3) After you see that the computer has powered up in the machine language monitor, release the `<RUN/STOP>` key.
- 4) Check to ensure that location `$0A04` has a `$C1` in it to ensure that the `BASIC IRQ`, etc., has been initialized. If it has a `$C0` in it, change it to `$C1`.
- 5) Type an `X` and press `<RETURN>` to return to `BASIC`.
- 6) Type in `LIST` and press `<RETURN>` to list the recovered program.

### 1.3 Variable Storage Format in the C-128

The way the C-128 handles its variables is not too much different than the way that the C-64 handles its variables. However, in the 128, the zero page pointers have been moved and the variables themselves are stored in RAM BANK 1 instead of being stored immediately following the BASIC text, as it's done in the 64. This is a distinct advantage to the BASIC programmer. With the 128, if a program stops due to a 'SYNTAX' error or any other reason short of crashing the computer, the programmer can edit the line that is causing the problem and restart the program with a single GOSUB or GOTO command. Unlike the 64, this will not destroy the variable values that were stored in memory between the time the program started running and the time it crashed. This is the main difference in the way the 128 and the 64 handle their variables. Now that we have covered the difference between the way that the 64 and 128 handle their variables, let's move on to discuss the actual format used by the 128 to store variables.

In the 128, as in the 64, there are three categories of variables: Simple, Complex, and Function variables. The Simple and Complex variables can be broken down into three types: Floating Point (FP), Integer, and String variables. However, Function variables can only be Floating Point type.

Each variable is allocated a seven byte descriptor in RAM BANK 1 starting at \$0400 to indicate the type of variable (the value of the variable or length of the variable if it is a string, and the address of the variable in RAM BANK 1, if the variable is a string).

You assign each variable a name. A variable name may consist of one or two characters, and include any alphanumeric character (as long as the first character of the variable name is a letter, not a number). If the first character of the variable name is found to be a number by the BASIC interpreter, a 'SYNTAX' error will occur.

Also, if the variable name is followed by a percent sign (%) or a dollar sign (\$), it is defined as an Integer or String variable, respectively. However, if the variable name is not followed by either % or \$, the variable is designated as a Floating Point variable.

There are a few variable names that are reserved exclusively for the use of the 128's operating system. These variable names are as follows: DS, DS\$, EL, ER, ERR\$, ST, TI, TI\$. Also, any variable name that has a reserved



keyword imbedded in it cannot be used. For instance, you cannot use a variable name such as 'TO' because this is one of the reserved keywords of the BASIC operating system.

After a variable name has been designated, the BASIC operating system evaluates the name to determine if it is valid. If the variable name is invalid, a 'SYNTAX' error will occur. However, if the variable name is valid, bit seven of one, both, or neither of the characters in the variable name is set to indicate which of the three types the variable is. Let's take a moment to clarify how the type of variable is represented in the variable name.

- 1) If the variable is a Floating Point variable, the values stored in Bytes 0 and 1 of the variable descriptor represent the HEX value of the two characters assigned to that particular variable. If there is only one character assigned to the variable, Byte 1 will contain the HEX value \$00.
- 2) If the variable is an Integer (%), the values stored in Bytes 0 and 1 of the variable descriptor represent the HEX value of the two characters assigned to that particular variable, but, with bit seven set in both of the characters. If there is only one character assigned to the variable, Byte 1 will contain the HEX value \$80.
- 3) If the variable is a String (\$), the values stored in Bytes 0 and 1 of the variable descriptor represent the HEX value of the two characters assigned to that particular variable. However, this time, bit seven is set only in the second character which is stored in Byte 1. If the variable name only contains one character, Byte 1 of the variable descriptor will contain the HEX value \$80.
- 4) If the variable is a Function variable (FN), the values stored in Bytes 0 and 1 of the variable descriptor represent the HEX value of the two characters assigned to that particular variable. However, bit seven is set only in the first character which is stored in Byte 0. If there is only one character assigned to the variable, Byte 1 will contain the HEX value \$00.

To further clarify this point, let's see what each type of variable name looks like in memory. The following figure illustrates how each type of variable is represented by setting bit seven in one, both, or neither of the characters in the variable name.

VARIABLE NAME	BYTE 00, 01	VARIABLE TYPE
AA	41, 41	Floating Point
AA%	C1, C1	Integer
AA\$	41, C1	String
DEF FN AA (XX)	C1, 41	Function variable

Now that we have discussed how each variable type is identified in the variable name, let's cover how these different types appear in memory for the three categories of variables: Simple, Complex, and Function variables.

### 1.3.1 Simple Variables

Simple variables are non-array variables. This category of variables can be broken down into three types: Floating Point, Integer, and String variables. We will take each type and show how each appears in memory and describe the meaning of each byte that is associated with that type of variable.

First, type in and run the following short program, it will make it easier to see how actual values are stored using each type.

```

10 AA = 1.7
20 AA% = -117
30 AA$ = "C-128"
    
```

The table below shows a memory dump of the area where those variables are stored. Remember, variable storage starts in RAM BANK 1 at \$0400.

ADDRESS: IN HEX	BYTE	VARIABLE TYPE	BASIC STATEMENT
>10400	:41:41:81:59:99:99:9A:	Floating Point	AA = 1.7
>10407	:C1:C1:FF:8B:00:00:00:	Integer	AA% = -117
>1040E	:41:C1:05:F9:FE:00:00:	String	AA\$ = "C-128"

### 1.3.1.1 Floating Point Variables

Floating Point variables (FP) are used when a high degree of accuracy is needed in mathematical calculations. This is because a FP value can range from 2.93873588 E - 39 to 1.70141183 E + 38.

The C-128 calculates the FP value to 10 digits of precision which requires 5 bytes of storage area as seen in the table below. For an example of how a FP variable is stored in memory, type the following in DIRECT mode and press <RETURN>:

```
CLR:A = 128
```

Now, enter the MONITOR and type M 10400 10407 and press <RETURN>. This will display the memory in RAM BANK 1 from \$0400 to \$0407 which contains the descriptor of the variable A as shown below.

```
>10400: 41 00 88 00 00 00 00
```

Since this type of variable is a Floating Point variable, bit 7 is not set in either of the two characters of the variable name stored in bytes 0 and 1.

The description of the seven bytes that make up the Floating Point Variable Descriptor is shown below.

FLOATING POINT VARIABLE DESCRIPTOR		
BYTE	00	Variable name with bit seven not set
	01	Variable name with bit seven not set
	02	Exponent value (EXP)
	03	M1
	04	M2
	05	M3
	06	M4

The way the C-128 represents a FP value at first seems to be complicated but in reality, it is not. For example, to convert the Integer value 53 to its binary FP equivalent, the following steps must be followed :

1. Calculate the highest power of 2 that will divide evenly into the specified value. The closest power of two that will divide into the value 53 is 5. After dividing the value of 53 by 2 to the fifth power, this leaves a remainder of 21 (\$15).

$$53 = 2^5 \text{ with a remainder of } 21 \text{ (\$15)}$$

2. Derive the exponent by adding 129 (\$81) to power of two that was calculated in the step 1.

$$5 + 129 = 134 \text{ (\$86)}$$

3. Convert the remainder from the operation in step 1 to binary format.

$$21 = 00010101 = (2^4 + 2^2 + 2^0) = (16 + 4 + 1)$$

4. Take the highest power of 2 that was obtained in step 1 (which is 5 in this example) and subtract one from it which will give you  $2^4$ . Then, starting with bit 6 of M1, write 24, etc until you reach bit 0 of M4 which would have  $2^{-26}$ .

Then take the binary representation of 21 and starting from bit 6 in M1 (Bit 7 is the sign bit) moving toward bit 0, locate the first bit that is set, which in this example would be bit 4. This bit represents  $2^4$ , so place that '1' into BIT 6 of M4. Then place the remaining bits, regardless if they are set or cleared, into the remaining bits in M1 until you place the last bit of the binary value for 21, which is a one in this case, into bit 2 of M1. Then fill the remaining bits in M1 with zeros along with M2 - M4. Finally, since the original value was positive, place a zero for BIT 7 in M1. If the value would have been negative you would have set BIT 7 to indicate this. After you have completed the representation of M1 thru M4, they should look like the figure on the next page.

<p style="text-align: center;">1 0 0 0 0 1 1 0 7 6 5 4 3 2 1 0 Exponent</p>						
<p>0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 0      0 1 0 1 0 1 0 0      0 1 0 1 0 1 0 0</p>			Binary Representation Bit value			
<p>4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10      -11-12-13-14-15-16-17-18      -19-20-21-22-23-24-25-26</p>						
<p>2 2 2 2 2 2 2 2 2 2 2 2 2 2 2      2 2 2 2 2 2 2 2      2 2 2 2 2 2 2 2</p>						
<p>7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0      7 6 5 4 3 2 1 0      7 6 5 4 3 2 1 0</p>			Mantissa			
M1	M2			M3		M4
<p>Floating Point representation of the value 53</p> <p>Exp = \$86 M1 = \$54 M2 = \$00 M3 = \$00</p>						

### 1.3.1.2 Integer Variables

As you can see in the table below, an Integer variable's value is stored in MSB, LSB format respectively. An Integer value can range from -32768 to 32767. In HEX, these values are represented as \$FFFF to \$7FFF with \$8000 to \$FFFF representing -32768 to -1, and \$0000 to \$7FFF representing 0 to 32767. Integer variables should be used as often as possible to increase the execution speed of a program.

For an example of how an Integer is stored in memory, type the following in the DIRECT mode and press <RETURN>:

```
CLR:A% = 32767
```

Now, enter the MONITOR and type M 10400 10407 and press <RETURN>. This will display the memory in RAM BANK 1 from \$0400 to \$0407 which contains the descriptor of the variable A% as shown below.

```
>10400: C1 80 7F FF 00 00 00
```

Since this type of variable is an Integer variable, bit 7 is set in both of the two characters of the variable name that are stored in Bytes 0 and 1.

The description of the seven bytes that make up an Integer Variable Descriptor is shown below.

INTEGER VARIABLE DESCRIPTOR		
BYTE	00	Variable name with bit seven set
	01	Variable name with bit seven set
	02	MSB of the 16 bit binary value
	03	LSB of the 16 bit binary value
	04	not used filled with zero
	05	not used filled with zero
	06	not used filled with zero

### 1.3.1.3 String Variables

String variables are not quite as difficult to handle or understand as Floating Point Numbers but require more effort to manage than Integer variables. The creation of a string variable by the operating system is fairly straightforward. With the exception of the system's reserved strings such as DS\$, the main entry point for the creation of a string is usually through the BASIC LET command at \$53C6. There are many routines that are required to manage string variables such as the routines that add and delete the string from memory by either adding it to the bottom of the string storage area or moving the strings that are below it up over the one to be deleted. The actual storage format for the string variable itself is very simple.

For an example on String variable storage, let's create a string in memory by using the normal BASIC method. Type in the following statement and press <RETURN>:

```
CLR:A$="C-128"
```

Now let's see what happened. When the CLR command was executed, it reset the system pointers back to their normal values. This simply told the operating system to start placing the variable descriptors at \$0400 moving upward towards \$FF00 and to start placing the actual strings at \$FF00 moving downward towards \$0400.

Now enter the monitor by typing MONITOR in the DIRECT mode and pressing <RETURN>. Enter M 10400 10407 and press <RETURN> to inform the monitor that you wish to see a HEX dump of the memory from \$0400 to \$0407 in RAM BANK 1. This area of memory is where the variable descriptors are stored. If you look at the first seven bytes which are the variable descriptor for A\$, you will see the following:

```
>10400 :41 80 05 F9 FE 00 00
```

Now if you start with the first byte of the descriptor (nominal byte zero) which is at \$0400, you will find a HEX value of \$41. This is the first character of the variable name which is the letter A. The second character of the variable name, stored in Byte 1, was not specified. Therefore, the operating system set bit seven in the second byte of the descriptor in order to indicate that this descriptor is for a string.



The description of the seven bytes that make up a String Variable Descriptor is shown below.

STRING DESCRIPTOR		
BYTE	00	Variable name with bit seven not set
	01	Variable name with bit seven set
	02	Length of the string
	03	LSB of the address of the string in RAM BANK 1
	04	MSB of the address of the string in RAM BANK 1
	05	not used filled with zero
	06	not used filled with zero

The following table indicates how the string that was assigned to A\$ above would be stored in RAM.

STRING STORED IN RAM BANK 1		
ADDRESS	FEF9	C
	FEFA	-
	FEFB	1
	FEFC	2
	FEFD	8
	FEFE	02 : LSB, MSB of the
	FEFF	04 : address of this string's descriptor

Remember that strings are added one character at a time moving downward toward \$0400. The two bytes that precede the string (\$FEFE and \$FEFF) are the address which points to the length of the string in the variable descriptor. The reason for this address is to enable garbage collection to scan through the string storage area and locate which strings are no longer required. The flag for the garbage collection is very simple. When a string is to be discarded, a routine will place an \$FF in the MSB of the descriptor's address after the string and place the length of the string in the LSB of the descriptor's address. For example, in DIRECT mode type the following statement and press <RETURN>.

CLR:A\$ = "COMMODORE":B\$ = "64":C\$ = "128":B\$=""

The operating system first performed a CLR. Then it assigned the string variables and placed the strings in the string storage area. When it reached the statement `B$=""`, it placed the new string ("") at the bottom of the string storage area and changed the address of the variable descriptor after the `B$="64"` to `$FF02`, in LSB, MSB format. Go into the MONITOR and enter `J F92EA` which forces a garbage collection. Now look at what happened to your strings. You now know how garbage collection is performed and why, if you have a lot of strings and you delete the first string that was created, your computer seems to hang on you!

### 1.3.2 Complex Variables

A Complex variable is another name for an Array variable. This category can be broken down into three types: Floating Point, Integer, and String variables. We will take each type and show how each appears in memory and then describe the meaning of each byte that is associated with that type of variable.

The first thing that should be noted is that the three types of complex variables are identified by setting Bit 7 in one, both, or neither of the characters in the variable name that is assigned to the array. This is the same method of identification that is use with the simple variable types.

The following figure illustrates how the name of each type of complex variable is represented in memory.

VARIABLE NAME	BYTE 00,01	VARIABLE TYPE
AA(0)	41,41	Floating Point
AA%(0)	C1,C1	Integer
AA\$(0)	41,C1	String

To better understand how the different variable types are stored in memory, type in and RUN the following BASIC program:

```

10 DIM AA(0), AA%(0), AA$(0)
20 AA(0) = 1.7
30 AA%(0) = -117
40 AA$(0) = "C-128"

```

The following figure represents what a memory dump of the area where the variables are stored would look like:

ADDRESS: IN HEX	BYTE	VARIABLE TYPE	STATEMENT
>10400	:41:41:0C:00:01:00:01:81:59:99:99:9A:	Floating Point	AA(0) = 1.7
>1040C	:C1:C1:09:00:01:00:01:FF:8B:41:C1:0A:	Integer	AA%(0) = -117
>10418	:00:01:00:01:05:F9:FE:	String	AA\$(0) = "C-128"

### 1.3.2.1 The Array Descriptor

The array descriptor is basically the same for all three types of complex variables. The first two bytes contain the variable name and the aforementioned flags set in the name to indicate the variable type. The next five bytes describe the array size and how many dimensions and how many elements that are in the array. The array descriptor is only used once for each different array that is defined. For example, if you define an array of ten elements, the descriptor is used once followed by a list of the elements.

The table below describes each of the bytes that can be found in an array descriptor:

ARRAY DESCRIPTOR	
BYTE 00	Variable name with the flags set in
01	The same manner as simple variables
02	This byte contains the number of bytes in the ARRAY descriptor (7) + (the number of bytes which make up
03	the ARRAY ELEMENT * the number of elements which is stored in LSB,MSB format
04	The number of dimensions in the array
05	The number of elements in the array which was
06	specified in the DIM statement +1 because the A\$(0) is a valid element

When an ARRAY descriptor is created it is formed in a totally different manner than the SIMPLE descriptors. Let's take each one of the bytes in the descriptor and spend a little time with them. Enter the following BASIC program so that you will be able to follow along with us.

```

10 DIM A % (1,2,3,4,5)
20 DIM B $ (10,10)
30 A % (0,0,0,0,0) = 1
40 B $ (0,0) = "BASIC 7.0 INTERNALS"

```

RUN the aforementioned program then do a MEMORY dump of 10400 thru 10410 and 109AF 109BF which should look like this.

```

                                NOMINAL BYTE
                                -----
                                00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
-----
>10400 :C1:80 :AF:05 :05 :00:06 :00:05 :00:04 :00:03 :00:02 :00:01
                                -----
                                ^
                                FIRST ELEMENT -----
>109AF :42:80 :74:01 :02 :00:0B :00:0B :13:EB :FE:00 :00:00 :00:00
                                -----
                                ^         ^
                                FIRST ELEMENT ---- ^
                                ^
                                SECOND ELEMENT -----

```

Bytes 0 through 1 are used in the same manner as SIMPLE variables.

Bytes 2 through 3 are used to inform the operating system of two things. First of all, they are used to find the address of the next ARRAY descriptor. In our example these bytes contain AF 05 which, when added to the current descriptor's address (\$0400), would give us the address of the next ARRAY descriptor, which is \$09AF. This is also used by the operating system as the address of the last ARRAY's element. This value is obtained by the following formula:

$$\text{ADDRESS} = 7 + (2 * (\text{ND} - 1)) + (\text{VAR} * (\text{EL}1 + 1 * \text{EL}2 + 1 \dots))$$

Where:

ND = The number of dimensions - 1 (5,5 would be 2 dimensions)

EL = The element value of the array (5,5 - EL1 = 5 , EL2 = 5)

VAR = Number of bytes per variable type :

Integers	=	2
Strings	=	3
Floating point	=	5

Using this formula ,bytes 2 and 3 of our first array would be calculated as follows :

$$\begin{aligned} &7 + (2 * 4) + (2 * (2 * 3 * 4 * 5 * 6)) \\ &15 + (2 * 720) \\ &1455 \end{aligned}$$

When 1455 is converted to hex format the result is \$05AF which is stored in byte 2 and 3 as AF:05.

Byte 4 is used to indicate the number of DIMENSIONS in the ARRAY, this byte is used in conjunction with byte \$05 thru NN and in our first example this value is 5 which indicates the number of DIMENSIONS that are used in this array.

Byte 5 thru NN (0A) is used to indicate the number of elements in each dimension, in the statement DIMA%(10,10) these two bytes would contain \$00,\$0B in LSB/MSB format, followed by \$00,\$0B, followed by byte 9 which is the first element in the ARRAY which is (0,0,0).

As stated before, the main difference between the different types of arrays is in the way that the elements of each type are stored. We will take each type, describe each byte of the element, and then show what an array of that particular type would look like in memory.

### 1.3.2.2 Floating Point Arrays

The Floating Point array is used to stored numbers with 10 digit accuracy.

The array consists of the array descriptor followed by a list of the values of the five byte Floating Point numbers that were assigned to the array.

For example, type in the following statement and press <RETURN>.

```
CLR:DIM A(0):A(0) = 328.67
```

The following table illustrates how this type of array would appear in memory if you were to do a M 10400 1040C from the MONITOR.

```
>10400 :41 00 0C 00 01 00 01 89 24 55 C2 8F
```

The following table describes the meaning of the individual bytes that follow the array descriptor. The first Floating Point element begins at location \$10408.

FLOATING POINT NUMBER ELEMENT		
BYTE	00	Exponent value
	01	M1
	02	M2
	03	M3
	04	M4

If there was more than one element assigned to the array, the same sequence of bytes for each Floating Point Number element would be listed following the descriptor.

### 1.3.2.3 Integer Arrays

The Integer array is used to store numbers that do not require the degree of accuracy that the Floating Point Number array has to offer. The format in which the Integer array is stored is the same as the Floating Point array except, the Integer array only requires two bytes for each element.

For example, type in the following statement and press <RETURN>.

```
CLR: DIM A%(0) : A%(0) = 1296
```

The following line illustrates how this type of array would appear in memory if you were to do a M 10400 10409 from the MONITOR.

```
>10400 :C1 80 09 00 01 00 01 05 10
```

The following table describes the meaning of the individual bytes that follow the array descriptor. The first Integer element begins at location \$10407.

INTEGER ARRAY ELEMENT	
BYTE 00	MSB of the 16 bit binary value
01	LSB of the 16 bit binary value

If there were more than one element assigned to the array, the same sequence of bytes for each Integer element would be listed following the array descriptor.

### 1.3.2.4 String Arrays

The String array is used to store strings of characters and not numerical values like the Floating Point and Integer arrays do. With a String array, the list of String elements that follow the array descriptor are the lengths and addresses of the strings that are stored in the String Storage Area.

For example, type in the following statement and press <RETURN>.

```
CLR:DIM A$(0):A$(0) = "C-128"
```

The following table illustrates how this type of array would appear in memory if you were to do a M 10400 1040A from the MONITOR.

```
>10400 :41 80 0A 00 01 00 01 05 F9 FE
```

The following table describes the meaning of the individual bytes that follow the array descriptor. The first String element begins at \$10408.

STRING ARRAY ELEMENT		
BYTE	00	Length of the string
	01	LSB of the address of the string in RAM BANK 1
	02	MSB of the address of the string in RAM BANK 1

### 1.3.3 Function Variables

The last category of variables is the FuNction (FN) variables. This type of variable can only be defined using the Floating Point format. The main use of this variable is to allow the user to assign a formula or value to be used later when doing mathematical calculations. For instance, type in the following line and RUN it.

```
10 DEF FN AA (AA) = 123
```

The following figure would represent the way that a memory dump of the area where this variable is stored would look like. Remember, variable storage starts in RAM BANK 1 at \$0400.

```
>10400 :C1 41 11 1C 09 04 31 41 41
```



As you can see by the values of the variable name, the Function variable is identified by setting Bit 7 in the first character of the variable name. The following table gives a description of each of the bytes that are used to store this type of variable.

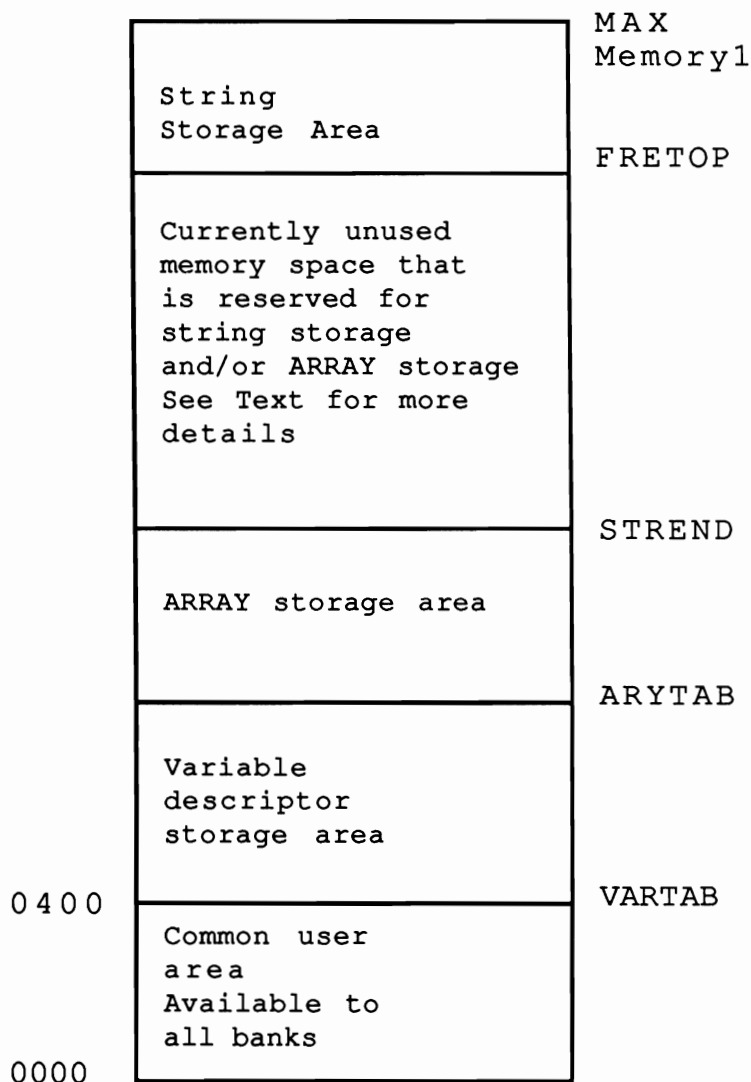
USER DEFINED FUNCTION		
BYTE	00	Variable name with bit seven set
	01	Variable name with bit seven not set
	02	LSB of the address of the expression in RAM BANK 0
	03	MSB of the address of the expression in RAM BANK 0
	04	LSB of the address of the variable descriptor in RAM BANK 1
	05	MSB of the address of the variable descriptor in RAM BANK 1
	06	Contains the first character of the expression

### 1.3.4 Variable Pointers

To make the storing of the different variable types possible, several pointers are used to keep track of where everything is. The following table and diagram will give you the different pointers that are used when storing variables in RAM BANK 1.

LABEL	LOCATION	USE
VARTAB	\$2F, \$30	This location holds the starting address of the variable descriptor storage area. It is normally set to \$0400.
ARYTAB	\$31, \$32	This location holds the starting address of the array storage area.
STREND	\$33, \$34	This location holds the address of the end of the BASIC arrays plus one.
FRETOP	\$35, \$36	This location holds the address of the bottom of the string storage area.
MAXMEM1	\$39, \$40	This location holds the address of the the top of the string storage area. It is always set to \$FF00.

When the 128 is first turned on, VARTAB is set to point to the start of the BASIC variable storage area in RAM BANK 1 and MAXMEM1 is set to the end of the BASIC variable storage area in RAM BANK 1. Normally, VARTAB is set to point to \$0400 and MAXMEM1 is set to point to \$FF00.



When the BASIC CLR routine is called during system initialization, the address that is stored in VARTAB is stored in ARYTAB and STREND and the address that is stored in MAXMEM1 is stored into FRETOP. This resets all of

the pointers to their starting addresses to indicate to the operating system that there are not any variables in the variable storage area.

However, if in the DIRECT mode we were to define a Floating Point variable, ARYTAB and STREND would be incremented by seven in order to make room for the descriptor of the variable that was being defined. These pointers would then point to the next available location for the next variable descriptor or for an array.

If we then define a Floating Point array, ARYTAB would stay where it is, but STREND would be incremented the number of bytes that the array needs to make room for that array. So far, we have defined a Floating Point variable and a Floating Point array. Now let's try something different. Let's define a String variable and see what happens.

If we define a String variable, ARYTAB and STREND will be incremented by seven to make room for the String variable's descriptor. However, this time FRETOP will be decremented the number of bytes needed for the actual string, plus two. The two extra bytes of memory are used to hold the address that points to the length of the string in the variable's descriptor. Consequently, anytime you define a string, not only are ARYTAB and STREND moved but, FRETOP as well. Let's go one step further and define a String array.

When you define a String array, ARYTAB remains where it is and STREND is incremented the number of bytes that are needed for the new array. Also, FRETOP is decremented the number of bytes needed for the string part of the array plus two. Once again, the two extra bytes of memory are used to hold the address that points to the length of the string in the variable's descriptor.

As you can see from the examples above, ARYTAB, STREND, and FRETOP change with each variable that is defined. The pointers are incremented or decremented as necessary when a variable is added or deleted.

If in the process of adding a new variable, the pointers STREND and FRETOP collide, an 'OUT OF MEMORY' error message is generated.

### 1.3.5 A Variable Dump Program

As we have seen, there are several types of variables, and each variable type has its own format when stored in memory. It would therefore be quite useful if a routine was designed to display the value of the variables that are stored in memory. Many times it becomes necessary when writing a BASIC program to see what the variable values are so that an assessment can be made as to whether the program that is being written is operating as it should. Without a program such as the `VARIABLE DUMP` program that is presented here, it would become quite tedious to print all of the variable values to the screen one at a time. However, `VARIABLE DUMP` makes the job of displaying the individual variable values easier by providing a two character command in order to initiate a dump to the screen of all of the variable values that are stored in memory. The only variable type that is not handled by the `VARIABLE DUMP` program is arrays. This is due to the large number of elements that are present in arrays. However, with a little effort on your part, this function could be added to the present `VARIABLE DUMP` program.

#### 1.3.5.1 How To Use The `VARIABLE DUMP` Program

In order to use the `VARIABLE DUMP` program, type in the BASIC generator program listed in the section below. This program when `RUN` will `POKE` the data to memory that is necessary to construct the `VARIABLE DUMP` machine code. After you have typed in the BASIC generator program, save a copy of the program to disk. This will ensure that you will still have a copy of the program should the computer crash or an error in typing was made. This is a good practice to get into, especially with lengthy programs, or programs that contain a large amount of `DATA` statements.

Once the program has been saved to disk, type `RUN` and press `<RETURN>`. The program will indicate that it is running by displaying the message `CURRENT DATA LINE` followed by the line number of the `DATA` statement that is currently being put into memory. If an error in the `DATA` statements should be encountered, an `ERROR IN DATA-LINE` message will be displayed followed by the line number of the `DATA` statement where the error lies.

Once the generator program has finished placing the variable dump machine code into memory, you will be prompted DO YOU WISH TO SAVE AN OBJECT FILE (Y/N). If you want an object file to be saved to disk type <Y>. After the object file has been saved to disk by the BASIC generator program, you will be prompted once again DO YOU WISH TO SAVE AN OBJECT FILE. This feature was included in the program so that a copy of the variable dump program's object file could be saved to as many disks as desired without having to rerun the original BASIC generator program each time you wanted to save a copy of the object file to another disk. It is recommended that you save a copy of the variable dump program's object file to each disk that you use for the development of BASIC programs as you will soon find out that the function that this program serves will be used over and over again when developing a BASIC program.

Once you have finished saving a copy of the object file to the disks you want it on, type <N> in response to the prompt DO YOU WISH TO SAVE AN OBJECT FILE. After you have done this, the message TYPE SYS 3072 TO ACTIVATE will be displayed on the screen and the generator program will end. The SYS address that is displayed is the address that will initialize the VARIABLE DUMP program. Therefore, type <SYS 3072> and press <RETURN>.

Once the program has been initialized, the READY prompt will appear on the screen. To use the new added function, type <@V> and press <RETURN>. Immediately, the program will print the values of all of the variables that are presently stored in memory except the array variables. In order to print the variable dump to the printer, enter the following BASIC statement in the DIRECT mode:

```
OPEN 4,4,7:CMD4:@V
```

Once the VARIABLE DUMP program's object file has been saved to a disk, the next time that you wish to use the program, type the following:

```
1541 Disk Drive: BLOAD"VARIABLE DUMP.O",B15:SYS 3072
```

or

```
1571 Disk Drive: BOOT"VARIABLE DUMP.O",B15
```

### 1.3.5.2 The Variable Dump Generator Program

The following BASIC program is designed to generate the VARIABLE DUMP program as well as allow you to save the object file to disk.

```

10 GOTO140
20 :
30 : VARIABLE DUMP GENERATOR
40 :
50 : COPYRIGHT (C) 1986 BY
60 :
70 :     JIM D. SPRINGER
80 :
90 :         AND
100 :
110 :     DENNIS J. JARVIS
120 :
130 :
140 SCNCLR
150 Y=0:LN=250:AD=DEC("0C00")
160 FORI=ADTOAD+310STEP8:Y=XOR(Y,15):
    VOLY:PRINT"[HOME] CURRENT DATA LINE"LN
170 READD$:POKEI+X,DEC(D$)
180 X=X+1:CK=CK+DEC(D$):IFX<>8THENGOTO170
190 READCK$:IF(CK AND 255)<>DEC(CK$)THENPRINT"ERROR
    IN DATA - LINE"LN:END
200 X=0:CK=0:LN=LN+10:NEXT
210 PRINT"[CURDN][CURDN]DO YOU WISH TO SAVE AN
    OBJECT FILE (Y/N)"
220 GETKEYA$:IFA$<>"Y"AND A$<>"N"THEN220
230 IFA$="Y"THEN SCRATCH"VARIABLE DUMP.O":
    BSAVE"VARIABLE DUMP.O",B0,P(AD) TO P(I):
    VERIFY"*",8: ELSE PRINT"[DOWN]TYPE SYS"AD
    " TO ACTIVATE":END
240 GOTO210
250 DATA A0,0C,A9,10,A2,4C,8E,8D, 6E
260 DATA 03,8D,8E,03,8C,8F,03,60, 9F
270 DATA 8D,03,FF,48,C9,40,D0,12, C2
280 DATA A0,01,20,C9,03,C9,56,D0, 7C
290 DATA 09,C8,20,C9,03,F0,07,4C, 00
300 DATA 6C,79,68,4C,90,03,20,F0, 3C

```

---

```
310 DATA 84,68,A9,00,8D,00,FF,8D, AE
320 DATA 04,D5,A5,2F,A4,30,85,FB, 01
330 DATA 84,FC,C4,32,D0,02,C5,31, 3E
340 DATA 90,09,20,7A,41,4C,37,4D, 44
350 DATA 4C,0C,40,A9,0D,20,D2,FF, 3F
360 DATA A0,00,84,0F,20,2F,0D,AA, 39
370 DATA 29,80,4A,20,1A,0D,20,2F, 89
380 DATA 0D,AA,29,80,20,1A,0D,A5, 4C
390 DATA 0F,F0,2A,C9,80,F0,36,C9, 61
400 DATA 40,F0,52,20,D5,0C,20,F5, 98
410 DATA 0C,A0,00,20,2A,0D,A8,8A, 35
420 DATA 20,3C,79,20,42,8E,A0,00, 65
430 DATA B9,00,01,F0,06,20,D2,FF, A1
440 DATA C8,D0,F5,F0,64,20,DB,0C, E8
450 DATA 20,F5,0C,A5,FB,A4,FC,20, 81
460 DATA 85,7A,4C,8B,0C,20,F5,0C, 03
470 DATA 20,D8,0C,20,F0,0C,A0,00, C0
480 DATA 20,2A,0D,F0,0B,85,24,C8, C3
490 DATA 20,2F,0D,85,25,20,E9,55, 64
500 DATA 20,F0,0C,D0,34,20,DB,0C, 27
510 DATA 20,ED,0C,D0,2F,A9,25,2C, 12
520 DATA A9,24,2C,A9,20,20,D2,FF, B3
530 DATA A9,20,20,D2,FF,A9,3D,20, C0
540 DATA D2,FF,A9,20,2C,A9,2A,2C, C5
550 DATA A9,22,4C,D2,FF,A5,FB,18, A0
560 DATA 69,02,85,FB,90,02,E6,FC, 5F
570 DATA 60,A9,05,2C,A9,07,18,85, 87
580 DATA 0D,A5,FB,65,0D,85,FB,90, 2F
590 DATA 02,E6,FC,A5,FB,A4,FC,4C, 70
600 DATA 42,0C,05,0F,85,0F,8A,29, A9
610 DATA 7F,D0,02,A9,20,20,D2,FF, 0B
620 DATA C8,60,20,2F,0D,AA,C8,A9, 9F
630 DATA FB,20,AB,03,60,FF,C8,60, 50
```

**NOTE:** The BASIC generator program above (as well as the MERLIN source file of the VARIABLE DUMP program) is included with the other programs appearing in this book on the optional disk available from Abacus Software. See the back of this book for order information.

## 1.4 Memory Expansion for the C-128

There are two main types of memory devices used in the 128 to store data, programs, and the routines that allow the 128 to operate. These two types of memory devices are RAM (Random Access Memory) and ROM (Read Only Memory).

RAM is a storage device that can be both written to and read from. However, when power is removed from a RAM chip, all the data that was stored is lost. This is one of the biggest disadvantages of this type of storage device. Therefore, RAM is intended for the temporary storage of data.

One of the advantages of RAM is that it is faster than many other types of storage devices such as a disk drive or cassette drive. In the 128, RAM chips are used for the 128K of memory space, for the screen storage area for the VDC, and for the screen storage area for the VIC.

ROM is a storage device that usually can only be read. The function of a ROM is to supply a permanent storage area for data. This data cannot be erased by merely turning off the power switch.

In the 128, there are several ROMS: Character ROM, BASIC ROM Low, BASIC ROM High, and the KERNAL ROM. Each one contains data that is necessary to enable the 128 to accomplish all of the functions that it was programmed for.

In the following text we will discuss both the RAM and ROM expansion methods in the C-128.



## 1.4.1 RAM Expansion

There are two other ways in which RAM is used with the 128 that are particularly interesting. In fact, they allow the user to be able to expand the memory capabilities of the 128. The two ways to expand the memory capability of the 128 are adding RAM for BANKS 2 and 3 and the RAM Expansion Module (REM).

Although the software exists to support the addition of RAM for BANKS 2 and 3, the 128's motherboard does not provide the hardware necessary to implement these RAM BANKS. When the 128 was produced, the memory addressing scheme was never completed for this type of RAM expansion. Unless someone comes out with an internal modification to the motherboard, the 128 will never have any more than 128K of internal contiguous RAM.

However, the second method of expanding the 128's RAM is a little more usable. This method involves plugging a REM into the expansion port to increase the 128's memory capacity. The two RAM modules currently on the market are the 1700 and 1750 expansion modules that let you expand the 128's memory by 128K and 512K bytes, respectively. The RAM that is added cannot be accessed in the same manner as RAM inside the 128. The module acts more like a RAM disk and is accessed by using the BASIC commands `FETCH`, `STASH`, and `SWAP`.

### 1.4.1.1 The RAM Expansion Unit's Controller

The process that makes all of this possible is known as Direct Memory Access, whereby an external controller is allowed to take over the computer's bus system to affect the data transfer from the external memory to the computer's memory (`FETCH`), from the computer's memory to the external memory (`STASH`), or exchanging the computer memory with the external memory (`SWAP`).

The RAM Expansion Controller (REC) used in the 1700 and 1750 RAM expansion modules is the 8726. It is responsible for handling all the operations that are necessary for transferring data between the expansion module and the 128.

The 8726 has 11 registers that are used to control the DMA process. These registers appear in locations \$DF00 to \$DF0A.

The following table gives the various registers and the description of each that is used to control the DMA process.

Address	Label	Label description
\$DF00	DMAST	DMA status
\$DF01	DMACMD	DMA command
\$DF02	DMAADL	DMA Address Low
\$DF03	DMAADH	DMA Address High
\$DF04	DMALO	DMA LOw
\$DF05	DMAHI	DMA High
\$DF06	DMABNK	DMA BaNK
\$DF07	DMADAL	DMA Data Address Low
\$DF08	DMADAH	DMA Data Address High
\$DF09	DMAIMR	DMA Interrupt Mask Register
\$DF0A	DMAST	DMA Address Control Register

### 1.4.1.1.1 DMA SStatus register (DMAST)

The DMA SStatus register for the REC is located at \$DF00. This is a read only register that is used to store the status of the REC. The definition of each bit of this register is as follows:

- BIT 7 This bit is used to indicate whether there is an interrupt pending. If this bit is set, this indicates that there is an interrupt waiting to be serviced. Note: If Bit 7 is set in the DMAIMR (see Section 1.4.1.1.7), then whenever Bit 6 or 5 of this register is set, then a hardware IRQ is generated, and must be handled by the KERNAL IRQ vector at \$0314/\$0315.
- BIT 6 This bit is used to indicate whether the transfer of a block of data has been completed. If this bit is set, the block has been transferred (see \$DF09).
- BIT 5 This bit when set indicates a verify error (see \$DF09).
- BIT 4 This bit when set indicates that the RAM expansion is a 1750 model (512K) and not the 1700 model (128K).
- BITS 3-0 Contain the version number of the RAM Expansion Controller.

### 1.4.1.1.2 DMA CoMmanD register (DMACMD)

The DMA CoMmanD register for the REC is located at \$DF01. This register is used to store the command to the REC. The definition of each bit of this register is as follows:

- BIT 7 This bit when set can have two meanings. To execute the DMA process when location \$FF00 is accessed again, or to execute the DMA process immediately depending on BIT 4 of this register.
- BIT 6 Not used at this time.
- BIT 5 This bit when set will activate the AUTOLOAD routine in the REC. This option, when activated, will preserve these registers:

DF02, DF03, DF04, DF05, DF07, DF08. This will enable you to perform the same DMA over and over again without having to set these addresses up every time you want to perform a DMA. The main purpose of the AUTOLOAD function is to save you memory.

- BIT 4 When set, this bit will execute the DMA immediately without having to store a value at \$FF00. This bit defaults to 0 .
- BITS 3-2 Are not used at this time. NOTE: BASIC sets this bit each time a DMA is performed, it has no effect!
- BITS 1-0 These bits inform the REC of what type of DMA is to be performed and have the following meaning.

BIT	1	0	TYPE OF OPERATION
	0	0	Transfer from C-128 to REM
	0	1	Transfer from REM to C-128
	1	0	Swap C-128 for REM
	1	1	Verify C-128 with REM

#### 1.4.1.1.3 DMA Address Low/High register (DMAADL,DMAADH)

The DMA ADDRESS LOW/HIGH registers are located at \$DF02 and \$DF03 respectively. These two registers are used to store the LSB and MSB of the starting address where the DMA process is to take place inside the C-128.

#### 1.4.1.1.4 DMA LOW/HIGH register (DMALO,DMAHI)

The DMA LOW/HIGH registers are located at \$DF04 and \$DF05 respectively. These two registers are used to store the LSB and MSB of the starting address where the DMA process is to take place inside the RAM Expansion Module .

#### 1.4.1.1.5 DMA BaNK register (DMABNK)

The DMA BANK register is located at \$DF06. This register contains the BaNK number inside the RAM Expansion Module that the DMA is to take place in. The meaning of each of the bit values is shown below.

BIT	2	1	0	REM BANK	APPLICABLE REM UNIT
	0	0	0	0	128K and 512K
	0	0	1	1	128K and 512K
	0	1	0	2	512K only
	0	1	1	3	512K only
	1	0	0	4	512K only
	1	0	1	5	512K only
	1	1	0	6	512K only
	1	1	1	7	512K only

#### **1.4.1.1.6 The DMA data Address Low/High register (DMAAL,DMAAH)**

The DMA DATA ADDRESS LOW/HIGH registers are located at \$DF07 and \$DF08 respectively. These registers are used to store the LSB and MSB of the number of bytes that the DMA is to process STASH, SWAP, etc.

#### **1.4.1.1.7 The DMA Interrupt Mask Register (DMAIMR)**

The DMA INTERRUPT register is located at \$DF09 and is similar to the ICR register in a CIA. That is, you can select what type of situation or incident you wish to test for, and if this situation does occur, then BIT 7 in the DMA ST is set and the appropriate bit in that register is set to indicate which one of the following occurred.

- BIT 7** This bit is used to enable the interrupt flag. It is the responsibility of the software programmer to test the DAM SStatus register which is located at \$DF00, to see what has actually occurred, and process it accordingly.
- BIT 6** This bit, when set, indicates to the REC that you want it to set BIT 6 in location \$DF00 when the End Of Block occurs.
- BIT 5** This bit, when set, indicates to the REC that you want it to set BIT 5 in location \$DF00 when their is a verify error.
- BITS 4 - 0** Not used at this time.

### 1.4.1.1.8 The DMA Address Control Register

The DMA ADDRESS CONTROL register is located at \$DF0A. This register is used to inform the REC if it is to increment the addresses that are used for the DMA transfer. For instance, this register allows you to increment the address in the REM and not in the C-128 and vice versa.

BIT		DESCRIPTION	REGISTER AFFECTED
0	1		
0	0	Increment both addresses (DEFAULT)	DMA ADL/ADH, DMA LO/HI
0	1	Do not increment REM address	DMA LO/HI
1	0	Do not increment C-128 address	DMA ADL/ADH
1	1	Do not increment either address	

### 1.4.1.2 The Software Nuts and Bolts of DMA

The process that makes the RAM expansion module possible is known as Direct Memory Access, whereby an external controller is allowed to take over the computer's bus system to affect the data transfer from the external memory to the computer's memory, from the computer's memory to the external memory, or exchanging computer memory with external memory.

The BASIC commands that are used with the RAM expansion module are the `FETCH`, `STASH`, and `SWAP` commands. The BASIC syntax for each is identical except for the command that is being used. The table below gives the syntax for these commands:

Command syntax:

```
FETCH, STASH, SWAP #bytes, insta, expsa, expb
```

Where: #bytes = the number of bytes to be fetched, stashed, or swapped

`insta` = internal starting address (address in C-128)  
`expa` = external starting address (address in REM)  
`expb` = external bank number (bank in REM)

The `FETCH` command allows to transfer data from the RAM expansion module into the computer. When this command is used, the machine language routine at `$AA28` is called to handle the execution of the command.

For example, the statement `FETCH 1024,1024,0,0` transfers 1K of data from the RAM expansion module starting at `$0000`, ROM BANK 0, to the computer's screen memory starting at location `$0400`.

The `STASH` command allows you to transfer data from the computer's memory to the RAM expansion module.

For example, the statement `STASH 1024,1024,0,0` transfers 1K of data from the computer's screen memory starting at location `$0400` to the RAM expansion module starting at location `$0000` in BANK 0.

The `SWAP` command allows you to exchange data between the computer's memory and the RAM expansion module.

For example, the statement `SWAP 1024,1024,0,0` exchanges 1K of data from the RAM expansion module starting at location `$0000` in BANK 0 to the computer's screen memory starting at location `$0400`.

The manner in which BASIC accomplishes the DMA is fairly straightforward. It first converts the values that follow the BASIC statement and place them into the proper addresses to set up the REC (see the description of the REC registers above). Then it gets the current BANK number out of `$03D5` which was placed there by the BASIC `BANK` command. After this has been completed, it calls the DMA `CALL` routine out of the `KERNAL JUMP TABLE`, which is located at `$FF50`.

When the DMA `CALL` routine is called, the BASIC BANK number must be in the accumulator with the DMA command in the Y-register. When the DMA `CALL` routine is reached (`$F7A5`), it will take the BASIC BANK number and convert it over to the appropriate memory configuration value (for more information on memory configuration refer to the Appendices) with bit 0 omitted to keep the I/O register intact. Next, the routine will place this value into the Accumulator and the X-register and jump to the EDMA



routine at \$03F0. This is the actual nuts and bolts of the DMA execution, so let's take a closer look at it.

```

*****
*
*                      EDMA
*
*  Execute Direct Memory Access
*
*****

$03F0:  LDX    $FF00
$03F3:  STY    $DF01
$03F6:  STA    $FF00
$03F9:  STX    $FF00
$03FC:  RTS

```

ADDRESS

DESCRIPTION

\$03F0	Preserve the current memory configuration in the X-register
\$03F3	Place DMA command in the DMACMD register for the REC
\$03F6	Set the requested memory configuration, execute the DMA
\$03F9	Restore the old memory configuration

As you can see, the previous routine is fairly straightforward. However, there is one thing to remember. The STA \$FF00 is what actually executed the DMA process, because BASIC does not set BIT 4 in the DMACMD register for the REC. This instruction is why you cannot access any RAM BANK except RAM BANK 0. The REC in the RAM expansion module does not actually use the value that is in \$FF00. It obtains its RAM information from the Memory Managment Unit Ram Configuration Register, or simply MMURCR. To help you past this shortcoming, we have reworked the Execute Direct Memory Access routine for you which will automatically extract the RAM BANK that you want to access and set the MMURCR for you. Then after the DMA process has been completed, the routine restores the MMURCR register back to its previous configuration value.

The main entry point is still \$03F0 to maintain compatibility with the existing operating system which is the top priority.

```

ORG    $03E4                ;Spare memory area just above OLD DMA
                                ;routine
EXECUTE =    %00010000    ;Set the bit to DISABLE the
                                ;$FF00 TRIGGER
MMURCR  =    $D506        ;Memory Management Unit Ram
                                ;Configuration Reg.
DMAST   =    $DF00        ;Direct Memory Access SStatus
                                ;register
DMACMD  =    $DF01        ;DMA CoMmanD register
    
```

```

*****
*
*                               SMMURCR
*
*           Set MMU Ram Configuration Reg
*
* This routine will take the value that is in the
* Accumulator which during a BASIC DMA is the memory
* configuration of the current BASIC BANK number.
*
* Bits 5 thru 0 are discarded because we are only
* concerned with the RAM BANK number that is
* requested.
*
* If your using this routine with machine language,
* just call EDMA with the RAM BANK number in the
* Accumulator (see the section on the MMU for more
* details).
*
* ON ENTRY:  A = Memory configuration
*            X = Not used in this routine
*            Y = Not used in this routine
*
* ON EXIT:   A = New MMURCR value
*            X = Old MMURCR value
*            Y = DMACMD value
*
*****
    
```

```

03E4: AE 06 D5    SMMURCR LDX    MMURCR    ;Get the current RAM
                                                ;Configuration register
03E7: 29 C0                AND    %11000000 ;Only use the RAM BANK
03E9: 0D 06 D5                ORA    MMURCR    ;Configuration
03EC: 8D 06 D5                STA    MMURCR    ;Set VIC RAM BANK number
03EF: 60                RTS                ;Exit the routine
    
```

```

*****
*
*                      EDMA                      *
*
*          Execute Direct Memory Access          *
*
*  This routine takes the memory configuration value *
*  that is in the Accumulator, sets Bit 4 in the value *
*  in preparation to execute the DMA process, and then *
*  stores the value in the DMA command register so that *
*  the DMA will be executed.                      *
*
*  ON ENTRY:  A = Memory configuration            *
*             X = Unused                          *
*             Y = DMACMD VALUE                   *
*
*  ON EXIT:   A = Status of the DMA process      *
*             X = MMURCR configuration           *
*             Y = DMACMD value                   *
*
*****

```

```

03F0: 20 E4 03          JSR   SMMURCR    ;Set the RAM BANK number
                                     ;For the VIC memory
03F3: 98                TYA                    ;Move the DMACMD into the
                                     ;Accumulator
03F4: 09 10            ORA   #EXECUTE   ;Set BIT 4 to Execute the
                                     ;DMA process immediately

```

```

-----*
*
*  NOTE: The DMA process will occur on the next *
*  instruction.                                *
*
*-----*

```

```

03F6: 8D 01 DF          STA   DMACMD    ;Save the DMA command
03F9: 8E 06 D5          STX   MMURCR    ;Restore the MMURCR to its
                                     ;Previous memory config.
03FC: 60                RTS                    ;Exit EDMA

```

The following BASIC program demonstrates the usefulness of the new EDMA routine. Type it in exactly as it is listed and save a copy to the disk. Note: Make sure that your RAM expansion module is plugged in before you try this demo program.

```

0 SCNCLR:REM FAST
5 FORA=0TO1
10 BANKA:FORI=DEC("2000")TODEC("200A"):
    POKEI,A+48:NEXTI,A:A$="BEFORE":GOSUB100
20 SLOW
31 BYTES = 10
32 C128 = DEC("2000")
33 REC = 0
34 BNK = 0
40 BANK0:STASH BYTES,C128,REC,BNK
50 BANK1:SWAP BYTES,C128,REC,BNK
60 BANK0:FETCH BYTES,C128,REC,BNK
70 A$="AFTER":GOSUB140
8- END
100 PRINT:PRINT:PRINT"CURRENT CONTENTS ";A$;" DMA"
101 FORA=0TO1:PRINT:PRINT"RAM BANK";A
110 BANKA:FORI=DEC("2000")TODEC("200A"):
    PRINTCHR$(PEEK(I));:NEXTI,A:RETURN

```

Next, run the BASIC demo program to see that the data that was stored in RAM BANK 0 and RAM BANK 1 before and after the SWAP are the same. This will show you that normally BASIC will only enable you to use RAM BANK 0 for DMA. However, with this short and simple modification to the EDMA routine, DMA access to any RAM BANK is possible.

Next, type in the new EDMA routine with the machine language MONITOR or an assembler and save a copy to disk. To demonstrate that the routine works, ensure that the new EDMA routine is in memory and rerun the BASIC demo program listed above. What you will see is that with the new EDMA routine, the data that was stored in RAM BANK 0 was indeed swapped with the data that was in RAM BANK 1.

### 1.4.1.3 Graphics Demo Program

The following program will give you some idea of what you can do with the extra RAM in the RAM expansion module. The program first draws 16 hi-res screens and saves them in the REM. The program will then FETCH the different screens one at a time to animate the figures on the screen. The 16 screens use 128K of the RAM in the RAM expansion module.

Type in the program as it is listed below and save a copy to disk. Next, run the program. It will take a couple of minutes to set up the screens. Feel free to play around with the graphics commands to generate different effects.

```

1 GOSUB28:SCNCLR:PRINT V"K OF RAM IS INSTALLED":SLEEP2
2 SZ=16:FORH=0TO8:N=INT(RND(1)*4)+1:X1(H)=N:NEXT
3 GRAPHIC3,1
4 FORJ=0TO8:X=X+15:X(J)=X:NEXT
5 FORJ=0TO8:Y(J)=INT(RND(1)*20)+1:NEXT
6 FORL=0TOSZ-1:SCNCLR
7 FORI=0TO8:COLOR2,I+1
8 FAST:Y1=180-(3*Y(I))
9 CIRCLE2,X(I),180,5,5,,,,90:PAINT2,X(I),180
10 CIRCLE2,X(I),Y1,5,5,,,,90:PAINT2,X(I),Y1
11 BOX2,X(I)-5,Y1,X(I)+5,180,,1
12 CIRCLE0,X(I),Y1,5,5,90,270,,90
13 DRAW0,X(I),Y1+5 TO X(I),180+Y1-5
14 NEXTI
15 FORK=0TO8:Y(K)=Y(K)+X1(K):IFY(K)<=1 OR Y(K)>=42THENX1(K)=0
16 NEXTK
17 GOSUB32
18 SLOW:STASH V,IA,AD,BN
19 NEXTL
20 GRAPHIC3
21 FORL=0TO15:GOSUB32
22 FETCH V,IA,AD,BN
23 NEXT
24 FORL=15TO0STEP-1:GOSUB32
25 FETCH V,IA,AD,BN
26 NEXT
27 RUN21
28 REM ** ENSURE THAT RAM EXP. IS PRESENT **
29 A=DEC("DF06"):POKEA,255:IFPEEK(A)<>255THENPRINT"RAM NOT INSTALLED
!!":END
30 POKEA,0
31 A=DEC("DF00"):IF PEEK(A)AND16 THEN V=512:ELSE V=128:RETURN
32 V=8192:IA=V:MX=65536:BN=- (L>=8):AD=L*V-MX*BN:RETURN

```

## 1.4.2 ROM Expansion

There are two other ways in which ROMs are used in the C-128 that are particularly interesting. In fact, they allow users to use ROMs for their own purposes. The two other ways in which ROMs are used in the 128 are for the Internal and External function ROMs.

The Internal ROM is presently an empty socket that was designed to be used in the future. This ROM socket allows the use of up to 32K of memory area that is divided in two parts, \$8000 - \$BFFF and \$C000 - \$FFFF. This area is accessible in BASIC with some configurations provided by the BASIC BANK command.

The External ROM, on the other hand, is quite a bit more accessible and usable. One of the most widely used applications of the External function ROM is the game cartridge. The cartridge is inserted into the expansion port on the rear panel of the C-128. This port also allows the use of up to 32K of memory to be used by the cartridge in two parts: \$8000 - \$BFFF and \$C000 - \$CFFF.

### 1.4.2.1 ROM Software Requirements

In order to use the Internal/External ROM(s), certain requirements have been placed upon the software developer of the cartridge. Even though these requirements are simple to meet, the operating system is quite picky about them. When the computer is first turned on, a routine in ROM located at \$E26B will check all four possible locations where the ROM (s) could be located. These locations are shown below in their priority of testing.

BASIC BANK	ADDRESS	PHYSICAL ADDRESS TABLE (PAT)
8	\$8000	\$0AC4
8	\$C000	\$0AC3
4	\$8000	\$0AC2
4	\$C000	\$0AC1

The first thing that the routine will check for is the characters 'cbm' (not <SHIFT>ed). If a character in the ROM does not match, then the next ROM is tested. If there are no ROMs found with these characters, then the system will place zeros in location \$0AC1 thru \$0AC4 which are used by another routine which we will discuss in just a moment.

If the characters 'cbm' are found to be in a ROM, then the routine will check to see if the ROM is an autostart ROM or not. If the value preceding the characters 'cbm' is a \$01, then the routine will jump to the COLDSTART address of the ROM.

This may start to sound complicated, but it's not. The format that the software in the C-128 requires is as follows. NOTE: The X could either be an 8 or a C depending which ROM address you are creating.

ADDRESS	DESCRIPTION
\$X000 - \$X002	COLDSTART address to JMP to or JSR to.
\$X003 - \$X005	SPARE ADDRESS (Old 64 warm start address.)
\$X006	Auto start flag (1=auto start see text)
\$X007 - \$X009	Lower case text 'cbm'

### 1.4.2.2 Auto-starting ROMs

The KERNAL routine that checks to see if you want a ROM to AUTO START checks byte 6 of the ROM. If it contains a value of \$01, then the routine terminates and JSRs to the ROM at the starting address. For example, if you had a game ROM inserted at \$8000 in the external ROM and you wanted it to autostart, the ROM would be programmed as follows:

ADDRESS	SOURCE CODE	DESCRIPTION
\$8000 - \$8002	JSR \$800A	Display start up screen.
\$8003 - \$8005	JMP \$9023	Enter the main loop of the program.
\$8006	.BYTE \$01	Flag to auto start rom
\$8007 - \$8009	TXT 'cbm'	Flag to indicate a ROM is inserted here
\$800A		From this point on it's up to you

### 1.4.2.3 Non-Auto-Starting ROMs

Let's say, for example, you have developed a series of routines for performing structural analysis which you want to market and you wish to give the user the capability of using these routines in their own programs by just simply SYSing to them. One way of accomplishing this would be to create the ROM as explained above and place, for instance, your trademark character in place of the autostart byte, byte 6 in the ROM.

For example, let's say you used the character '@', the routine at \$E26B would find the ROM cartridge installed and log it in the Physical Address Table (PAT). The reason behind the PAT table is very simple. Once a ROM has been found when the system is first turned on or reset, the routine at \$E26B will 'log' (see the table at 1.4.2.1) it in its PAT table address. This table in turn will be used the next time the computer is reset.

Upon powerup or reset, one of the first things that the computer does after some initializations is call another ROM routine called PHOENIX which will check for a boot sector on track 1 sector 0 on the disk. The very first thing the PHOENIX (\$F867) routine will do is check the PAT table to see if any ROMs have been logged in and if so, it will JSR to the ROM in the same manner as an AUTOSTART.



## **Chapter Two**

# **Wedging into BASIC**



## Wedging Into BASIC

BASIC 7.0 on the C-128 is one of the most powerful languages ever implemented on any of the Commodore computers. However, even with its powerful BASIC and all of the new commands, there are probably still some commands that you would like to have implemented. For instance, maybe some commands like `KEYOFF` and `KEYON` would be a nice addition. These two programs enable you to turn off the function keys and enable you to read and check each function key in your program, then restore them at any time to the original C-128 configuration. In this section we are going to create the `KEYOFF` and `KEYON` commands in a wedge example.

Adding these commands and others at first may seem to be difficult to do. With the C-128 as is with the C-64, commands could be added by using the various `VECTORS` or the `SYS` command (limited on the C-128). However, the most popular way of wedging into BASIC is to wedge into the `CHRGET` (`CHaRacter GET`) routine.

### 2.1 The `CHRGET/CHRGOT` Routine

When the computer is first turned on or after the reset button is pressed, BASIC copies several routines from ROM (Read Only Memory) into RAM (Random Access Memory) to be used by the BASIC and KERNAL routines. One of these routines is called `CHRGET`. It is copied from ROM at \$4279 - \$4297 into RAM at \$0380 to \$039E.

BASIC uses the `CHRGET` routine to get a character from the BASIC program memory or the input buffer. The input buffer is used to store characters from the keyboard until some action is taken to use these characters for other routines. Before we go any further, let's take a look at the `CHRGET` routine.

On the next page is a disassembly of the `CHRGET` routine as it appears in memory in the C-128.

```

$0380: INC $3D
$0382: BNE $0386
$0384: INC $3E
$0386: STA $FF01
$0389: LDY #$00
$038B: LDA ($3D),Y
$038D: STA $FF03
$0390: CMP #$3A
$0392: BCS $039E
$0394: CMP #$20
$0396: BEQ $0380
$0398: SEC
$0399: SBC #$30
$039B: SEC
$039C: SBC #$D0
$039E: RTS

```

**NOTE:** Refer to the disassembly of the CHRGET routine listed above when reading the step-by-step description of the routine.

LOCATION	DESCRIPTION
\$0380 - \$0384	;The TeXT PoiNteR (TXTPTR) is incremented by one ;to point it to the next character in memory. ;NOTE: The TeXT PoiNteR is stored in locations \$3D/\$3E in ;LSB/MSB format and is used to hold the address of the ;next character in memory.
\$0386	;This instruction is used to ensure that the character is ;taken from RAM BANK 0 which is used for BASIC program ;storage. If the CHRGET routine is entered here, it is ;called CHRGOT (CHaRacter GOT).CHRGOT accomplishes the ;same job as CHRGET except the TeXT PoiNteR is not ;incremented.
\$0389 - \$038B	;Here the character is taken from memory and stored into ;the Accumulator. If the CHRGET routine is entered here, ;it is called QNUM (Query for NUMber).
\$038D	;This instruction is used to ensure that the memory is ;reconfigured to have all ROMs in place and RAM BANK 0 ;enabled which is the same thing as a BASIC BANK 14.

\$0390 ;If the character is a colon or greater, the routine  
;exits with the carry flag set. If the CHRGET routine is  
;entered at this point, it is called QNUM which stands  
;for Query NUMber.

\$0394 - \$0396 ;If character is a space, then the routine skips it and  
;gets the next character.

\$0398 - \$039E ;The character is checked to see if it is an ASCII digit  
;0-9 by first subtracting \$30 and then \$D0. If the char-  
;acter is a digit, the carry flag will be cleared. If the  
;character is not a digit, then the carry flag is set.

## 2.2 Wedging into the CHRGET Routine

One of the reasons that you might want to wedge into the CHRGET routine is to implement a new command. However, before you can implement a new command, you must modify the CHRGET routine so that it checks for the new command.

If you were to put a JMP or JSR to the routine for the new command at the beginning of CHRGET, the new routine would be executed each time that CHRGET was called but, not when CHRGOT was called. In addition, the new routine would have to increment TXTPTR and read in the next character itself. Consequently, in order to accomplish what you set out to do with the minimum amount of work, you will have to put the JMP to the new routine at \$038D, where the new routine will be executed whenever CHRGET or CHRGOT is called and the current character that the operating system is about to process is already in the Accumulator.

**NOTE:** Whenever you make any changes to CHRGET or CHRGOT, remember that you must perform the instructions that you replaced with the JSR or JMP. If you do not perform the instructions that were deleted, CHRGET will not function properly and it could cause the system to crash.

To illustrate how to wedge into CHRGET to implement a new command, a routine was written to implement the KEYON and KEYOFF commands. As you will find by reading the ROM listings for the KEY command, even though these commands are tokenized by the BASIC operating system, they were not implemented when BASIC 7.0 was written.

The **KEYOFF** command redefines the eight function keys to the **CHR\$** values that are used for the function keys in the C-64. The **KEYON** command returns the definitions of the function keys to the C-128 **KEY** definitions. These commands may also be used in the **PROGRAM (RUN)** mode as well as the **DIRECT** mode.

The starting address for the **KEYON/KEYOFF** routine is **\$0C00**. However, the machine code from **\$0C00** to **\$0C09** puts the **JMP \$0C10** into **\$038D**. The actual **KEYON/KEYOFF** command routine starts at **\$0C10**. Therefore, we will place a **JMP \$0C10** instruction at location **\$038D** as shown below in the partial disassembly of the modified **CHRGET** routine.

<u>BEFORE THE MODIFICATION</u>	<u>AFTER THE MODIFICATION</u>
<u>\$0389: LDY #\$00</u>	<u>\$0389: LDY #\$00</u>
<u>\$038B: LDA (\$3D),Y</u>	<u>\$038B: LDA (\$3D),Y</u>
<u>\$038D: STA \$FF03</u>	<u>\$038D: JMP \$0C10</u>
<u>\$0390: CMP #\$3A</u>	<u>\$0390: CMP #\$3A</u>

The following disassembly and description are for the new routine that implements the **KEYON/KEYOFF** commands. Once you have typed in either the **BASIC** generator program or the source code using a machine language **MONITOR**, you must **SYS DEC ("0C00")** to initialize the routine.

Once the routine is initialized, the only way that you can interrupt the operation of the routine is to either shut the 128 off, press the **RESET** button, or go into the C-64 mode. To use the new commands, type **KEYOFF** to define the function keys to the C-64 definitions and type **KEYON** to return the function keys to the C-128 definitions.

```

ORG      =      $0C00
CHRGET   =      $0380      ;CHaRacter GET routine
CHRGOT   =      $0390      ;CHaRacter GOT routine
INDTXT   =      $03C9      ;INDirect load from TeXT
                        ;pointer
PKYBUF   =      $1000      ;Programmable KeY BUFFER
PKYDEF   =      $100A      ;Programable KeY DEFINitions
    
```

```

*-----*
*
* Here is where we WEDGE into the CHRGET routine by
* inserting a JMP instruction to the new routine at
* $038D.
*
*-----*
    
```

```

0C00: A0 0C          LDY    #>WEDGE    ;Get the MSB and
0C02: A9 10          LDA    #<WEDGE    ;The LSB of the address of
                        ;The new routine
0C04: A2 4C          LDX    #$4C       ;Opcode for JMP
0C06: 8E 8D 03      STX    $038D      ;Save the JuMP
0C09: 8D 8E 03      STA    $038E      ;To the new routine
0C0C: 8C 8F 03      STY    $038F      ;In CHRGET
0C0F: 60            RTS           ;And exit to activate it
    
```

```

*-----*
*
* This section checks to see if the command KEYON or
* KEYOFF was used.
*
*-----*
    
```

```

0C10: C9 F9          WEDGE  CMP    #$F9    ;Is it the token for 'KEY'?
0C12: D0 1F          BNE    NOTKEY    ;If not, then exit
0C14: A0 00          LDY    #$00      ;If it is, then set up an
                        ;Index to next character
0C16: C8            SPACE  INY           ;Increment index register
0C17: 20 C9 03      JSR    INDTXT    ;Get next character after
                        ;The token for 'KEY'
0C1A: C9 91          CMP    #$91      ;Is next character the
                        ;Token for 'ON' ?
0C1C: F0 36          BEQ    KEYON     ;Yes, then turn the
                        ;function key definition on
0C1E: C9 FE          CMP    #$FE      ;If not, is it the value
                        ;For a dual token ?
0C20: F0 07          BEQ    CHKDUAL   ;Yes (first token for OFF)
                        ;then check the next token
    
```

```

0C22: C9 20          CMP    #$20      ;Is there a space between
                                ;Key and ON or OFF?
0C24: F0 F0          BEQ    SPACE     ;If so, then skip it and get
                                ;The next character
    
```

```

*-----*
*
*   At this point, the TOKEN for KEY was found, but the
*   next character after it was not the TOKEN for
*   either of the new commands. So a JuMP is made to
*   execute the old KEY command.
*
*-----*
    
```

```

0C26: 4C 31 0C      JMP    OLDKEY
    
```

```

*-----*
*
*           CHECK FOR THE DUAL TOKEN FOR KEYOFF
*
*   This section of machine code will check to see if
*   the token after $FE is $24. If it is, then the dual
*   command token is for the new command KEY OFF.
*
*-----*
    
```

```

0C29: C8            CHKDUAL  INY      ;Move to character after
                                ;The dual token
0C2A: 20 C9 03      JSR    INDTXT    ;Get the character
0C2D: C9 24          CMP    #$24      ;Is it the token for 'OFF'?
0C2F: F0 08          BEQ    KEYOFF    ;Yes, then turn the
                                ;Function key definition off
0C31: A9 F9          OLDKEY  LDA    #$F9 ;Get the token for 'KEY'
0C33: 8D 03 FF      NOTKEY  STA    $FF03 ;Enable BANK 14 for CHRGET
0C36: 4C 90 03      JMP    CHRGOT    ;Return to normal routine
    
```

```

*-----*
*
*           THE KEYOFF ROUTINE
*
*   This routine defines the eight function keys to the
*   values that are used in the C-64.
*
*-----*
    
```

```

0C39: 20 80 03      KEYOFF  JSR    CHRGET ;Update TXTPTR
    
```



```

*-----*
*
* This section of machine code changes the eight
* function key definitions to the values that are used
* on the C-64.
*
* The following table give the definitions for the
* eight function keys after the KEYOFF command has
* been executed.
*
*
* KEYOFF FUNCTION KEY DEFINITIONS
* -----
*
* F1 = CHR$(133)      F2 = CHR$(137)
* F3 = CHR$(134)      F4 = CHR$(138)
* F5 = CHR$(135)      F6 = CHR$(139)
* F7 = CHR$(136)      F8 = CHR$(140)
*
* The following listed keys are also redefined by the
* KEYOFF command.
*
* SHIFT/RUN-STOP = CHR$(131)
* HELP KEY = CHR$(132)
*-----*
    
```

```

0C3C: A0 09          LDY    #$09          ;Set up an index to the
                                ;Number of definitions
0C3E: B9 DD C6      NEXTKEY LDA    $C6DD,Y      ;Get the 64 key
0C41: 99 0A 10          STA    PKYDEF,Y      ;configuration place it in
                                ;the key definition table
0C44: 88            DEY
0C45: 10 F7         BPL    NEXTKEY      ;Move to next definition
                                ;Continue until all 10 keys
                                ;Are done
    
```

```

*-----*
*
* Here we reset the definition lengths of the eight
* function keys to 1.
*-----*
    
```

```

0C47: A0 09          LDY    #$09          ;Index to number of keys -1
0C49: A9 01          LDA    #$01          ;Length of key definition
0C4B: 99 00 10      NEXTKEY1 STA    PKYBUF,Y      ;Set the key definition
                                ;Length to 1
0C4E: 88            DEY          ;Move to the next key
    
```

```

OC4F: 10 FA          BPL  NEXTKEY1  ;Continue until all 8 keys
                                ;Are done
OC51: 4C 5F 0C      JMP  EXIT      ;And exit

*-----*
*
*              THE KEYON ROUTINE
*
* This routine will restore the eight function keys to
* their original definitions.
*
*-----*

OC54: A0 76          KEYON  LDY  #$76      ;Set up an index to the
                                ;default key definitions
OC56: B9 A8 CE      NEXTKEY2 LDA  $CEA8,Y  ;Get a chracter from the
                                ;Original definitions
OC59: 99 00 10          STA  PKYBUF,Y  ;Save them
OC5C: 88              DEY                ;Move to the next character
OC5D: 10 F7          BPL  NEXTKEY2      ;Continue until all of the
                                ;Keys have been restored
OC5F: 20 80 03      EXIT   JSR  CHRGET    ;Update TXTPTR
OC62: 4C 80 03      JMP  CHRGET        ;And exit to the normal
                                ;routine

```

If you choose to use the BASIC generator program for the KEYON/KEYOFF routine, type the program exactly as it is listed here.

```

10 REM KEYON/KEYOFF, BASIC 7.0 INTERNALS
20 LN=160:A=DEC("0C00")
30 FORI=ATO+102STEP8
40 FORJ=0TO7:READD$
50 V=DEC(D$)
60 POKEI+J,V:CK=CK+V
70 NEXTJ
80 READCK$:IF(CKAND255)<>DEC(CK$)THEN
  PRINT"ERROR IN LINE "LN:GOTO150
90 LN=LN+10
100 CK=0:NEXTI
110 PRINT"DO YOU WANT TO SAVE THIS PROGRAM ?"
120 GETKEYA$:IFA$="N"THEN 140
130 BSAVE "KEYON/KEYOFF.O",B0,P3072 TO P3175
140 PRINT"TYPE 'SYS 3072' TO ACTIVATE"
150 END

```

```
160 DATA0,0C,A9,10,A2,4C,8E,8D, 6E
170 DATA03,8D,8E,03,8C,8F,03,60, 9F
180 DATAC9,F9,D0,1F,A0,00,C8,20, 39
190 DATAC9,03,C9,91,F0,36,C9,FE, 13
200 DATAF0,07,C9,20,F0,F0,4C,31, 3D
210 DATA0C,C8,20,C9,03,C9,24,F0, 9D
220 DATA08,A9,F9,8D,03,FF,4C,90, 15
230 DATA03,20,80,03,A0,09,B9,DD, E5
240 DATAC6,99,0A,10,88,10,F7,A0, A8
250 DATA09,A9,01,99,00,10,88,10, F4
260 DATAFA,4C,5F,0C,A0,76,B9,A8, 28
270 DATA0E,99,00,10,88,10,F7,20, 26
280 DATA80,03,4C,80,03,6E,73,2D, 60
```

After you have finished typing in the generator program, save a copy of the program to disk just in case an error causes the loss of the program in memory. Next, type `RUN` and press `<RETURN>`. After a few seconds, the program will ask if you wish to save the object code of the program. Type `<Y>` to save, `<N>` to run without saving. Once the `KEYON/KEYOFF` program has been saved to the disk, type `SYS 3072` or `SYS DEC("0C00")` and press `<RETURN>` to activate the new command routines. You can also attempt to use the BASIC command:

```
BOOT"KEYON/KEYOFF",B0
```

It might even work, but there is a bug in the `BOOT` command which causes the system to break if you are using a disk drive that doesn't support the fast serial bus—see the `BOOT` routine for more details.

If an error is found in the `DATA` statements, an error message will be printed indicating which `DATA` statement the error is in. Correct the error in the `DATA` statement, save a copy of the program to disk, and `RUN` the generator program again to save the `KEYON/KEYOFF` routine to disk.

To test the new commands, enter `KEYOFF` and `<RETURN>`. Then type `KEY` and press `<RETURN>` to display the current function key definitions.

What you should see is that the function keys are now defined to the same values as the C-64 function keys. To return the function keys to the initial C-128 definitions, type `KEYON` and press `<RETURN>`. To confirm that the function keys have been redefined to their original C-128 values, type `KEY` and press `<RETURN>` to display the current function key definitions.

---

The KEYON/KEYOFF routine does not save the current function key definitions before it redefines them to the C-64 values. Consequently, if you redefined the function keys and then did a KEYOFF and then a KEYON, the function keys would be returned to their original C-128 definitions and not to the definitions you had before you executed the KEYOFF command.

If you want to use the new commands in your BASIC program, simply use the keyword as you would any other keyword that the C-128 uses. For example, this little program will display the function key definitions for the keys in both the KEYON and KEYOFF states.

```
10 SCNCLR
20 KEYOFF
30 PRINT "C-64 DEFINITIONS"
40 PRINT "-----"
50 KEY
60 KEYON
70 PRINT:PRINT
80 PRINT "C-128 DEFINITIONS"
90 PRINT "-----"
100 KEY
```

The following BASIC program will demonstrate the use of the KEYOFF and KEYON commands.

```
10 KEYOFF
20 DO
30 GETKEYA$
40 B=( (A<137)+85-(A-133) )
50 A=ASC(A$)
60 IFA>136 THEN B=B+(7+(A>136) )
70 PRINT"THAT WAS THE F"CHR$(A-B) " KEY"
80 LOOP
```

As you have seen, it is quite easy to wedge into the CHRGET routine to implement new commands and new routines. The number of routines that could be written to perform a function similar to the KEYON/KEYOFF routine are only limited by the ambition and imagination of the programmer.

## **Chapter Three**

# **Graphics**



## Graphics

BASIC 7.0 offers a total of 10 graphics commands to greatly simplify the programming of High resolution (Hi-res) graphics on the Commodore 128. You can now draw lines, boxes, circles, etc., with just a few commands, as compared to the lengthy C-64 programs requiring PEEKs and POKES. But even with all of its specialized commands, BASIC 7.0 cannot cover all of the capabilities of the C-128. We'll cover the different aspects and capabilities of the C-128 graphics system in this chapter .

### 3.1 Managing Memory

The Commodore 128 has a total of 128K of RAM, which is divided into two RAM banks of 64K each. The reason for this is because the 8502 microprocessor can only access 64K of RAM at any one time. Both the 8502 and the VIC chip can each address a different 64K RAM bank.

The VIC chip RAM bank is selected by the RAM configuration register at location \$D506 (Bits 6 and 7). Bit 7 of this location is currently unused. To select RAM bank 1 for the VIC chip, bit 6 must be set to 1 and for RAM bank 0, bit 6 must be set to 0.

The following BASIC and machine language instructions accomplish this:

To select RAM bank 0 :

```
BANK 15:POKE DEC("D506"),PEEK (DEC("D506")) AND 191
```

In machine language :

```
LDA #$00 ;Set the memory configuration
STA $FF00 ;To BANK 15 to enable I/O
LDA $D506 ;Get the current value in $D506
AND #$BF ;Clear bit 6 to select RAM bank 0
STA $D506 ;And store it as the new value
```

To select RAM bank 1:

```
BANK 15:POKE DEC("D506"),PEEK (DEC("D506")) OR 64
```

In machine language:

```
LDA #$00 ;Set the memory configuration
STA $FF00 ;To BANK 15 to enable I/O
LDA $D506 ;Get the current value in $D506
ORA #$40 ;Set bit 6 to select RAM bank 1
STA $D506 ;Store it as the new value
```

Even though the 8502 can access 64K of RAM at a time, the VIC chip can only access 16K of memory at a time. Each of these 16K blocks of RAM are called VIDEO BANKS. So with 128K of RAM, that gives us 4 video banks in each 64K RAM bank for a total of 8 video banks.

Two types of memory must be contained within a video bank; screen and character memory. You can select the video bank that the VIC chip will access by modifying bits 0 and 1 in location \$DD00.

In BASIC :

```
POKE DEC("DD00"),(PEEK(DEC("DD00"))AND 252) OR X
```

In machine language:

```
LDA $DD00 ;Get the current value in $DD00
AND #$FC ;Clear bits 1 and 0
ORA #$X ;Set the selected configuration
STA $DD00 ;And store the value back
```

X = Value from the table below

NOTE: To be able to read or write to RAM in the same area as ROM, you must select the memory configuration that contains RAM in these areas.

VIDEO BANK	MEMORY RANGE	BITS	DEC. VALUE (X)
0	\$0000-\$3FFF	1 1	3 DEFAULT
1	\$4000-\$7FFF	1 0	2
2	\$8000-\$BFFF	0 1	1
3	\$C000-\$FFFF	0 0	0



### 3.1.1 Moving Screen and Character Memory

When changing to a different video bank, you must move the location of screen and character memory. The C-128 uses the same register for moving screen and character memory as the C-64 (\$D018), but the C-128's Interrupt driven screen editor uses the indirect registers at locations \$0A2C (2604) and \$0A2D (2605) to select the value in \$D018. If you try to modify the register at \$D018, the Interrupt routine will automatically reset it to the values that are at locations \$0A2C and \$0A2D.

Consequently, if you want to program the VIC chip directly, you must disable the Interrupt routine by storing a 255 in location \$D8 (216). Then you can program the VIC chip just as you would on the C-64.

NOTE: The register at location \$D8 (216) is automatically reset and the Interrupt driven screen editor is restarted when any BASIC graphics commands are executed.

The values below are valid only if the Interrupt driven screen editor is on.

Location	\$D8 (216)	Bit 7	Multicolor Bitmap mode
		Bit 6	Split screen
		Bit 5	Standard Bitmap mode
Location	\$0A2C(2604)	Used for moving character and screen memory in text mode.	
		Bits 7 - 4	control screen memory
		Bits 3 - 1	control character memory
Location	\$0A2D(2605)	Used for moving screen memory and the location of the Hi-res screen in the Bitmap mode	
		Bits 7 - 4	control screen memory
		Bit 3	controls location of the Bitmap

The BASIC and machine code instructions below move the screen memory.

In BASIC :

```
POKE V, (PEEK(V) AND 15) OR X
```

In machine language :

```
LDA $V ;Get the current screen location
AND #$0F ;Drop the old screen location
ORA #$ X ;Replace it with the new location
STA $V ;And store it back
```

V = \$0A2C for text mode and \$0A2D for bitmap mode or \$D018 if the Interrupt driven screen editor is disabled

X = Value from the table below:

Location of Screen Memory

MEMORY RANGE	BITS	7	6	5	4	DECIMAL VALUE (X)
\$0000-\$03FF		0	0	0	0	0
\$0400-\$07FF		0	0	0	1	16    DEFAULT
\$0800-\$0BFF		0	0	1	0	32
\$0C00-\$0FFF		0	0	1	1	48
\$1000-\$13FF		0	1	0	0	64
\$1400-\$17FF		0	1	0	1	80
\$1800-\$1BFF		0	1	1	0	96
\$1C00-\$1FFF		0	1	1	1	112
\$2000-\$23FF		1	0	0	0	128
\$2400-\$27FF		1	0	0	1	144
\$2800-\$2BFF		1	0	1	0	160
\$2C00-\$2FFF		1	0	1	1	176
\$3000-\$33FF		1	1	0	0	192
\$3400-\$37FF		1	1	0	1	208
\$3800-\$3BFF		1	1	1	0	224
\$3C00-\$3FFF		1	1	1	1	240

NOTE: The value that results when you multiply the video bank number by the value \$4000 (VB# \* \$4000) must be added to each of the starting addresses.

For example, to generate the offset value for the first starting address, you would calculate the following:

$$(VB\# * \$4000) = 2 * \$4000 = \$8000$$

Next take the offset value and add it to the first address in the table to find the starting address of the RAM bank that is desired (SA + offset).

In our example, this would be:

$$(\text{SA} + \text{offset}) = \$0000 + \$8000 = \$8000$$

In the standard character mode, the character set is located in ROM starting at location \$D000. In the C-128, you have the ability to have the standard character set appear in any one of the 8 video banks by clearing the CHAREN bit (bit 2) in location 1. For example, this allows the VIC chip to 'see' the ROM character set in any video bank. If you wish to use your own character set, bit 2 of location 1 must be set to 1. Bits 3-1 in location \$0A2C and \$D018 are used to move the location of the character set. Bit 0 is not used.

The following BASIC and machine language instructions allow you to move character memory.

In BASIC :

```
POKE V, (PEEK(V) AND 240) OR X
```

machine language :

```
LDA $V ;Get the current character location
AND #$F0 ;Drop the old character location
ORA #$X ;Replace it with the new location
STA $V ;And store it back
```

V = \$0A2C for text mode or \$D018 if the Interrupt driven screen editor is disabled

X = Value from the table on the next page:

Location of Character Memory

MEMORY RANGE	BITS 3	2	1	DECIMAL VALUE (X)
\$0000-\$07FF	0	0	0	0
\$0800-\$0FFF	0	0	1	2
\$1000-\$17FF	0	1	0	4
\$1800-\$1FFF	0	1	1	6
\$2000-\$27FF	1	0	0	8
\$2800-\$2FFF	1	0	1	10
\$3000-\$37FF	1	1	0	12
\$3800-\$3FFF	1	1	1	14

NOTE: The value that results when you multiply the video bank number by the value \$4000 ( $VB\# * \$4000$ ) must be added to each of the starting addresses.

For example, to generate the offset value for the first starting address, you would calculate the following:

$$(VB\# * \$4000) = 2 * \$4000 = \$8000$$

Next take the offset value and add it to the first address in the table to find the starting address of the RAM bank that is desired ( $SA + \text{offset}$ ).

In our example, this would be:

$$(SA + \text{offset}) = \$0000 + \$8000 = \$8000$$

### 3.1.2 Color Memory

In the C-128, the color memory occupies the range \$D800 (55296) to \$DBE7 (56295). This corresponds directly to the screen memory in the standard character mode. For example, with the text screen at location \$0400-\$07FF, location \$D800 controls the color of location \$0400, \$D801 controls the color of location \$0401, etc.

The C-128 has 2 color RAM banks that are controlled from bits 0 and 1 of location \$0001. These 2 bits are labeled LORAM and HIRAM respectively. LORAM selects the color RAM bank that the 8502 microprocessor is currently accessing and HIRAM selects the color RAM bank that the VIC chip is accessing. This allows you to change the color of the text screen or the multicolor bitmap instantly. It also allows you to change the color of one screen while viewing another. Normally, the operating system will only allow you to use color RAM bank 0. Every time a graphics command is executed, the operating system automatically selects color RAM bank 0. However, there is a way around this.

First, you must disable the Interrupt driven screen editor by storing a 255 (\$FF) to location \$D8 (216) so you can modify location 1 without the editor automatically resetting it. Then set bit 0 of the DDR register in location 0 to an input (0). This will not allow the operating system to modify the LORAM bit in location 1. The following program illustrates this process.

```

10 COLOR 3,1 :REM SET MULTICOLOR 2 TO BLACK
20 GRAPHIC 3,1 :REM ENABLE MULTICOLOR BITMAP AND CLEAR IT
30 BOX 3,90,90,60,60,45,1 :REM DRAW A BLACK BOX ON THE BITMAP
40 POKE 216,255 :REM DISABLE INTERRUPT DRIVEN SCREEN EDITOR
50 POKE 1,(PEEK(1) AND 252) OR 3 :REM ENABLE COLOR RAM 1 FOR VIC & 8502
60 POKE 0,PEEK(0) AND 252 :REM SET DDR TO INPUT
70 COLOR 3,7:COLOR 2,2:
    REM SET MULTICOLOR 2 TO BLUE,MULTICOLOR 1 TO WHITE
75 CIRCLE 2,75,75,23,23 :REM DRAW A WHITE CIRCLE
80 BOX 3,90,90,60,60,45,1 :REM DRAW A BLUE BOX ON THE BITMAP
90 POKE 0,(PEEK(0) AND 252) OR 3 :REM RESET DDR TO OUTPUT
100 POKE 216,255 :REM DISABLE INTERRUPT DRIVEN SCREEN EDITOR
110 DO :REM START OF LOOP
120 POKE 1,PEEK(1) AND 252 :REM ENABLE COLOR
130 SLEEP 1 :REM DELAY
140 POKE 1,(PEEK(1) AND 252) OR 3 :REM ENABLE COLOR RAM 1
150 SLEEP 1 :REM DELAY
160 LOOP :REM CONTINUE

```

This program first draws a black box on the screen whose color will be stored in color RAM 0. Then the LORAM and HIRAM bits are set to 1 to enable the 8502 to write to color RAM 1. This also allows the VIC to display color RAM 1. The corresponding bits in the DDR register at location \$0000 are then set to input so the operating system does not default to color RAM 0. Then a white circle is drawn around a blue box which is in the same location as the black box, only the color of the blue box goes into color RAM 1. Then the program loops, first displaying color RAM 0 then color RAM 1. To stop the program, press <RUN/STOP> <RESTORE>.

### 3.1.3 Moving the Bitmap Screen

The standard bitmap screen on the C-128, as on the C-64, is 320 pixels wide by 200 pixels high. This adds up to a total of 64000 pixels that comprise a whole bitmap screen. Each pixel corresponds to one bit and 8 bits make 1 byte, so a standard bitmap screen uses a total of 8000 bytes of memory. Since one video bank is 16K this means that there are only 2 places in each video bank for a bitmap screen. Whether the bitmap resides in the upper or lower half of the video bank is dependent on bit 3 of location \$0A2D or \$D018. Even though there is room enough for 2 bitmaps in each video bank, you need 1K of the video bank for the screen. So we can have only one bitmap per video bank, or a total of 8 bitmaps stored in memory.

The following instructions allow you to select the location of the bitmap.

**BASIC:**

```
POKE V, (PEEK(V) AND 247) OR X
```

**machine language :**

```
LDA $V  
AND #$F7  
ORA #$ X  
STA $V
```

**Where:** V = \$0A2D or \$D018 if the Interrupt driven screen editor is disabled  
X = Value from the table below:

Location of the Bitmap

Video Bank	Value of bit 3 (X)	
	0 (0)	1 (8)
0	\$0000	\$2000
1	\$4000	\$6000
2	\$8000	\$A000
3	\$C000	\$E000

**NOTE:** When using any of the above information to move the bitmap or use a different video bank, remember that the operating system expects the bitmap to be at location \$2000 in video bank 0 and its corresponding screen memory starting at location \$0400.

For example, when a graphics command such as DRAW is executed, the operating system always writes to location \$2000 - \$3FFF expecting the bitmap to be there.

So if you move the location of the bitmap, BASIC 7.0 graphics commands cannot be used to draw on the hi-res screen.

### 3.2 Hi-res Graphics from Machine Language

Although BASIC 7.0 has some powerful graphics commands, these same routines can easily be used from machine language. This keeps you from having to write your own graphics routines from scratch. The following table shows some of the different graphics routines accessible from machine language, and the appropriate parameters for each.

<u>ACTION</u>	<u>BASIC EQUIVALENT</u>	<u>ADDRESS</u>	<u>PARAMETERS</u>
Plot a point	DRAW 1,X,Y	\$9BFB	\$1131,\$1132 X Coordinate \$1133,\$1134 Y Coordinate \$116B 1 = Double width 0 = Single width
Draw a line	DRAW 1,X,Y TO X,Y	\$9B30	\$1131,\$1132 Start X Coord. \$1133,\$1134 Start Y Coord. \$1135,\$1136 End X Coord. \$1137,\$1138 End Y Coord. \$116B 1 = Double width 0 = Single width
Draw a box	BOX 1,X1,Y1,X2,Y2	\$62D7	\$1150,\$1151 Top right X Coord. \$1152,\$1153 Top right Y Coord. \$115C,\$115D Bottom rt X Coord. \$115E,\$115F Bottom rt Y Coord. \$1154 Rotation angle X Register Fill flag 0 = No fill
Fill area	PAINT 1,X,Y	\$61C0	Accumulator Mode flag 128 = Fill an area defined by non-background source \$1131,\$1132 X Coordinate \$1133,\$1134 Y Coordinate
Clear screen	SCNCLR (mode)	\$6A92	X Register Screen to clear 0 = 40 Col. text 1 = Standard Bitmap 2 = Split-screen Std. Bitmap 3 = Multicolor Bitmap 4 = Multicolor Split 5 = 80 Col. text

<u>ACTION</u>	<u>BASIC EQUIVALENT</u>	<u>ADDRESS</u>	<u>PARAMETERS</u>
Print chars.	CHAR1,Col,Row,Str	\$68DB	Accumulator
			Character to output
			X Register Column
			Y Register Row
		\$113D	Reverse Flag
			128 = Reverse Chars.
			0 = Normal Chars.
		\$1168	Page number (MSB) of
			character set in ROM.
			Ex: \$D0 for the standard character set

**NOTE:** On all of the above, location \$83 should contain the color source number. All X and Y coordinates are stored in LSB, MSB format.

The following routine is the equivalent of the BASIC 7.0 command `BOX 1, 20, 20, 50, 50, 45, 1`, which draws a filled box that is rotated at a 45 degree angle on the hi-res screen.

```

LDX #$01 ;Select standard bitmap
JSR $6A92 ;Clear it
LDA #$01
STA $83 ;Select color source 1
LDA #$00
STA $1151 ;Set the MSB of the
STA $1153 ;Coordinates
STA $115D ;To zero
STA $115F
LDA #$14 ;Set the top right
STA $1150 ;X coordinate
STA $1152 ;And the top right Y coord. to 20
LDA #$32 ;Set the bottom right
STA $115C ;X coordinate and
STA $115E ;The bottom right Y coord. to 50
LDA #$2D ;Set the rotation angle
STA $1154 ;To 45 degrees
TAX ;Select Fill
JMP $62D7 ;Draw the BOX

```

Of course, these are not the only graphics routines that can be called from machine language. There are several routines that can be used. They can be found in commented form in the ROM listings for \$6000 and \$9000.



## **Chapter Four**

### **The BASIC 7.0 ROM Listing**



## The BASIC 7.0 ROM Listing

This is probably the longest single chapter you'll ever see, encompassing about 500 pages. This disassembled and commented ROM listing details the BASIC commands, and how the operating system determines which command or function is called, as well as when each command or function is required.

Since different versions of the C-128 operating system have been released, your addresses and jumps may occasionally be different from this listing. To confirm any discrepancies, examine your own BASIC 7.0 with the machine language MONITOR equipped on your C-128:

1. Turn on your C-128.
2. Press function key 8 (<SHIFT>-<F7>), or type MONITOR and press <RETURN>.
3. To display a memory range in BASIC 7.0, use the M (memory) command. For example, to see memory from \$451E to \$4576, type M F451E F4576 on the keyboard and end this input with <RETURN>.
4. A more detailed listing can be seen by using the D (disassemble). To disassemble machine code into assembly language mnemonics and operands from \$4000 to \$401A, type D F4000 F401A <RETURN>. The disassembled code also includes the hexadecimal values of the code.
5. To exit the monitor and get back to BASIC, type X <RETURN>.

If you have any questions about the MONITOR, refer to Appendix J of the Commodore 128 System Guide.

```
*****
*
*          COMMODORE 128 ROM LISTINGS
*
*          BASIC ROM LOW      ( $4000 - $7FFF )
*
*
*****
```

```
*****
*
*          KERNAL TO BASIC JUMP ADDRESSES
*
*  Addresses $4000, $4003, and $4006 are used as jump
*  addresses for the KERNAL so that it may allow
*  the BASIC operating system to take control.
*
*****
```

ADDR	SOURCE CODE	COMMENTS
----	-----	-----
4000:	JMP \$4023	;Cold start entry point
4003:	JMP \$4009	;Warm start entry point
4006:	JMP \$A84D	;Interrupt ReQuest entry point (BASIC IRQ)

```
*****
*
*          WRMSTRT
*
*          WaRM STaRT entry point
*
*  This routine resets the I/O channels to the screen
*  for output and the keyboard for input, sets up the
*  sprite and sound tables, sets up the Memory
*  Management Unit for BASIC, resets the string stack,
*  and then prints the READY message to the screen.
*  This routine is also jumped to when the RUN/STOP
*  RESTORE key combination is used, when the C-128
*  is first turned on, and when you exit the monitor.
*  NOTE: You may wish to change the jump address at
*  $0A00 to point to $40FF or $401C to prevent the
*  monitor from resetting these pointers.
*
*****
```

```

4009: JSR  $FFCC      ;Clear any open channels; Set system to default
                        ;I/O (screen as output and keyboard as input).
400C: JSR  $417A     ;Set up MMU configuration registers for the
                        ;BASIC interpreter
400F: JSR  $418D     ;Clear sprite speed/direction table
4012: JSR  $4112     ;Set up the default values for sound registers
4015: JSR  $5238     ;Reset the string stack pointer to $1B
4018: LDA  #0        ;Set the current input device
401A: STA  $15       ;for screen prompting
401C: CLI          ;Allow background jobs to start (INTERRUPTS)
401D: JMP  $4D37     ;Print READY to the screen, enable KERNAL
                        ;Messages and disable control messages
4020: .BYTE $0,$FF,$FF ;Spare jump address (unused)
    
```

```

*****
*
*                               *
*                CLDSTRT       *
*                               *
*                CoLD STaRT entry point
*                               *
*
*  This routine sets up the Memory Management Unit and
*  various BASIC vectors and routines, prints the
*  start-up message to the screen, and then has the
*  KERNAL routine PHOENIX check for an auto boot disk
*  in DRIVE 0, DEVICE 8.
*
*
*  This routine is normally called when the 128 is
*  first turned on, and also whenever the reset button
*  is depressed.
*
*****
    
```

```

4023: JSR  $417A     ;Set up Memory Managment Unit
4026: JSR  $4251     ;Set up the vector tables for BASIC
4029: JSR  $4045     ;Set up the various jmp/zero page RAM routines
402C: JSR  $419B     ;Print the CBM and MICROSOFT display messages
402F: LDA  $0A04     ;Set bit 0 so the KERNAL IRQ
4032: ORA  #1        ;Routine will call the
4034: STA  $0A04     ;BASIC IRQ routine for GRAPHICS/SOUND/LIGHT PEN
4037: LDX  #$03      ;Set up the system restart vector to
4039: STX  $0A00     ;$4003 (normal warm start)
403C: LDX  #$FB      ;Reset the stack pointer
403E: TXS
403F: JSR  $FF56     ;Check TRACK 1 SECTOR 0 for auto boot info.
4042: JMP  $401C     ;Jump to BASIC warm start to restart the
                        ;background Jobs and print the READY message to
                        ;the screen
    
```

```

*****
*
*                               *
*               INIT           *
*           INITIALize registers *
*
* This routine is called by the CLDSTRT routine above *
* to initialize all of the BASIC ZERO PAGE address *
* which have a fixed value. Some of these routines *
* are the USER call routine and the CHRGET routine. *
* NOTE: See the routine block notes below          *
*
*****

*-----*
*
* This routine sets up jump for function evaluation *
* and the USR vector.                               *
*
*-----*

4045: LDA  #$4C      ;Set up the jump for function evaluations
4047: STA  $56
4049: STA  $1218     ;Set up the jump for the USR jump address

*-----*
*
*                               *
*               Point the USR vector to           *
*               an 'ILLEGAL QUANTITY' error message *
*
*-----*

404C: LDA  #$28
404E: LDY  #$7D
4050: STA  $1219
4053: STY  $121A

*-----*
*
* Set up the vector for converting a floating point *
* number to an integer ($849F)                    *
*
*-----*

4056: LDA  #$9F
4058: LDY  #$84
405A: STA  $117A
405D: STY  $117B

```

```

*-----*
*
* Set up the vector for converting an integer to
* a floating point number ($793C)
*
*-----*

4060: LDA  #$3C
4062: LDY  #$79
4064: STA  $117C
4067: STY  $117D

*-----*
*
* Copy ROM addresses $4278-$42CD to RAM: $037F-$03D4
*
*-----*

406A: LDX  #85           ;Copy 85 bytes
406C: LDA  $4278,X      ;Copy subroutines in ROM to $037F-$03D4
406F: STA  $037F,X
4072: DEX
4073: BNE  $406C

*-----*
*
* Initialize various registers
*
*-----*

4075: STX  $03DF      ;Clear the OVERFLOW marker for FAC1
4078: STX  $15        ;Set up the screen prompting flag
407A: STX  $1A        ;Set the MSB of the last descriptor to zero
407C: STX  $116F      ;Turn off the TRACE mode
407F: STX  $1C00      ;1st END OF LINE for BASIC. Note: You must do
                    ;this if you relocate BASIC
4082: STX  $76        ;Clear the HI-RES flag
4084: STX  $74        ;Clear the offset value for the AUTO command
4086: STX  $75        ;Clear MSB of the AUTO command offset
4088: STX  $116B      ;Double width off
408B: STX  $116A      ;Screen scaling to 320 * 200
408E: STX  $116C      ;Clear the temporary pointer for FILL
4091: STX  $121B      ;Clear the initial value for RND command
4094: STX  $011C      ;Clear MSB for the BASIC BSAVE command
4097: STX  $1276      ;Clear the BASIC COLLISION 1 interrupt
409A: STX  $1277      ;Clear the BASIC COLLISION 2 interrupt
409D: STX  $1278      ;Clear the BASIC COLLISION 3 interrupt

```

```

40A0: STX   $127F      ;Clear the INTERRUPT IN PROGRESS flag
40A3: LDY   #$58      ;Clear the sprite data
40A5: STA   $117E,Y   ;Area with $E6
40A8: DEY
40A9: BPL   $40A5
40AB: INX
40AC: STX   $01FD      ;Useless code,
40AF: STX   $01FC      ;unused by any routines
40B2: LDX   #15       ;BANK 15 for
40B4: STX   $03D5      ;BASIC SYS, POKE, PEEK, and WAIT commands
40B7: LDX   #13       ;Medium gray
40B9: STX   $86       ;Foreground color
40BB: LDX   #1        ;Black
40BD: STX   $84       ;Multicolor 1 foreground
40BF: LDX   #2        ;White
40C1: STX   $85       ;Multicolor 2 foreground
40C3: JSR   $6A5C      ;Set up the bkground/border & mult./border reg.
40C6: LDX   #$1B      ;Start of temporary string
40C8: STX   $18       ;Stack for string descriptors
40CA: LDX   #$01      ;Set the START of BASIC to $1C01
40CC: LDY   #$1C
40CE: STX   $2D       ;Save it to TXTTAB
40D0: STY   $2E
40D2: LDA   #$00      ;Set the START of BASIC VARIABLES
40D4: LDY   #$04      ;To $0400 in RAM BANK 1
40D6: STA   $2F
40D8: STY   $30
40DA: LDA   #$00      ;Set the highest address
40DC: LDY   #$FF      ;Available to BASIC to $FF00
40DE: STA   $1212     ;( RAM BANK 0 )
40E1: STY   $1213
40E4: LDA   #$00      ;Set the highest address
40E6: LDY   #$FF      ;Available to BASIC strings
40E8: STA   $39       ;and variables to $FF00
40EA: STY   $3A       ;( RAM BANK 1 )

*-----*
*
* Set up $07F8 - $07FF for Sprite ID area (Data Table) *
*
*-----*

40EC: LDX   #$3F      ;Starting value to be stored at $07FF
40EE: LDY   #7        ;Number of bytes to do
40F0: TXA
40F1: STA   $07F8,Y   ;Store value in SPRITE ID area address
40F4: DEX           ;Decrement the value to be stored by one

```



```

40F5: DEY                ;Decrement the index
40F6: BPL    $40F0      ;Continue until done

*-----*
*
*          Clear the sprite speed/direction table
*
*-----*

40F8: LDA    #0
40FA: LDX    #$6C      ;For 109 bytes to clear
40FC: STA    $117E,X  ;Clear area
40FF: DEX
4100: BPL    $40FC      ;Continue until done
4102: JSR    $4112      ;Set up default values for the sound registers
4104: LDA    #$D0      ;Set up the Page number for
4107: STA    $11EC      ;Uppercase/Graphics character set
410A: LDA    #$D8      ;Set up the Page number for
410C: STA    $11EB      ;Lowercase/Uppercase character set
410F: JMP    $51D9      ;Do a BASIC NEW command

*-----*
*
*  This routine sets up the various default values
*  used for the PLAY command.  It also sets up the SID
*  sound registers.
*
*-----*

4112: LDA    #$20      ;Set up Note TIME to
4114: LDY    #1        ;The length
4116: STA    $1229      ;Of a sixteenth-note
4119: STY    $122A      ;As the default value
411C: LDA    #4        ;Set the default
411E: STA    $122B      ;Octave to 4 in OCTAVE
4121: LDA    #16       ;Set up
4123: STA    $1222      ;TEMPO rate to 16
4126: LDA    #0        ;Turn off
4128: STA    $D404      ;VOICE 1
412B: STA    $D40B      ;VOICE 2
412E: STA    $D412      ;VOICE 3
4131: STA    $12FD      ;Enable the BASIC IRQ routine for
                          ;GRAPHICS/SOUND/LIGHT PEN

4134: LDA    #15
4136: STA    $1274      ;Set up default
4139: STA    $1275      ;FILTER values
413C: STA    $D418      ;Set sound VOLume to maximum

```

```

*-----*
*
* The default sound values are set up in the memory
* locations as listed below.
*
* LOCATIONS           DESCRIPTION
* $123F - $1248        ATTACK, DECAY
* $1249 - $1252        SUSTAIN, RELEASE
* $1253 - $125C        WAVEFORM
* $125D - $1266        PULSE WIDTH LSB
* $1267 - $1270        PULSE WIDTH MSB
*
*-----*

```

```

413F: LDY  #$1D
4141: LDA  $7011,Y    ;Set UP ATTACK, DECAY
4144: STA  $123F,Y    ;SUSTAIN, RELEASE and WAVEFORM
4147: DEY
4148: BPL  $4141

```

```

*-----*
*
*           Set up pulse width MSB
*
*-----*

```

```

414A: LDX  #9
414C: LDA  $702F,X    ;Set up PULSE WIDTH
414F: STA  $1267,X
4152: DEX
4153: BPL  $414C
4155: STX  $1285      ;Clear TIME STORAGE low
4158: STX  $1286      ;Clear TIME STORAGE med
415B: STX  $1287      ;Clear TIME STORAGE high
415E: STX  $1224
4161: STX  $1226
4164: STX  $1228

```

```

*-----*
*
*           Initialize SID VOICES 1 - 3
*
*-----*

```

```

4167: LDY  #2          ;Counter to voice number
4169: STY  $122F      ;Save it as a counter
416C: LDX  #0

```

```

416E: JSR   $6EB2      ;Initialize voice w/ values stored $123F-$1271
4171: DEC   $122F      ;To next voice
4174: BPL   $416C
4176: INC   $122F      ;Counter to zero
4179: RTS
    
```

```

*****
*
*                               SETMMU                               *
*
*           SET up the Memory Management Unit                       *
*
*           Configure the MMU registers A-D.                         *
*
*****
    
```

```

417A: JSR   $A845      ;Do a BASIC BANK 15 command
417D: LDX   #3         ;Index pointer
417F: LDA   $4189,X    ;Preset values in ROM
4182: STA   $D501,X    ;To set up MMU registers
4185: DEX
4186: BPL   $417F
4188: RTS              ;And exit
    
```

```

*****
*
*                               MMUTAB                               *
*
*           Memory Management Unit TABLE                           *
*
*           Table for preconfiguration of PCRA through PCRD        *
*
*****
    
```

```

BITS                                                                 76543210
-----
4189: .BYTE  $3F        ;BASIC BANK 0                               % 00111111
418A: .BYTE  $7F        ;BASIC BANK 1                               % 01111111
418B: .BYTE  $01        ;BASIC BANK 14                              % 00000001
418C: .BYTE  $41        ;RAM BANK 1, BASIC, KERNAL, CHR ROM % 01000001
                                ;The same as BASIC BANK 14 except
                                ;RAM BANK 1
    
```

```

*****
*
*                               SETSPR                               *
*
*   SET speed/direction table for the SPrites.                       *
*
*****

418D: LDA    #0
418F: LDY    #7           ;Index pointer
4191: LDX    $6DD9,Y     ;Pointer to register
4194: STA    $117E,X     ;Clear register
4197: DEY
4198: BPL    $4191
419A: RTS                ;And exit

*****
*
*                               STRMSG                               *
*
*   Print the STaRTup MeSsaGe to the screen                          *
*
*****

419B: LDY    #0
419D: LDA    $41BB,Y     ;Get a byte of the startup message
41A0: CMP    #'@'       ;New line flag?
41A2: BNE    $41B2       ;No, then branch to print the character
41A4: BIT    $D7         ;Check to see if we're in 40- or 80-column mode
41A6: BPL    $41B5       ;Branch if in 40-column mode

*-----*
*
*   If you are in the 80-column mode, this routine                  *
*   will print nineteen spaces before the start of                  *
*   each new sentence to center the text on the 80-                 *
*   column screen.                                                  *
*
*-----*

41A8: LDX    #19         ;Nineteen spaces
41AA: LDA    #32         ;To the screen
41AC: JSR    $9269       ;Continue to print one space at a time
41AF: DEX
41B0: BNE    $41AA       ;Zero.
41B2: JSR    $9269       ;Print the character in the accumulator
41B5: INY                ;Increment the index pointer

```

```
41B6: CPY   #$96      ;97 characters printed yet?
41B8: BNE   $419D    ;No, then continue printing
41BA: RTS                      ;Exit the routine
```

```
*****
*
*                               MSGTBL
*
*       The startup MeSsage TaBLe is as follows:
*
*****
```

```
41BB: .BYTE  $93      ;Clear the screen
41BC: .BYTE  $0D      ;<CR>
41BD: TXT    '@'      ;Start of line pointer
41BE: TXT    '        commodore basic v7.0 122365 bytes free'
41E5: .BYTE  $0D      ;<CR>
41E6: TXT    '@'      ;Start of line pointer
41E7: TXT    '        (c)1985 commodore electronics, ltd.'
420D: .BYTE  $0D      ;<CR>
420E: TXT    '@'      ;Start of line pointer
420F: TXT    '        (c)1977 microsoft corp.'
         $0D          ;<CR>
4230: TXT    '@'      ;Start of line pointer
4231: TXT    '        all rights reserved'
424F: .BYTE  $0D,$00  ;<CR>, End of text flag
```

```
*****
*
*                               COPYVEC
*
*       COPY the system VECtor tables to RAM
*
*****
```

```
4251: LDX   #17        ;Copy 18 bytes or 9 vector addresses to RAM
         ;Starting at $0300
4253: LDA   $4267,X    ;Get the BASIC vectors from ROM
4256: STA   $0300,X    ;And store them to RAM
4259: DEX                      ;Do 18 bytes
425A: BPL   $4253
```

```

*****
*
* This routine sets up an indirect jump vector at
* $02FC/$02FD to jump to $4C78. When entering this
* routine, if the carry is set a SYNTAX ERROR will
* result. If the carry is clear, the VALTYP flag
* at $0F is checked for to see if the type is numeric.
* If the type is not numeric, a SYNTAX ERROR will
* result.
*
*****

```

```

425C: LDA    #$78      ;Low byte of address
425E: STA    $02FC     ;Store it
4261: LDA    #$4C      ;High byte of address
4263: STA    $02FD     ;Store it
4266: RTS              ;And exit

```

```

*****
*
*                                VECTBL
*
*                                system VECTOR TaBLE
*
* This table contains the vectors for some of the
* important routines used by BASIC. This table is
* copied to RAM starting at $0300.
*
*****

```

```

4267: DA    $4D3F     ;Vector: Error message routine (IERROR)
4269: DA    $4DC6     ;Vector: Read/exec. basic line (IMAIN)
426B: DA    $430D     ;Vector: Convert interpreter code (ICRNCH)
426D: DA    $5151     ;Vector: Convert to text (list) (IQPLOP)
426F: DA    $4AA2     ;Vector: Execute the keyword (IGONE)
4271: DA    $78DA     ;Vector: Evaluate expression (IEVAL)
4273: DA    $4321     ;Vector: Conversion routine (ICON)
4275: DA    $51CD     ;Vector: Escape list (IESLST)
4277: DA    $4BA9     ;Vector: Execute escape (ISERROR)

```

```

*****
*
*                               CHRGET
*
*                               CHaRacter GET routine
*
*                               Transferred into RAM located at $0380
*
* This routine will read one character at a time from
* the memory location pointed to by TXTPTR ($3D/$3E).
* Before each character is read, this routine will
* increment the TXTPTR, and will fall through to the
* CHRGOT routine below.
*
*****

```

```

4279: INC   $3D           ;Increment BASIC text pointer LOW byte
427B: BNE   $427F        ;If low byte is not equal to zero,
                        ;Then don't increment the high byte
427D: INC   $3E           ;If low byte is equal to zero,
                        ;Then increment the high byte

```

```

*****
*
*                               CHRGOT
*
*                               CHaRacter GOT routine
*
*                               Transferred into RAM located at $0386
*
* This routine reads the character that TXTPTR points
* to plus the index register. Then it will fall through
* to QNUM. This routine also will read the byte from
* BANK 0 and return to BANK 14.
*
*****

```

```

427F: STA   $FF01        ;Enable BASIC BANK 0
4282: LDY   #0           ;Index pointer
4284: LDA   ($3D),Y      ;Get character from BASIC text
4286: STA   $FF03        ;Enable BASIC BANK 14

```

```

*****
*
*                               QNUM
*
*                               Query NUMBER routine
*
* CARRY FLAG - is cleared if the character is an ASCII
* digit 0-9.  If it is not a digit, the carry flag is
* set.
*
* ZERO FLAG - is set if the character is the statement
* terminator ($00), or if it is the statement chain
* flag which is the colon ':'.  The zero flag is
* cleared if the character is neither the statement
* terminator or the chain flag.
*
*****

```

```

4289: CMP   #' : '      ;If character greater than ':' then
                        ;The carry flag is set
428B: BCS   $4297      ;If char. is greater than or equal to ':' branch
428D: CMP   #$20       ;Check to see if the character is a space
428F: BEQ   $4279      ;If it is then skip it and get the next char.
4291: SEC
                        ;Set the carry for subtraction
4292: SBC   #'0'       ;Subtract to make true numbers if they are 0 - 9
4294: SEC
                        ;Set the carry for subtraction
4295: SBC   #$D0       ;If the character is less than 0,
                        ;Then set the carry flag
4297: RTS
                        ;The carry flag is cleared for numbers on exit

```

```

*****
*
*                               BASIC INDSUB RAM 0 ROUTINE
*
*                               INDirect SUBroutine load from RAM bank 0
*
* This routine reads a byte from BANK 0 and then
* returns to BANK 14.
*
* EXAMPLE: To get a byte from location $0100, BANK 0,
* store the LOW/HIGH byte address ($00,$01) to any
* ZERO-PAGE location such as $61, $62.  Then put the
* low byte of the zero - page address in the
* accumulator.  In this case, it would be $61.
* Then put the index to the byte wanted in the Y
* register, and JSR to this routine.  On return,
* the byte wanted will be in the accumulator.
*

```



```

* NOTE: These steps also apply to the next two      *
* routines.                                          *
*                                                    *
*****
*                                                    *
*                Stored in RAM at $039F             *
*                                                    *
*****

```

```

4298: STA  $03A6      ;Zero page location to load from
429B: STA  $FF01      ;Enable BASIC BANK 0
429E: LDA  ($00),Y    ;Get a byte from BANK 0 at indirect address in
                        ;$03A6
42A0: STA  $FF03      ;Enable BASIC BANK 14
42A3: RTS                ;And exit

```

```

*****
*                                                    *
*                BASIC INDSUB RAM 1 ROUTINE         *
*                                                    *
* This routine reads a byte from BANK 1 and then   *
* returns to the RAM BANK 1, BASIC, KERNAL and I/O *
* memory configuration.                            *
*                                                    *
*****
*                                                    *
*                Stored in RAM at $03AB             *
*                                                    *
*****

```

```

42A4: STA  $03B2      ;Zero page location to load from
42A7: STA  $FF02      ;Enable BASIC BANK 1
42AA: LDA  ($00),Y    ;Get a byte from BANK 1 at indirect address in
                        ;$03A6
42AC: STA  $FF04      ;Enable RAM BANK 1, BASIC, KERNAL, and Char. ROM
42AF: RTS                ;And exit

```

```

*****
*                                                    *
*                BASIC INDIN1 RAM 1 ROUTINE         *
*                                                    *
* This routine reads a byte stored in BANK 1 from the *
* location that is stored in $24,$25 + the index to *
* byte in the Y register. The value of the byte     *
* is returned in the accumulator. After this has   *
* been completed, this routine returns to the BANK 15 *

```

```

* with RAM BANK 1 enabled, not RAM BANK 0.
*
*****
*
*           Stored in RAM at $03B7
*
*****

```

```

42B0: STA  $FF02      ;Enable BANK 1
42B3: LDA  ($24),Y   ;Get a byte from memory at indirect address
                        ;Stored at $24,$25
42B5: STA  $FF04      ;Enable RAM BANK 1, BASIC, KERNAL, and Char. ROM
42B8: RTS                        ;And exit

```

```

*****
*
*           BASIC INDIN2 RAM 0 ROUTINE
*
* This routine reads a byte stored in BANK 0 from the
* location that is stored in $26,$27 + the index to
* byte in the Y register. The value of the byte
* is returned in the accumulator. After this has
* been completed, this routine returns to BANK 14.
*
*****
*
*           Stored in RAM at $03C0
*
*****

```

```

42B9: STA  $FF01      ;Enable BANK 0
42BC: LDA  ($26),Y   ;Get a byte of memory from the indirect address
                        ;at $26,$27
42BE: STA  $FF03      ;Enable BANK 14
42C1: RTS                        ;And exit

```

```

*****
*
*           BASIC INDTXTPTR RAM 0 ROUTINE
*
* This routine reads a byte stored in BANK 0 from
* the location that is stored in TXTPTR pointer at
* $3D, $3E + the index to byte in the Y register.
* The value of the byte read is returned in the
* accumulator. After this has been completed, this
* routine returns to BANK 14.
*
*****

```

```
*****
*
*           Stored in RAM at $03C9
*
*****
```

```
42C2: STA  $FF01      ;Enable BANK 0
42C5: LDA  ($3D),Y   ;Get a character from BASIC text
42C7: STA  $FF03      ;Enable BANK 14
42CA: RTS                ;And exit
42CB: .BYTE 0,0,0    ;Spare storage area in RAM
```

```
*****
*
*   End of the ROM area that is copied into zero page.
*
*****
```

```
*****
*
*           BASIC INTERPRETER JUMP TABLE
*
*   This jump table is used by the BASIC interpreter
*   to load a byte from a BANK and then return to the
*   bank as indicated.
*
*****
```

* BANK		BANK		INDIRECT		ROUTINE	*
* LOAD		RETURN		ADDRESS		CALLED	*
* FROM		TO		LOCATION			*

-----

```
42CE: LDA  #$50      ;BANK 1      BANK *      $50, $51
42D0: JMP  $03AB      ;                               INDSUB: RAM 1
42D3: LDA  #$3F      ;BANK 1      BANK *      $3F, $40
42D5: JMP  $03AB      ;                               INDSUB: RAM 1
42D8: LDA  #$52      ;BANK 1      BANK *      $52, $53
42DA: JMP  $03AB      ;                               INDSUB: RAM 1
42DD: LDA  #$5C      ;BANK 0      BANK 14     $5C, $5D
42DF: JMP  $039F      ;                               INDSUB: RAM 0
42E2: LDA  #$5C      ;BANK 1      BANK *      $5C, $5D
42E4: JMP  $03AB      ;                               INDSUB: RAM 1
42E7: LDA  #$66      ;BANK 1      BANK *      $66, $67
42E9: JMP  $03AB      ;                               INDSUB: RAM 1
42EC: LDA  #$61      ;BANK 0      BANK 14     $61, $62
42EE: JMP  $039F      ;                               INDSUB: RAM 0
```

```

42F1: LDA  #$70      ;BANK 0      BANK 14      $70, $71
42F3: JMP  $039F      ;                                INDSUB: RAM 0
42F6: LDA  #$70      ;BANK 1      BANK   *      $70, $71
42F8: JMP  $03AB      ;                                INDSUB: RAM 1
42FB: LDA  #$50      ;BANK 1      BANK   *      $50, $51
42FD: JMP  $03AB      ;                                INDSUB: RAM 1
4300: LDA  #$61      ;BANK 1      BANK   *      $61, $62
4302: JMP  $03AB      ;                                INDSUB: RAM 1
4305: LDA  #$24      ;BANK 0      BANK 14      $24, $25
4307: JMP  $039F      ;                                INDSUB: RAM 0
;          Same as BANK 14 except with RAM BANK 1

```

```

*****
*
*                               *
*          CRNCH                 *
*
*          CRUNCH tokens        *
*
*          CONVERT INTERPRETER CODE
*
*-----*
*
* This routine and CON below are used to convert
* a BASIC line when it is first entered into tokenized
* text (but not in quotes). This routine is vectored
* through $0304; you can 'tokenize' your own commands.
*
*****

```

```

430A: JMP  ($0304)    ;Normal entry point is $430D
430D: LDA  $3D        ;Get the current BASIC text pointers
430F: PHA                ;And save them onto the stack
4310: LDA  $3E        ;For recall after routine
4312: PHA                ;Execution
4313: JSR  $0386      ;Get the current character that pointers are at
4316: JMP  $431C      ;Process it
4319: JSR  $0380      ;Get the next character by incrementing pointer
431C: BCC  $4319      ;Loop until the next character is found
;That is not an ASCII character that represents
;A digit 0-9 (the BASIC line number)

```

```

*****
*
*                               CON                               *
*                               CONVersion ROUTINE              *
*
*-----*
*
* This routine searches a BASIC program line, finds
* the BASIC keywords, tokenizes them into their respec-
* tive one or two byte tokens, and stores the token(s)
* into the BASIC program line. Enter this routine with
* a value in the accumulator and the carry set for a
* character or with the carry cleared for a digit
*
*****

```

```

431E: JMP    ($030C)    ;Normal entry point $4321
4321: BCS    $4326      ;If value is a character, then continue
4323: JMP    $43B2      ;If it is not a character, then branch
4326: CMP    #0          ;Is it the end of the line?
4328: BEQ    $43A1      ;Yes, then restore TXTPTR and exit to READY
432A: CMP    #':'        ;Is it a colon?
432C: BEQ    $4319      ;Yes, so skip it and get the next character
432E: CMP    #'?'       ;Is it the print command?
4330: BNE    $4336      ;No, then continue checking
4332: LDA    #$99        ;It is the ? command, so load the accumulator
                          ;With the token for PRINT
4334: BNE    $4386      ;Branch to store the token in the line
4336: CMP    #$80        ;Is it greater than or equal to 128?
4338: BCC    $4345      ;No, continue checking
433A: CMP    #$FF        ;Is it the π character?
433C: BEQ    $4319      ;If it is, then skip it
433E: LDY    #1          ;Offset of one byte
4340: JSR    $43CC
4343: BEQ    $4313      ;If $80 or greater then erase it
4345: CMP    #'"'        ;Is it a quote mark?
4347: BNE    $4356      ;No, then check for keyword
4349: JSR    $0380      ;If quote mark, get next char from BASIC text
434c: CMP    #0          ;Is it the end of the line $00?
434E: BEQ    $43A1      ;Yes , then restore TXTPTR and exit
4350: CMP    #'"'        ;Is it the second quote mark?
4352: BEQ    $4319      ;Yes it is, so skip it
4354: BNE    $4349      ;No it is not the second quote mark,
                          ;Then keep looking until the end of the line is
                          ;Found or the second quote mark is found
                          ;In other words, skip the text between quote
                          ;Marks or from first quote to the end of line

```

```

*****
*
*           SEARCH FOR DUAL TOKEN COMMANDS           *
*
*****

4356: LDA   #$46       ;MSB of table
4358: LDY   #$09       ;LSB of table
435A: JSR   $43E2      ;Check for a match
435D: BCC   $4365      ;Not found, then check for dual token functions
435F: LDA   #$81       ;Base value for
4361: LDX   #0         ;Command token
4363: BEQ   $43B0      ;Unconditional branch to put token in line

*****
*
*           SEARCH FOR DUAL TOKEN FUNCTIONS           *
*
*****

4365: LDA   #$46       ;MSB of table
4367: LDY   #$C9       ;LSB of table
4369: JSR   $43E2      ;Check for a match
436C: BCC   $4374      ;Not found, then branch and continue checking
436E: LDA   #$81       ;Base value for
4370: LDX   #$FF       ;Function token
4372: BNE   $43B0      ;Unconditional branch to put token in line

*****
*
*           SEARCH FOR SINGLE TOKEN FUNCTIONS         *
*
*****

4374: LDA   #$44       ;MSB of table
4376: LDY   #$17       ;LSB of table
4378: JSR   $43E2      ;Check for a match
437B: BCC   $4319      ;Not found, then get the next character

*****
*
*           INSERT A TOKEN INTO A PROGRAM LINE         *
*
*****

437D: CPY   #0         ;For no offset
437F: BEQ   $4384      ;If no offset, then skip the next instruction

```

```

4381: JSR   $43CC      ;Shift line back Y number of bytes
4384: LDA   $0D        ;Get the token value
4386: LDY   #0          ;Zero the index pointer
4388: STA   ($3D),Y     ;Place the token with the BASIC text
438A: CMP   #$8F        ;Is it the REM token?
438C: BEQ   $439B      ;Yes it is, then ignore the rest of line
438e: CMP   #$83        ;Is it a RETURN?
4390: BNE   $4319      ;No, then get the next character
4392: JSR   $0380      ;Get the next character
4395: JSR   $528F      ;Search for colon or $00, end of line marker
4398: JMP   $4313      ;Continues with loop
439B: JSR   $0380      ;Get the next character
439E: JSR   $529D      ;Searches for end of line $00

```

```

*****
*
* This routine calculates the offset between the
* beginning of a keyword and the end of the keyword.
* The resulting offset is returned in the Y register.
*
*****

```

```

43A1: LDX   $3D        ;BASIC text pointer low byte
43A3: PLA
43A4: STA   $3E        ;Text pointer from
43A6: PLA
43A7: STA   $3D        ;And restore TXTPTR
43A9: SEC
43AA: TXA
43AB: SBC   $3D        ;Subtract where we left off from
43AD: TAY
43AE: INY
43AF: RTS

```

```

*****
*
* This routine places dual tokens into the BASIC
* program line.
*
*****

```

```

43B0: ADC   $0D        ;Add the base value $81 to the keyword number
43B2: PHA
43B3: DEY
43B4: JSR   $43CC      ;Shift line back Y number of bytes
43B7: LDA   #$FE        ;Token for dual BASIC command
43B9: INX

```

```

43BA: BNE   $43BE      ;Not at the end of the buffer? Then branch
43BC: LDA   #$CE      ;Token for dual token BASIC function
43BE: LDY   #0         ;Zero the index pointer
43C0: STA   ($3D),Y   ;Place the token with the BASIC text
43C2: INY                   ;Increment the text pointer
43C3: PLA                   ;Get the original token off the stack
43C4: STA   ($3D),Y   ;And place the token with the BASIC text
43C6: JSR   $0380     ;Get the next BASIC text character
43C9: JMP   $4319     ;Get next character

```

```

*****
*
* This routine shifts the BASIC line back the number
* of bytes stored in the Y register.
*
*****

```

```

43CC: CLC                   ;Clear the carry flag for subtraction
43CD: TYA                   ;Transfer offset value into the accumulator
43CE: ADC   $3D             ;Add it to the LSB of TXTPTR
43D0: STA   $24             ;Save it in the temporary work area
43D2: LDA   $3E             ;Get the TXTPTR MSB
43D4: ADC   #0              ;Add the carry flag
43D6: STA   $25             ;Save it in the temporary work area

```

```

*****
*
* Now we have the start of the keyword in $3D, $3E
* and the end of the keyword in $24, $25.
*
*****

```

```

43D8: LDY   #$FF          ;Y=$FF to compensate for the next instruction
43DA: INY                   ;On the first pass Y = 0, all other passes
                               ;increment the Y register by 1
43DB: LDA   ($24),Y       ;Get the byte after the keyword in the line
43DD: STA   ($3D),Y       ;Store Y number of bytes from start of keyword
43DF: BNE   $43DA         ;If not the end of line $00, continue
43E1: RTS                   ;If it is the end of line, then exit

```



```

*****
*
* This routine checks for a match in the keyword      *
* table whose address is in the Y register and the   *
* accumulator. (LOW/HIGH byte format)                *
*
*****

```

```

43E2: STA  $25      ;Save the high byte of command table for search
43E4: STY  $24      ;Save the low byte
43E6: LDY  #0       ;Zero the index pointer
43E8: STY  $0D      ;Save it in TEMP as a keyword number
43EA: DEY                ;Decrement pointer
43EB: INY                ;Increment pointer to zero for start of loop
43EC: LDA  ($3D),Y    ;Get a character from the BASIC text
43EE: SEC                ;And compare it to the character in the
43EF: SBC  ($24),Y    ;BASIC command/function table
43F1: BEQ  $43EB      ;If they are the same, then get the next char.
43F3: CMP  #$80       ;Is it the end of the single
43F5: BEQ  $4412      ;BASIC commands flag? If so, branch
43F7: LDA  ($24),Y    ;If not, get the next character
43F9: BMI  $43FE      ;If it is a shifted character, then branch
43FB: INY                ;If it is not, then increment the pointer
43FC: BNE  $43F7      ;Keep looping until the end flag is found or Y=0
43FE: INY                ;Increment Y to the next character
43FF: INC  $0D        ;Keyword number + 1
4401: CLC                ;Clear the carry for addition
4402: TYA                ;Add the offset to the table address
4403: ADC  $24          ;To get to the
4405: STA  $24          ;Next keyword
4407: BCC  $440B      ;If carry is clear, then skip ahead
4409: INC  $25          ;If carry is set, then increment MSB by 1
440B: CLC                ;Clear the carry
440C: LDY  #0          ;Index to character
440E: LDA  ($24),Y    ;Get a byte from table
4410: BNE  $43EC      ;If not zero, then back to checking
4412: ORA  $0D        ;If zero or called by $43F5 combine accumulator
4414: STA  $0D        ;With the keyword number to make the
                          ;Token and store it in $0D
4416: RTS                ;And exit the routine

```

```

*****
*
*                               RESLST
*
*                               REServed word LiST
*
* The area of memory from $4417 to $46F7 contains
* the list of reserved words that are used by the
* BASIC operating system for its various routines.
*
* NOTE: The last character of each reserved word
*       is stored as a shifted character (bit 7 set)
*       to indicate the end of that reserved word.
*       For instance, the command 'end' would be
*       stored as the HEX values of $45, $4E, and
*       $C4.  If the reserved word consists of only
*       one character, then that character is
*       stored as a shifted character.
*
*****

```

```

-----*
*
*                               SINGLE TOKEN BASIC COMMANDS
*
*-----*

```

	COMMAND	TOKEN
	-----	-----
4417:	TXT 'end'	;\$80
441A:	TXT 'for'	;\$81
441D:	TXT 'next'	;\$82
4421:	TXT 'data'	;\$83
4425:	TXT 'input#'	;\$84
442B:	TXT 'input'	;\$85
4430:	TXT 'dim'	;\$86
4433:	TXT 'read'	;\$87
4437:	TXT 'let'	;\$88
443A:	TXT 'goto'	;\$89
443E:	TXT 'run'	;\$8A
4441:	TXT 'if'	;\$8B
4443:	TXT 'restore'	;\$8C
444A:	TXT 'gosub'	;\$8D
444F:	TXT 'return'	;\$8E
4455:	TXT 'rem'	;\$8F
4458:	TXT 'stop'	;\$90

```

445C: TXT 'on' ;$91
445E: TXT 'wait' ;$92
4462: TXT 'load' ;$93
4466: TXT 'save' ;$94
446A: TXT 'verify' ;$95
4470: TXT 'def' ;$96
4473: TXT 'poke' ;$97
4477: TXT 'print#' ;$98
447D: TXT 'print' ;$99
4482: TXT 'cont' ;$9A
4486: TXT 'list' ;$9B
448A: TXT 'clr' ;$9C
448D: TXT 'cmd' ;$9D
4490: TXT 'sys' ;$9E
4493: TXT 'open' ;$9F
4497: TXT 'close' ;$A0
449C: TXT 'get' ;$A1
449F: TXT 'new' ;$A2
44A2: TXT 'tab(' ;$A3
44A6: TXT 'to' ;$A4
44A8: TXT 'fn' ;$A5
44AA: TXT 'spc(' ;$A6
44AE: TXT 'then' ;$A7
44B2: TXT 'not' ;$A8
44B5: TXT 'step' ;$A9
    
```

```

*-----*
*
*          BASIC MATH OPERATORS          *
*
*-----*
    
```

	OPERATOR	TOKEN
	-----	-----
44B9:	TXT '+'	;\$AA
44BA:	TXT '-'	;\$AB
44BB:	TXT '*'	;\$AC
44BC:	TXT '/'	;\$AD
44BD:	TXT '^'	;\$AE
44BE:	TXT 'and'	;\$AF
44C1:	TXT 'or'	;\$B0
44C3:	TXT '>'	;\$B1
44C4:	TXT '='	;\$B2
44C5:	TXT '<'	;\$B3

```

*-----*
*
*          BASIC NUMERIC FUNCTIONS
*
*-----*
    
```

	FUNCTION	TOKEN
	-----	-----
44C6:	TXT 'sgn'	;\$B4
44C9:	TXT 'int'	;\$B5
44CC:	TXT 'abs'	;\$B6
44CF:	TXT 'usr'	;\$B7
44D2:	TXT 'fre'	;\$B8
44D5:	TXT 'pos'	;\$B9
44D8:	TXT 'sqr'	;\$BA
44DB:	TXT 'rnd'	;\$BB
44DE:	TXT 'log'	;\$BC
44E1:	TXT 'exp'	;\$BD
44E4:	TXT 'cos'	;\$BE
44E7:	TXT 'sin'	;\$BF
44EA:	TXT 'tan'	;\$C0
44ED:	TXT 'atn'	;\$C1
44F0:	TXT 'peek'	;\$C2
44F4:	TXT 'len'	;\$C3

```

*-----*
*
*          BASIC STRING FUNCTIONS
*
*-----*
    
```

	FUNCTION	TOKEN
	-----	-----
44F7:	TXT 'str\$'	;\$C4
44FB:	TXT 'val'	;\$C5
44FE:	TXT 'asc'	;\$C6
4501:	TXT 'chr\$'	;\$C7
4505:	TXT 'left\$'	;\$C8
450A:	TXT 'right\$'	;\$C9
4510:	TXT 'mid\$'	;\$CA
4514:	TXT 'go'	;\$CB
4516:	TXT 'rgr'	;\$CC
4519:	TXT 'rclr'	;\$CD
451D:	.BYTE \$80	;\$CE

;\$CE) ;Flag for Dual Token

```

*-----*
*
*           ADDED BASIC COMMANDS           *
*
*-----*
    
```

	COMMAND	TOKEN	
	-----	-----	
451E:	TXT	'joy'	;\$CF
4521:	TXT	'rdot'	;\$D0
4525:	TXT	'dec'	;\$D1
4528:	TXT	'hex\$'	;\$D2
452C:	TXT	'err\$'	;\$D3
4530:	TXT	'instr'	;\$D4
4535:	TXT	'else'	;\$D5
4539:	TXT	'resume'	;\$D6
453F:	TXT	'trap'	;\$D7
4543:	TXT	'tron'	;\$D8
4547:	TXT	'troff'	;\$D9
454C:	TXT	'sound'	;\$DA
4551:	TXT	'vol'	;\$DB
4554:	TXT	'auto'	;\$DC
4558:	TXT	'pundef'	;\$DD
455D:	TXT	'graphic'	;\$DE
4564:	TXT	'paint'	;\$DF
4569:	TXT	'char'	;\$E0
456D:	TXT	'box'	;\$E1
4570:	TXT	'circle'	;\$E2
4576:	TXT	'gshape'	;\$E3
457C:	TXT	'sshape'	;\$E4
4582:	TXT	'draw'	;\$E5
4586:	TXT	'locate'	;\$E6
458C:	TXT	'color'	;\$E7
4591:	TXT	'scnclr'	;\$E8
4597:	TXT	'scale'	;\$E9
459C:	TXT	'help'	;\$EA
45A0:	TXT	'do'	;\$EB
45A2:	TXT	'loop'	;\$EC
45A6:	TXT	'exit'	;\$ED
45AA:	TXT	'directory'	;\$EE
45B3:	TXT	'dsave'	;\$EF
45B8:	TXT	'dload'	;\$F0
45BD:	TXT	'header'	;\$F1
45C3:	TXT	'scratch'	;\$F2
45CA:	TXT	'collect'	;\$F3
45D1:	TXT	'copy'	;\$F4

```

45D5: TXT 'rename' ;$F5
45DB: TXT 'backup' ;$F6
45E1: TXT 'delete' ;$F7
45E7: TXT 'renumber' ;$F8
45EF: TXT 'key' ;$F9
45F2: TXT 'monitor' ;$FA
45F9: TXT 'using' ;$FB
45FE: TXT 'until' ;$FC
4603: TXT 'while' ;$FD
4608: .BYTE $00 ;Flag for dual token command ($FE)

```

```

*-----*
*                                     *
*                               DTOKEC *
*                                     *
*                               Dual TOKE *
*                                     *
* Each token in this portion of the reserved word *
* list is prefixed by $FE. *
*                                     *
* Example: The BASIC command BANK would be tokenized *
*           as $FE, $02. *
*                                     *
*-----*

```

	COMMAND	TOKEN
	-----	-----
4609:	TXT 'bank'	;\$02
460D:	TXT 'filter'	;\$03
4613:	TXT 'play'	;\$04
4617:	TXT 'tempo'	;\$05
461C:	TXT 'movspr'	;\$06
4622:	TXT 'sprite'	;\$07
4628:	TXT 'sprcolor'	;\$08
4630:	TXT 'rreg'	;\$09
4634:	TXT 'envelope'	;\$0A
463C:	TXT 'sleep'	;\$0B
4641:	TXT 'catalog'	;\$0C
4648:	TXT 'dopen'	;\$0D
464D:	TXT 'append'	;\$0E
4653:	TXT 'dclose'	;\$0F
4659:	TXT 'bsave'	;\$10
465E:	TXT 'blood'	;\$11
4663:	TXT 'record'	;\$12
4669:	TXT 'concat'	;\$13
466F:	TXT 'dverify'	;\$14



```

*****
*
*                               ADDRST
*
* This area of memory from $46F8 to $484A contains
* an ADDRESS LIST for the routines that handle each
* of the reserved words that are used by the BASIC
* operating system.
*
*****
    
```

```

*-----*
*
*                               BASIC COMMAND ADDRESSES
*
* NOTE: Add one to the address shown to obtain
* the starting address of the routine that
* handles that particular reserved word.
*
* NOTE: The operand DA means Define Address which
* places the address after the operand in
* memory in the LSB/MSB format. For instance,
* DA $4BCC would store the address $4BCC in
* memory as $CC, $4B.
*
*-----*
    
```

	ADDRESS	COMMAND
	-----	-----
46FC:	DA \$4BCC	;END
46FE:	DA \$5DF8	;FOR
4700:	DA \$57F3	;NEXT
4702:	DA \$528E	;DATA
4704:	DA \$5647	;INPUT#
4706:	DA \$5661	;INPUT
4708:	DA \$587A	;DIM
470A:	DA \$56A8	;READ
470C:	DA \$53C5	;LET
470E:	DA \$59DA	;GOTO
4710:	DA \$5A9A	;RUN
4712:	DA \$52C4	;IF
4714:	DA \$5AC9	;RESTORE
4716:	DA \$59CE	;GOSUB
4718:	DA \$5261	;RETURN
471A:	DA \$529C	;REM
471C:	DA \$4BCA	;STOP



---

```
471E: DA $53A2 ;ON
4720: DA $6C2C ;WAIT
4722: DA $912B ;LOAD
4724: DA $9111 ;SAVE
4726: DA $9128 ;VERIFY
4728: DA $84F9 ;DEF
472A: DA $80E4 ;POKE
472C: DA $5539 ;PRINT#
472E: DA $5559 ;PRINT
4730: DA $5A5F ;CONT
4732: DA $50E1 ;LIST
4734: DA $51F7 ;CLR
4736: DA $553F ;CMD
4738: DA $5884 ;SYS
473A: DA $918C ;OPEN
473C: DA $9199 ;CLOSE
473E: DA $5611 ;GET
4740: DA $51D5 ;NEW
4742: DA $5390 ;GO
4744: DA $5F61 ;RESUME
4746: DA $5F4C ;TRAP
4748: DA $58B3 ;TRON
474A: DA $58B6 ;TROFF
474C: DA $71EB ;SOUND
474E: DA $71C4 ;VOL
4750: DA $5974 ;AUTO
4752: DA $5F33 ;PUDEF
4754: DA $6B59 ;GRAPHIC
4756: DA $61A7 ;PAINT
4758: DA $67D6 ;CHAR
475A: DA $62B6 ;BOX
475C: DA $668D ;CIRCLE
475E: DA $658C ;GSHAPE
4760: DA $642A ;SSHAPE
4762: DA $6796 ;DRAW
4764: DA $6954 ;LOCATE
4766: DA $69E1 ;COLOR
4768: DA $6A78 ;SCNCLR
476A: DA $695F ;SCALE
476C: DA $5985 ;HELP
476E: DA $5FDF ;DO
4770: DA $6089 ;LOOP
4772: DA $6038 ;EXIT
4774: DA $A07D ;DIRECTORY
4776: DA $A18B ;DSAVE
4778: DA $A1A6 ;DLOAD
477A: DA $A266 ;HEADER
```

---

```

477C: DA    $A2A0    ;SCRATCH
477E: DA    $A32E    ;COLLECT
4780: DA    $A345    ;COPY
4782: DA    $A36D    ;RENAME
4784: DA    $A37B    ;BACKUP
4786: DA    $5E86    ;DELETE
4788: DA    $5AF7    ;RENUMBER
478A: DA    $6109    ;KEY
478C: DA    $AFFF    ;MONITOR
478E: DA    $6BC8    ;BANK
4790: DA    $7045    ;FILTER
4792: DA    $6DE0    ;PLAY
4794: DA    $6FD6    ;TEMPO
4796: DA    $6CC5    ;MOVSPR
4798: DA    $6C4E    ;SPRITE
479A: DA    $718F    ;SPRCOLOR
479C: DA    $58BC    ;RREG
479E: DA    $70C0    ;ENVELOPE
47A0: DA    $6BD6    ;SLEEP
47A2: DA    $A07D    ;CATALOG
47A4: DA    $A11C    ;DOPEN
47A6: DA    $A133    ;APPEND
47A8: DA    $A16E    ;DCLOSE
47AA: DA    $A1C7    ;BSAVE
47AC: DA    $A217    ;BLOAD
47AE: DA    $A2D6    ;RECORD
47B0: DA    $A361    ;CONCAT
47B2: DA    $A1A3    ;DVERIFY
47B4: DA    $A321    ;DCLEAR
47B6: DA    $76EB    ;SPRSAV
47B8: DA    $7163    ;COLLISION
47BA: DA    $796B    ;BEGIN
47BC: DA    $528E    ;BEND
47BE: DA    $72CB    ;WINDOW
47C0: DA    $7334    ;BOOT
47C2: DA    $71B5    ;WIDTH
47C4: DA    $7371    ;SPRDEF
47C6: DA    $4845    ;QUIT - NOT IMPLEMENTED
47C8: DA    $AA1E    ;STASH
47CA: .BYTE 0,0      ;SPARE ADDRESS FOR TOKEN $20
47CC: DA    $AA23    ;FETCH
47CE: .BYTE 0,0      ;SPARE ADDRESS FOR TOKEN $22
47D0: DA    $AA28    ;SWAP
47D2: DA    $4845    ;OFF - NOT IMPLEMENTED
47D4: DA    $77B2    ;FAST
47D6: DA    $77C3    ;SLOW

```

```

*-----*
*
*           NFUN
*
*           Numeric FUNCTION addresses
*
* NOTE:  The address shown is the actual address
*        of the routine that handles that particular
*        reserved word.
*-----*
    
```

	ADDRESS	FUNCTION
	-----	-----
47D8:	DA \$8C65	;SGN
47DA:	DA \$8CFB	;INT
47DC:	DA \$8C84	;ABS
47DE:	DA \$1218	;USR
47E0:	DA \$8000	;FRE
47E2:	DA \$84D0	;POS
47E4:	DA \$8FB7	;SQR
47E6:	DA \$8434	;RND
47E8:	DA \$89CA	;LOG
47EA:	DA \$9033	;EXP
47EC:	DA \$9409	;COS
47EE:	DA \$9410	;SIN
47F0:	DA \$9459	;TAN
47F2:	DA \$94B3	;ATN
47F4:	DA \$80C5	;PEEK
47F6:	DA \$8668	;LEN

```

*-----*
*
*                               SFUN
*
*                               String FUNCTION addresses
*
* NOTE: The address shown is the actual address
*        of the routine that handles that particular
*        reserved word.
*-----*
    
```

	ADDRESS	FUNCTION
	-----	-----
47F8:	DA \$85AE	;STR\$
47FA:	DA \$804A	;VAL
47FC:	DA \$8677	;ASC
47FE:	DA \$85BF	;CHR\$
4800:	DA \$85D6	;LEFT\$
4802:	DA \$860A	;RIGHT\$
4804:	DA \$861C	;MID\$
4806:	DA \$8182	;RGR
4808:	DA \$819B	;RCLR
480A:	.BYTE 0,0	;END OF BASIC FUNCTIONS FLAG

```

*-----*
*
*                               AFUN
*
*                               Added FUNCTION addresses
*
* NOTE: The address shown is the actual address
*        of the routine which handles that particular
*        reserved word.
*-----*
    
```

	ADDRESS	FUNCTION
	-----	-----
480C:	DA \$8203	;JOY
480E:	DA \$9B0C	;RDOT
4810:	DA \$8076	;DEC
4812:	DA \$8142	;HEX\$
4814:	DA \$80F6	;ERR\$
4816:	DA \$824D	;POT

```

4818: DA    $837C    ;BUMP
481A: DA    $82AE    ;PEN
481C: DA    $8397    ;RSPRITE
481E: DA    $831E    ;SPRITE
4820: DA    $8361    ;RSPCOLOR
4822: DA    $83E1    ;XOR
4824: DA    $8407    ;RWINDOW
4826: DA    $82FA    ;POINTER
    
```

```

*-----*
*
*                OPTAB
*
* (Operator TABulation priority codes and addresses
*   for the BASIC math operators.)
*
* NOTE: The address shown is the actual address of
*       the routine for that particular reserved word.*
*
* NOTE: The priority code is used to determine which
*       mathematical operation is done first, then
*       second, then third and so forth. The hierarchy*
*       of the different operators is as follows:
*
*       1. Expressions in parentheses
*       2. Exponential expressions (raise to a power)*
*       3. Multiplication and Division
*       4. Addition and Subtraction
*       5. Relational tests ( <, >, =, <>, <=, >= )
*       6. AND (logical operator)
*       7. OR (logical operator)
*
*       If two operators with the same priority code
*       are used in an expression, then the operations*
*       are completed as read from left to right.
*
*-----*
    
```

```

ADDRESS      OPERATORS
-----
    
```

```

4828: .BYTE  $79      ; +  Priority code
4829: DA    $8847    ; +  (ADDITION)
482B: .BYTE  $79      ; -  Priority code
482C: DA    $8830    ; -  (SUBTRACT)
482E: .BYTE  $7B      ; *  Priority code
482F: DA    $8A26    ; *  (MULTIPLY)
    
```

```

4831: .BYTE $7B      ; /  Priority code
4832: DA   $8B4B     ; /  (DIVIDE)
4834: .BYTE $7F      ; ^  Priority code
4835: DA   $8FC0     ; ^  (EXPONENT)
4837: .BYTE $50      ; AND Priority code
4838: DA   $4C88     ; AND (logical operator)
483A: .BYTE $46      ; OR  Priority code
483B: DA   $4C85     ; OR  (logical operator)
483D: .BYTE $7D      ; >  Priority code
483E: DA   $8FF9     ; >  (GREATER THAN)
4840: .BYTE $5A      ; =  Priority code
4841: DA   $792F     ; =  (EQUAL TO)
4843: .BYTE $64      ; <  Priority code
4844: DA   $4CB5     ; <  (LESS THAN)
4846: LDX  #40       ;UNIMPLEMENTED COMMAND error
4848: JMP  $4D3C     ;Error message handler
    
```

```

*****
*
*
*           ERRTAB
*
*           ERROR TABLE
*
* This area of memory contains a table of the error
* messages used by the BASIC operating system routines.*
*
* NOTE: The last character of each error message is
*       stored as a shifted character to indicate the
*       end of that error message.
*
*****
    
```

	ERROR MESSAGE	ERROR NUMBER
	-----	-----
484B:	TXT 'too many files'	;1
4859:	TXT 'file open'	;2
4862:	TXT 'file not open'	;3
486F:	TXT 'file not found'	;4
487D:	TXT 'device not present'	;5
488F:	TXT 'not input file'	;6
489D:	TXT 'not output file'	;7
48AC:	TXT 'missing file name'	;8
48BD:	TXT 'illegal device number'	;9
48D2:	TXT 'next without for'	;10
48E2:	TXT 'syntax'	;11
48E8:	TXT 'return without gosub'	;12

	ERROR MESSAGE	ERROR NUMBER
	-----	-----
48FC:	TXT 'out of data'	;13
4907:	TXT 'illegal quantity'	;14
4917:	TXT 'overflow'	;15
491F:	TXT 'out of memory'	;16
492C:	TXT 'undef'd statement'	;17
493D:	TXT 'bad subscript'	;18
494A:	TXT 'redim'd array'	;19
4957:	TXT 'division by zero'	;20
4967:	TXT 'illegal direct'	;21
4975:	TXT 'type mismatch'	;22
4982:	TXT 'string too long'	;24
4991:	TXT 'file data'	;25
499A:	TXT 'formula too complex'	;25
49AD:	TXT 'can't continue'	;26
49BB:	TXT 'undef'd function'	;27
49CB:	TXT 'verify'	;28
49D1:	TXT 'load'	;29
49D5:	TXT 'break'	;30
49DA:	.BYTE 0	;End of old BASIC error messages ;And start of added error messages ;for the C-128
49DB:	.BYTE \$A0	;Shifted space used as the last ;Character of the BREAK message
49DC:	TXT 'can't resume'	;31
49E8:	TXT 'loop not found'	;32
49F6:	TXT 'loop without do'	;33
4A05:	TXT 'direct mode only'	;34
4A15:	TXT 'no graphics area'	;35
4A25:	TXT 'bad disk'	;36
4A2D:	TXT 'bend not found'	;37
4A3B:	TXT 'line number too large'	;38
4A50:	TXT 'unresolved referance'	;39
4A64:	TXT 'unimplemented command'	;40
4A79:	TXT 'file read'	;41

```
*-----*
*
*                               FNDERR
*
*                               FiND ERRor
*
* On entry into this routine the accumulator holds
* the error number. On exiting this routine,
* locations $26, $27 holds the LSB and MSB
* respectively of the address that points to the
* first letter of the error message.
*-----*

4A82: TAX                ;Move the error number into the X register
4A83: LDY    #0          ;Zero index pointer
4A85: LDA    #$4B        ;LSB of the start of the error messages
4A87: STA    $26         ;Save the LSB
4A89: LDA    #$48        ;MSB of the start of the error messages
4A8B: STA    $27         ;Save the MSB
4A8D: DEX                ;Subtract 1 from the error number
4A8E: BMI    $4A9E       ;If the error number was 0 then exit
4A90: LDA    ($26),Y     ;Get a character from the error message table
4A92: PHA                ;Save it onto the stack temporarily
4A93: INC    $26         ;Add one to the LSB
4A95: BNE    $4A99       ;If <> 0, then branch
4A97: INC    $27         ;Add one to the MSB if LSB overflowed
4A99: PLA                ;Get the error message character off the stack
4A9A: BPL    $4A90       ;If it's not a shifted character then branch
4A9C: BMI    $4A8D       ;If it's a shifted character branch
4A9E: RTS                ;Exit routine
```

```
*****
*
*                               GONE
*
* This routine checks to see if any of the sprite
* collisions have occurred and if they have, then it
* checks to see which type of collision it was and
* then GOSUBs to the line number specified in the
* program. If no sprite collisions have occurred, then
* the routine executes the keyword. If there is no
* GOSUB specified, then ignore the collision.
*
*****
```



```

4A9F: JMP    ($0308)    ;Normal entry $4AA2
4AA2: BIT    $7F        ;In direct mode?
4AA4: BPL    $4AF0      ;If so, branch to pass collision testing
4AA6: LDA    $127F      ;Check to see if a sprite collision is currently
4AA9: BMI    $4AF0      ;being processed;if so, then branch to prevent
                          ;More than one collision processed at a time

```

```

*-----*
*
* This routine checks for a collision interrupt set for*
* any of the three possible types of collisions.      *
*
*-----*

```

```

4AAB: LDX    #2          ;Check the 3 possible SPRITE
4AAD: LDA    $1276,X    ;COLLISION interrupts
4AB0: BEQ    $4AED      ;Check until all 3 are done
4AB2: LDA    #0          ;Clear the interrupt flag
4AB4: STA    $1276,X    ;For the one that generated the interrupt
4AB7: LDA    $1279,X    ;Get the address of the line
4ABA: STA    $16         ;Number to 'GOSUB' to and
4ABC: LDA    $127C,X    ;Place it into the current
4ABF: STA    $17         ;Line number pointers
4AC1: TXA                    ;Save the current COLLISION
4AC2: PHA                    ;Type being processed
4AC3: LDA    $3D          ;Save the current CHRGET/GOT
4AC5: PHA                    ;Pointers onto the stack to
4AC6: LDA    $3E          ;Save the current program
4AC8: PHA                    ;Pointers for restoration
4AC9: LDA    $127F        ;Set bit 7 to indicate
4ACC: ORA    #$80         ;A COLLISION interrupt is
4ACE: STA    $127F        ;In progress
4AD1: JSR    $0380        ;Get the token for GOSUB and discard it
4AD4: JSR    $5A1D        ;Place GOSUB parameters onto the stack
4AD7: JSR    $59E2        ;Execute a preprocessed GOTO cmd
4ADA: JSR    $4AF6        ;Execute the BASIC statement
4ADD: LDA    $127F        ;Clear the flag for a
4AE0: AND    #$7F        ;COLLISION interrupt no longer in progress
4AE2: STA    $127F
4AE5: PLA                    ;Get the pointers back off
4AE6: STA    $3E          ;The stack to continue where
4AE8: PLA                    ;We left off before the
4AE9: STA    $3D          ;COLLISION in CHRGET/GOT
4AEB: PLA                    ;Get the last COLLISION
4AEC: TAX                    ;Tested if all three are not
4AED: DEX                    ;Checked then continue until
4AEE: BPL    $4AAD        ;Done

```

```

*-----*
*
*           Execute the Keyword
*
*-----*
    
```

```

4AF0: JSR    $0380    ;Get the next character
4AF3: JSR    $4B3F    ;And execute the keyword
    
```

```

*****
*
*           BASIC INTERPRETER LOOP
*
* This routine will check the STOP key and if the
* STOP key is depressed, this routine will abort any
* currently active function with a BREAK message. If
* the STOP key is not depressed and you are not in
* the DIRECT mode, then this routine will save the
* current line number for the CONTINUE command. Also
* save the stack pointer for later use.
*
*****
    
```

```

4AF6: JSR    $4BB5    ;Check the STOP key and abort if depressed
4AF9: BIT    $7F      ;Check to see if we are in the
4AFB: BPL    $4B03    ;Direct mode and if so, branch to avoid saving
                        ;The line number for the CONTINUE command
4AFD: JSR    $4B34    ;Save the TXTPTR for the CONTINUE command
4B00: TSX
4B01: STX    $82      ;And save it in OLDSTX
4B03: LDY    #0       ;Zero the index pointer
4B05: JSR    $03C9    ;Get the next character after the STOP command
4B08: BEQ    $4B0D    ;Branch if the END OF STATEMENT FLAG $00
4B0A: JMP    $4BAE    ;Check for ':' if not found, then it is an error
4B0D: BIT    $7F      ;Check to see if we are in the direct
4B0F: BPL    $4B31    ;Mode if we are, branch to print 'READY'
4B11: LDY    #2       ;Move the pointer over to the
4B13: JSR    $03C9    ;MSB of the LINE LINK and if
4B16: BEQ    $4B31    ;It's a zero (end of program), then branch
4B18: INY
4B19: JSR    $03C9    ;Line number LSB and save it as
4B1C: STA    $3B      ;The current BASIC line number LSB
4B1E: INY
4B1F: JSR    $03C9    ;Line number MSB and save it as
4B22: STA    $3C      ;The current BASIC line number MSB
    
```

```

4B24: TYA          ;Add the pointer to the LINE
4B25: CLC          ;Number to TXTPTR
4B26: ADC   $3D    ;Which will have CHRGOT/GET
4B28: STA   $3D    ;Point to the LINE NUMBER MSB
4B2A: BCC   $4B2E  ;So that the next character
4B2C: INC   $3E    ;Read by CHRGET/GOT will be a token
4B2E: JMP   $4A9F  ;Execute BASIC statement
4B31: JMP   $4D37  ;Print 'READY.' to the screen

```

```

*-----*
*
* This routine saves the TXTPTR number for the
* CONTInue command.
*
*-----*

```

```

4B34: LDA   $3D    ;Get the address to the
4B36: LDY   $3E    ;Current line number
4B38: STA   $1202  ;And save it for the
4B3B: STY   $1203  ;CONT command
4B3E: RTS

```

```

*****
*
* EXECUTE A BASIC STATEMENT
*
* If the trace mode flag at $116F is not set, then
* skip over the TRACE routine. The TRACE routine is
* also skipped if the TRACE mode flag is set and you
* are in the DIRECT mode.
*
*****

```

```

4B3F: BEQ   $4B3E  ;Exit if equal to zero
4B41: BIT   $116F  ;Trace mode on or off (0 = off)
4B44: BPL   $4B59  ;If off, then skip ahead
4B46: BIT   $7F    ;If on, check for direct mode $00 = DIRECT
4B48: BPL   $4B59  ;Skip trace if in the direct mode

```

```

*-----*
*
*                               PTRACE
*
*                               Program TRACE
*
* This routine prints the current line number in
* brackets. On entry, the accumulator holds the
* token to be executed.
*
* Example: [ line number ]
*-----*

```

```

4B4A: PHA                ;Save value temporarily
4B4B: LDA    #'['        ;
4B4D: JSR    $560C       ;Print a left bracket '['
4B50: JSR    $8E2E       ;Print the current line number
4B53: LDA    #']'       ;
4B55: JSR    $560C       ;Print a right bracket ']'
4B58: PLA

```

```

*****
*
* This routine checks for a dual token command which
* is prefixed by an $FE. It also checks for the MID$
* and GO command, plus it masks out the token values
* of $A3-$D4 which cannot be entered as a command by
* themselves. If any of the tokens $A3-$D4 do occur
* (exceptions: GO and MID$), a SYNTAX ERROR will be
* generated. On entry to this routine, the accumulator
* holds the value of the token.
*
*****

```

```

4B59: CMP    #$FE        ;Dual token BASIC COMMAND
4B5B: BEQ    $4B94       ;Yes, go to dual token execute
4B5D: CMP    #$CB        ;Is it 'GO' ?
4B5F: BNE    $4B64       ;No, then skip ahead
4B61: JMP    $5A3D       ;Yes it is the GO command,then check for
                        ;The GOTO or GO commands and execute
4B64: CMP    #$CA        ;Is it 'MID$' ?
4B66: BEQ    $4B8B       ;Yes ,then go execute
4B68: CMP    #$FB        ;If it is equal to or greater than
4B6A: BCS    $4BAB       ;$FB, then generate a 'SYNTAX ERROR'
4B6C: CMP    #$A3        ;Is it 'TAB(' ?
4B6E: BCC    $4B76       ;If less than, then skip ahead

```

```

4B70: CMP    #$D5      ;'ELSE'?
4B72: BCC    $4BAB     ;Less than, 'SYNTAX ERROR'

```

```

*-----*
*
* This routine first subtracts 50 from the token if it
* is larger than $D4 and then subtracts 128 from the
* token and checks to see if it is a letter. If it is
* a letter, then the routine jumps to the LET routine
* at $53C6. If the token was less than $A3, then the
* routine only subtracts 128 from the token.
*
*-----*

```

```

4B74: SBC    #$32      ;Subtract #$32
4B76: SEC                      ;From token in the accumulator
4B77: SBC    #$80      ;Then subtract #$80
4B79: BCS    $4B7E     ;To create a pointer, if carry is set, then skip
4B7B: JMP    $53C6     ;Carry isn't set so it's a character, goto the
                        ;LET routine

```

```

*-----*
*
* This routine first multiplies the token value by 2
* after the subtraction performed in the preceding
* routine, and uses the resulting value as an index
* to the table of command addresses at $46FC to find
* the correct address for the command. The MSB and
* LSB of the address are put on the stack and the
* routine, then exits by jumping to the CHRGET routine
* to get the next character and execute the command.
*
*-----*

```

```

4B7E: ASL                      ;Take value created above and multiply it by two
4B7F: TAY                      ;To create a pointer in the Y register to the
                        ;Command address
4B80: LDA    $46FC+1,Y        ;Get the MSB of the address
4B83: PHA                      ;Put it on the stack
4B84: LDA    $46FC,Y         ;LSB of the address - 1
4B87: PHA                      ;Put it on the stack
4B88: JMP    $0380           ;Get the next character and execute the command

```

```

*-----*
*
* The following routine is used to execute the MID$
* command if the MID$ command is used by itself.
*
* EXAMPLE: MID$(A$,1,1) = B$
*
*-----*

```

```

4B8B: LDA    #$59      ;MSB
4B8D: PHA                ;Put it on the stack
4B8E: LDA    #$00      ;LSB
4B90: PHA                ;Put it on the stack
4B91: JMP    $0380     ;Get the next character and execute the command

```

```

*-----*
*
* This routine is used to execute the dual token
* commands that are prefixed by an $FE. First the
* routine checks the character following the $FE and
* if it is a zero, the routine aborts with a SYNTAX
* ERROR. If it is not a zero, then the range is
* checked to see if the token is less than 2 or
* greater than 38. If it is, then a SYNTAX ERROR is
* generated. If the token is in the right range,
* then a value of 71 is added to the token value and
* the routine branches back to the routine preceding
* this routine that executes the command.
*
*-----*

```

```

4B94: JSR    $0380     ;Get the next character
4B97: BEQ    $4BAB     ;If it is a zero then generate a 'SYNTAX ERROR'
4B99: CMP    #2        ;Compare char. to the first dual token command
4B9B: BCC    $4BA5     ;If it is less, then generate a 'SYNTAX ERROR'
4B9D: CMP    #$27     ;Compare the character to the last dual token + 1
4B9F: BCS    $4BA5     ;If its value is greater, then generate a
                        ;'SYNTAX ERROR'
4BA1: ADC    #$47     ;Add #$47 to create the pointer to the command
                        ;Address
4BA3: BNE    $4B7E     ;Multiply times 2 and execute the command
4BA5: SEC                ;Set the carry bit to generate a 'SYNTAX ERROR'

```



```

*****
*
*                                     *
*                               CHKSTP *
*                                     *
*                               CHecK  *
*                               for the *
*                               SToP key *
*                                     *
*   This routine is called by various *
*   BASIC routines to check to see if *
*   the STOP key is depressed.  If the *
*   STOP key is depressed, then the *
*   'BREAK' error message will be *
*   printed. *
*                                     *
*****

```

```

4BB5: JSR   $9293      ;Check the STOP KEY
4BB8: BEQ   $4BBB      ;Branch if it is depressed
4BBA: RTS                ;Exit the routine if it is not depressed

```

```

*-----*
*
*   Check to see if the TRAP command is specified. *
*
*-----*

```

```

4BBB: LDY   $120C      ;Get the line number to execute on error
4BBE: INY                ;Add one and if equal to zero, then no line
                                ;Number to go to
4BBF: BEQ   $4BD0      ;Execute a BASIC END COMMAND

```

```

*-----*
*
*   A TRAP command was specified, so wait until the *
*   STOP key is released before executing the TRAP *
*   line to prevent another BREAK (STOP). *
*-----*

```

```

4BC1: JSR   $9293      ;Check the STOP KEY
4BC4: BEQ   $4BC1      ;And wait for it to be released
4BC6: LDX   #30        ;BREAK error
4BC8: JMP   $4D3C      ;ERROR MESSAGE HANDLER

```

```

*****
*
*                               BASIC STOP COMMAND *
*
*   Command syntax:  STOP *
*
*****

```



```

*   In order to execute a STOP command, the carry flag   *
*   is set and then a BASIC END command is executed.     *
*                                                         *
*****

```

```
4BCB: BCS    $4BCE        ;Branch past end flag and leave carry set
```

```

*****
*                                                         *
*                   BASIC END COMMAND                   *
*                                                         *
*   Command syntax:  END                               *
*                                                         *
*   To discern between the END command and the STOP    *
*   command, the carry flag is set for the STOP command *
*   and the carry flag is cleared for the END command. *
*   The END command is basically the same as the STOP  *
*   command in that they both stop the program        *
*   execution and both save the current line number   *
*   for the CONTinue command if you are not in the    *
*   DIRECT mode.  The only difference between the two  *
*   commands is that the STOP command prints a BREAK  *
*   message when executed and the END command doesn't. *
*                                                         *
*****

```

```

4BCD: CLC                ;Clear the carry for the BASIC END flag
4BCE: BNE    $4BF6        ;Left over from 64 code, not used here
4BD0: BIT    $7F          ;Check to see if in DIRECT mode or PROGRAM mode
4BD2: BPL    $4BE1        ;If in DIRECT mode, no line number branch needed
4BD4: JSR    $4B34        ;Save the pointers for next line number for CONT
4BD7: LDA    $3B          ;Current basic line number LSB
4BD9: LDY    $3C          ;Current basic line number MSB
4BDB: STA    $1200        ;Save
4BDE: STY    $1201        ;them
4BE1: PLA                    ;Pull the return address off of
4BE2: PLA                    ;The stack
4BE3: BCC    $4BF3        ;If the END flag is set, branch and don't print
                          ;'BREAK'
4BE5: JSR    $9281        ;Print 'BREAK' to the screen
4BE8: DFB    $0D,$0A      ;<cr>, 10=(LINE FEED)
4BEA: TXT    'break'
4BEF: DFB    $00          ;End of message pointer
4BF0: JMP    $4DAF        ;Print 'IN' followed by the current line number

```

```

*****
*
*           Jump to print the READY message.
*
*****
4BF3: JMP    $4D37      ;Print 'READY' to the screen
4BF6: RTS                ;Exit routine

*****
*
*   This routine has 3 basic checks to separate the
*   Numeric Functions, Added Basic Functions ,and String
*   Functions with more than one parameter. Once this
*   routine determines which one of the three types the
*   command falls under, this routine calls the proper
*   routines for executing that particular command. In
*   the case of the string functions, this routine will
*   obtain the proper parameters, then call the proper
*   routine that handles the string function.
*
*****

4BF7: CMP    #$CE      ;Dual token BASIC FUNCTION?
4BF9: BEQ    $4C54      ;Yes ,branch to get second token
4BFB: CMP    #$D5      ;Is it 'ELSE' ?
4BFD: BCS    $4BAB      ;If greater ,then 'SYNTAX ERROR'
4BFF: CMP    #$CB      ;If less ,is it 'GO' ?
4C01: BCC    $4C05      ;If less ,then skip next instruction
4C03: SBC    #1         ;Token - 1
4C05: PHA                ;Put token on stack
4C06: TAX                ;And in the X register
4C07: JSR    $0380      ;Get the next character
4C0A: CPX    #$D3      ;Compare the first token to 'INSTR'
4C0C: BEQ    $4C16      ;Skip other comparisons if so
4C0E: CPX    #$CB      ;Is it 'RGR' ?
4C10: BCS    $4C3B      ;If greater than, then go to function execute
4C12: CPX    #$C8      ;Compare to 'LEFT$'
4C14: BCC    $4C3B      ;If less than, go to the function

*-----*
*
*   This routine obtains the string function paramters.
*
*-----*

4C16: JSR    $7959      ;Check on '(' and get the next character

```

```

4C19: JSR   $77EF       ;Evaluate expression (FRMEVL)
4C1C: JSR   $795C       ;Check for a comma (CHKCOM)
4C1F: JSR   $77DD       ;Check for a string (CHKSTR)
4C22: PLA                       ;Pull the first token off the stack
4C23: CMP   #$D3         ;Is it 'INSTR' ?
4C25: BEQ   $4C80       ;Yes, then execute
4C27: TAX                       ;No, then store the token to the Y register
4C28: LDA   $67         ;MSB floating point accumulator (FAC1)
4C2A: PHA                       ;Put on stack
4C2B: LDA   $66         ;LSB floating point accumulator (FAC1)
4C2D: PHA                       ;Put on stack
4C2E: TXA                       ;Transfer the token to the accumulator
4C2F: PHA                       ;Save the accumulator onto the stack
4C30: JSR   $87F4       ;Convert the accumulator to integer, and return
                        ;The resulting integer value to the X register
4C33: PLA                       ;Get the accumulator from the stack
4C34: TAY                       ;Transfer the token to the Y register
4C35: TXA                       ;Stored the converted value to the accumulator
4C36: PHA                       ;Put it on to the stack
4C37: TYA                       ;
4C38: JMP   $4C3F       ;Execute the function

```

```

*-----*
*
* This routine is used to execute the numeric
* functions. This routine first evaluates the
* expression in parentheses and calculates the address
* of the function to be executed by subtracting 180
* from the token value and then multiplying that value
* by two. The routine then executes the function that
* is pointed to by the address that was calculated
* from the function's token value.
*
*-----*

```

```

4C3B: JSR   $7950       ;Evaluate the expression in parentheses
4C3E: PLA                       ;Get the token off the stack
4C3F: SEC                       ;Subtract 180
4C40: SBC   #$B4         ;And
4C42: ASL                       ;Multiply the result by 2
4C43: TAY                       ;To make a pointer in the Y register
4C44: LDA   $47D8+1,Y   ;MSB of the function address
4C47: STA   $58         ;JMP vector high byte for function evaluations
4C49: LDA   $47D8,Y     ;LSB of the function address
4C4C: STA   $57         ;JMP vector low byte for the function evaluations
4C4E: JSR   $0056       ;Evaluate the function
4C51: JMP   $77DA       ;Check for numeric

```

```

*-----*
*
* This routine checks the range on a dual token and
* if the value of the token is in range, the function
* represented by that token is executed. However, if
* the token value is out of range a SYNTAX ERROR
* results.
*
*-----*

```

```

4C54: JSR   $0380   ;Get the next byte
4C57: BEQ   $4C83   ;If it is equal to zero, then it is an error
4C59: CMP   #$0A    ;Token for 'POINTER'
4C5B: BEQ   $4C68   ;Skip if it is the token for 'POINTER'
4C5D: PHA                   ;Save it on to the stack if it is not the
                           ;Token for the POINTER command
4C5E: JSR   $0380   ;Get the next character
4C61: JSR   $7959   ;Check for an opening parenthesis
4C64: JSR   $77EF   ;Evaluate the expression
4C67: PLA                   ;Get the token back
4C68: CMP   #2      ;Compare it to token value for the 'POT' function
4C6A: BCC   $4C74   ;If it is less than, then it is an error
4C6C: CMP   #$0B    ;Compare to the largest possible dual token + 1
4C6E: BCS   $4C74   ;If it is larger, then it is an error
4C70: ADC   #$D1    ;Add #$D1 to make the pointer to the address
4C72: BNE   $4C3F   ;Branch to make the pointer and execute
4C74: SEC                   ;Carry set for error
4C75: JSR   $4C7D   ;Normally JuMPs to $4C78
4C78: BCS   $4C83   ;If the carry is set, then it represents an error
4C7A: JMP   $77DA   ;Check for numeric

```

```

*****
*
*                               ESCFNVEC
*
*                               ESCape FuNction command VECtor
*
* This routine is vectored through $02FC (IESCFNVEC)
* to enable the addition of new BASIC commands which
* have the first token value of $CE and the second
* token value between $02 and $0A. If the second value
* is not in this range a SYNTAX ERROR results. If you
* wish to add commands to the C-128 operating system,
* wedge into this vector and check for your own
* tokens. When this vectored routine is called, the
* carry flag is set and the second token value is in
* the accumulator. Also, note that this is a FUNCTION
*

```

```

* VECTOR, and is called by such statements as PRINT *
* pointer ($A8). This routine will return you to *
* RAM BANK 14. *
* *
*****

```

```

4C7D: JMP ($02FC) ;JuMPs TO $4C78
4C80: JMP $99C1 ;JuMPs to INSTR command
4C83: JMP $796C ;SYNTAX ERROR

```

```

*****
* *
* BASIC OR COMMAND *
* *
* Command syntax: < expression > OR < expression > *
* *
* To perform a logical OR of two numbers, a flag at $0D*
* is set to $FF (255) to invert the value to be Ored *
* before performing an AND command. *
* *
*****

```

```

4C86: LDY #$FF ;Flag for the OR command
4C88: DFB $2C ;MASK to bypass the OR flag

```

```

*****
* *
* BASIC AND COMMAND *
* *
* Command syntax: < expression > AND < expression > *
* *
* This routine is used to perform a logical AND or an *
* OR between two numbers. To perform OR a flag at $0D *
* is set to $FF to invert the value before ANDing it. *
* The value is then inverted to complete the OR. For *
* AND, the routine clears the flag at $0D so the value *
* isn't inverted before the AND command. This routine *
* converts the two numbers from floating point format *
* to integer form then performs the specified function.*
* *
*****

```

```

4C89: LDY #0 ;Flag for AND command
4C8B: STY $0D ;Save the flag value
4C8D: JSR $84B4 ;Change the first value to an integer
4C90: LDA $66 ;Get first value from floating point accumulator
4C92: EOR $0D ;If the flag is set for an OR, invert the value

```

```

4C94: STA   $09           ;Store the result
4C96: LDA   $67           ;MSB
4C98: EOR   $0D           ;If OR, then value will be inverted
4C9A: STA   $0A           ;Store the result
4C9C: JSR   $8C28        ;Move argument to floating point accumulator
4C9F: JSR   $84B4        ;Change second value to an integer
4CA2: LDA   $67           ;MSB
4CA4: EOR   $0D           ;If OR the value will be inverted
4CA6: AND   $0A           ;And with value
4CA8: EOR   $0D           ;If OR the value will be inverted back
4CAA: TAY                   ;Save the MSB into the Y register
4CAB: LDA   $66           ;LSB
4CAD: EOR   $0D           ;If OR the value will be inverted
4CAF: AND   $09           ;And with value
4CB1: EOR   $0D           ;If OR the value will be inverted back
4CB3: JMP   $793C        ;Convert the Y register and the accumulator
                                ;To floating point. The LSB is in the accumulator
                                ;And the MSB is in the Y register

```

```

*****
*
*           BASIC LESS THAN FUNCTION ( < )           *
*
* This routine first checks to see if the comparison *
* is for a numeric value or a string. If the comparison*
* is for a string, the String Comparison routine at *
* $4CCE is called. However, if the comparison is for a *
* numeric value, Floating Point Accumulator 1 (FAC1) is*
* compared to the Argument in Floating Point Accumula- *
* tor 2 (FAC2) and the result of the comparison is re- *
* turned to the accumulator and then transferred to the*
* Floating Point Accumulator. The three possible values*
* that could be returned to the accumulator after a *
* comparison are: $01(greater) $FF(less) $00 (equal). *
*
*****

```

```

4CB6: JSR   $77DE        ;Check if string or numeric
4CB9: BCS   $4CCE        ;Branch to String Comparison if it is a string
4CBB: LDA   $6F           ;Sign of the argument
4CBD: ORA   #$7F         ;To mask for AND
4CBF: AND   $6B           ;AND with value, if value is positive, drop bit 7
4CC1: STA   $6B           ;Store the value back
4CC3: LDA   #$6A         ;Address of the argument
4CC5: LDY   #0           ;MSB
4CC7: JSR   $8C87        ;Compare the argument with floating point
                                ;Accumulator;put the result in the accumulator

```

```

4CCA: TAX                ;Place result in the X register
4CCB: JMP    $4D01       ;Give the results to floating point accumulator

```

```

*****
*
*                STRING COMPARISON                *
*
* This routine first checks VALTYP at $0F to be sure *
* the comparison is for a string. To do a comparison of*
* two strings, the length and character values of the *
* first string are compared to the length and character*
* values of the second string. If the comparison type *
* specified by zero page location $14 is found to be *
* true, the value returned to the accumulator is $00. *
* If, however, the comparison specified by location $14*
* is false, the vlaue returned is $FF.                *
*
*****

```

```

4CCE: LDA    #0
4CD0: STA    $0F        ;Set data type
4CD2: DEC    $4F        ;To numeric
4CD4: JSR    $8781      ;Find the 1st string
4CD7: STA    $63        ;1st String length
4CD9: STX    $64        ;1st String address LSB
4CDB: STY    $65        ;1st String address MSB
4CDD: LDA    $6D        ;Pointer to 2nd string
4CDF: LDY    $6E        ;For comparision
4CE1: JSR    $8785      ;Find the 2nd string and return the length value
                        ;Of the string to the accumulator
4CE4: STX    $6D        ;Save the address of the 2nd
4CE6: STY    $6E        ;String
4CE8: TAX                ;Save the length of the 2nd string
4CE9: SEC                ;Subtract length of 1st string from 2nd string
4CEA: SBC    $63        ;And if the result
4CEC: BEQ    $4CF6      ;Is zero, then branch
4CEE: LDA    #1        ;For 1st string is shorter
4CF0: BCC    $4CF6      ;Carry is clear if the 1st string is shorter
                        ;Than the 2nd string
4CF2: LDX    $63        ;Length of the 1st string
4CF4: LDA    #$FF      ;For 1st string longer than the 2nd string, save
4CF6: STA    $68        ;The value of the result of the length comparison
4CF8: LDY    #$FF      ;
4CFA: INX                ;
4CFB: INY                ;Increment the pointer
4CFC: DEX                ;Decrement the length counter by one
4CFD: BNE    $4D06      ;If not zero yet, then continue

```

```

4CFF: LDX    $68           ;If zero, then get the length flag
4D01: BMI    $4D1E        ;If longer, then branch
4D03: CLC                    ;If equal to or shorter
4D04: BCC    $4D1E        ;Branch
4D06: LDA    #$6D         ;LSB of the indirect address of the 2nd string
4D08: JSR    $03AB        ;Get a byte of the 2nd string from BANK 1
4D0B: PHA                    ;Save it to the stack
4DOC: LDA    #$64         ;LSB of the indirect address of the 1st string
4DOE: JSR    $03AB        ;Get a byte of the 1st string from BANK 1
4D11: STA    $79           ;Save the value
4D13: PLA                    ;Get a byte of the 2nd string off the stack
4D14: CMP    $79           ;Compare it to the byte of 1st string
4D16: BEQ    $4CFB        ;If they are the same, then continue checking
4D18: LDX    #$FF         ;For string longer
4D1A: BCS    $4D1E        ;If carry is set, then ok longer, branch
4D1C: LDX    #1           ;If not, then set for shorter string
4D1E: INX                    ;Increment the length flag by one
4D1F: TXA                    ;Put the value in the accumulator
4D20: ROL                    ;Shift one bit left to indicate
4D21: AND    $14          ;What type of comparison took place
4D23: BEQ    $4D27        ;If zero, then OK
4D25: LDA    #$FF         ;For no match
4D27: JMP    $8C68        ;Result to the floating point accumulator
4D2A: JSR    $9281        ;Print the following text
4D2D: DFB    13           ;Print a <cr>
4D2E: TXT    'ready.'
4D34: DFB    13           ;Print a <cr>
4D35: DFB    0            ;Until the end of text flag is seen
4D36: RTS

```

```

*****
*
*                      READY
*
* This routine will print 'READY' to the currently
* active output device, enable the KERNAL error
* messages, and disable the control messages. The
* routine then proceeds to the MAIN BASIC LOOP.
*
*****

```

```

4D37: LDX    #$80           ;Set bit 7 to fall through to warm start
4D39: .BYTE  $2C           ;Mask (BIT instruction)
4D3A: LDX    #16           ;OUT OF MEMORY error

```



```

*****
*
*                               ERROR
*
* This routine is vectored through location $0300.
* On entry into this routine, the X register holds
* the ERROR number. If bit 7 of this value is set,
* this routine will print 'READY' to the current
* output device, enable the KERNAL error messages,
* disable the KERNAL control messages, and proceed to
* execute the MAIN routine. If bit 7 is not set, this
* indicates that there is indeed an ERROR message to
* be printed. This routine will then save the ERROR
* number for the TRAP command and the system variable
* ER. Then a test is made to see if you are in the
* DIRECT or the PROGRAM mode. If you are in the
* DIRECT mode, a branch is made to the ERROR 1
* routine. If you are in the PROGRAM mode with a
* program running, the line number where the error
* occurred and the address of the end of line flag
* of the line prior to the line that the error
* occurred in is saved. A check is then made to see
* if a line number was specified for the TRAP command.
* If a line number has been specified, that line is
* then executed. However, if there is no line number
* specified for the TRAP command, this routine
* branches to the ERROR 1 routine.
*
*****

```

```

4D3C: JMP    ($0300)    ;Normal entry $4D3F
4D3F: STA    $FF03      ;Enable BANK 14
4D42: TXA                    ;Transfer the error number into the accumulator
4D43: BMI    $4DB7      ;If bit 7 is set then print 'READY'
4D45: STX    $1208      ;Save the error number for the TRAP command and
                        ;The system variable ER
4D48: BIT    $7F        ;Check to see if in direct mode
4D4A: BPL    $4D7C      ;Branch if in direct mode
4D4C: LDY    #1         ;Index pointer
4D4E: LDA    $003B,Y    ;Current basic line number
4D51: STA    $1209,Y    ;Save as last error number
4D54: LDA    $1202,Y    ;Save it for the CONT command
4D57: STA    $120E,Y    ;Pointer to the end of line flag ($00) of the
                        ;Line the error occurred in
4D5A: DEY                    ;Subtract one from the index pointer
4D5B: BPL    $4D4E      ;Do two bytes
4D5D: LDY    $120C      ;Line number to goto on ERROR

```

4D60: INY ;Increment the line number by one  
4D61: BEQ \$4D7C ;No line number, print the error message  
4D63: DEY ;Decrement the line number by one  
4D64: STY \$17 ;Save the line number MSB  
4D66: STY \$120D ;Save the pointer for the TRAP command  
4D69: LDY \$120B ;Line number to execute on system errors  
4D6C: STY \$16 ;Save the line number LSB  
4D6E: LDX #\$FF  
4D70: STX \$120C ;Clear the line number to be executed on ERRORS  
4D73: LDX \$82 ;Old stack pointer  
4D75: TXS ;Transfer the old stack pointer back onto stack  
4D76: JSR \$59FB ;Program pointer to line number  
4D79: JMP \$4AF6 ;To BASIC INTERPRETER LOOP

```
*****  
*                                     *  
*                               ERROR 1                               *  
*                                     *  
* This routine will be called by the ERROR routine if *  
* an error has occurred in direct mode or if an error *  
* has occurred in the program mode when no line number *  
* is specified for the TRAP command. When called, this *  
* routine will subtract one from the error number *  
* stored in the X register, clear all open channels, *  
* make the keyboard the current input device, and then *  
* print the error message to the current output device.*  
*                                     *  
*****
```

4D7C: DEX ;Error number-1  
4D7D: TXA ;Place the error value in the accumulator  
4D7E: JSR \$4A82 ;Move the pointer to the first character of  
;The error message  
4D81: JSR \$926F ;Basic CLEAr CHannels  
4D84: LDA #0 ;Set the KEYBOARD as the  
4D86: STA \$15 ;Current active input device  
4D88: BIT \$D7 ;In 40/80 column mode?  
4D8A: BMI \$4D8E ;Branch if in 80 column mode  
4D8C: STA \$D8 ;Set flag for TEXT MODE  
4D8E: JSR \$5598 ;Print a return to the screen  
4D91: JSR \$560A ;Print a question mark  
4D94: LDY #0 ;Index pointer  
4D96: LDA (\$26),Y ;Character of the error message  
4D98: PHA ;Place it onto the stack  
4D99: AND #\$7F ;Print the characters that are not shifted  
4D9B: JSR \$560C ;Output the error message  
4D9E: INY ;Increment the pointer to next text character

```
4D9F: PLA ;Check bit 7 for end of text flag
4DA0: BPL $4D96 ;If the flag is not set, then branch
4DA2: JSR $5238 ;Done, initialize BASIC-pointers
4DA5: JSR $9281 ;Print the following message
4DA8: TXT ' error' ;To the screen until
4DAE: .BYTE $00 ;The end of text flag is seen
4DAF: LDY $3C ;Current basic line number high
4DB1: INY ;Move pointer to next line
4DB2: BEQ $4DB7 ;Line number? No then 'READY'
4DB4: JSR $8E26 ;Print text ' IN' and line number
4DB7: JSR $4D2A ;Print ' ready.' to the current output device
4DBA: LDA #$80 ;Set bit 7 to enable KERNAL error messages
4DBC: JSR $FF90 ;and disable KERNAL control messages
4DBF: LDA #0 ;Set the current mode
4DC1: STA $7F ;DIRECT MODE
```

```
*****
*
*                                MAIN
*
* This routine is vectored through location $0302.
* This routine gets a character from the keyboard and
* places it into the BASIC Input Buffer. This
* process continues until either 161 characters have
* been entered or the RETURN key is pressed. If over
* 161 characters are entered a 'STRING TOO LONG' error
* will occur. However, when the RETURN key is pressed,
* this routine checks to see if there is a line number
* associated with the line being inputted. If there is
* a line number, the routine branches to the routine
* that tokenizes and stores a program line. If a line
* number is not found, then the routine branches to the
* routine that executes a direct mode command.
*
*****
```

```
4DC3: JMP ($0302) ;Normal entry $4DC6
4DC6: LDX #$$$
4DC8: STX $3C ;Current basic line number high
4DCA: JSR $4F93 ;Get input from the keyboard
4DCD: STX $3D ;Save the current pointers to keyboard
4DCF: STY $3E
4DD1: JSR $0380 ;Get the first character from keyboard
4DD4: TAX ;Transfer it into the X register
4DD5: BEQ $4DC3 ;If there are no more entries, branch
4DD7: BCC $4DE2 ;If carry = 0 then not a character
4DD9: JSR $430A ;Convert to token
```

```

4DDC: JSR   $0386      ;Get the last character back
4DDF: JMP   $4AF3      ;Execute the keyword

```

```

*****
*
*                               *
*                MAIN 1        *
*                               *
* This routine is called every time the user presses *
* the RETURN key over a BASIC program line to enter it.*
* First the ASCII line number is converted to its HEX *
* equivalent. The routine at $430A is then called to *
* tokenize the rest of the line. After this has been *
* accomplished, a search of the memory is made for *
* the line number of the line being entered. If the *
* line number is found, the carry flag is set and the *
* routine branches to the DELIN routine which deletes *
* the old line and replaces it with the new line. If *
* the line number cannot be found, the routine branches*
* to the INLIN routine which inserts the new line. *
*
* NOTE: THE REGISTERS ARE NOT PRESERVED *
*
*****

```

```

4DE2: JSR   $50A0      ;Convert to line number
4DE5: JSR   $430A      ;Convert to token
4DE8: STY   $0D        ;Save the index pointer to the keyboard
4DEA: JSR   $5064      ;Search for the line number
4DED: BCC   $4E6A      ;If the carry is clear then insert the line

```

```

*****
*
*                               *
*                DELIN        *
*                               *
*                DElete LINE  *
*                               *
* This routine deletes the BASIC program line whose *
* address is contained in locations $61, $62. These lo-*
* cations point to the link to the next line. This rou-*
* tine also updates the program pointers at $1210/$1211*
* then proceeds to the INSLIN routine which inserts a *
* program line into a BASIC program. *
*
* NOTE: THE REGISTERS ARE NOT PRESERVED *
*
*****

```

---

```

4DEF: LDY #0
4DF1: JSR $42EC ;Get the LSB of link to next line
4DF4: SEC ;For subtraction
4DF5: SBC $61 ;Subtract the LSB of the present line address
4DF7: SEC
4DF8: SBC #$04 ;Subtract offset of 4 for links & line number
4DFA: SBC $0D ;And subtract the index
4DFC: BCS $4E1A ;If the carry is set, then ok
4DFE: EOR #$FF ;Invert if not
4E00: ADC #1 ;Add one
4E02: LDY $1211 ;Load Y register with end of BASIC bank 0 MSB
4E05: ADC $1210 ;Add the calculated value to end of BASIC LSB
4E08: BCC $4E0B ;If the carry is clear, then branch
4EOA: INY ;If not then increment the MSB by one
4EOB: CPY $1213 ;Compare to the MSB BASIC limit
4EOE: BCC $4E1A ;If less than, then ok
4E10: BNE $4E17 ;If greater than, then OUT OF MEMORY
4E12: CMP $1212 ;If equal to, then compare to the LSB limit
4E15: BCC $4E1A ;If less than, then ok
4E17: JMP $4D3A ;Error 16 'OUT OF MEMORY'
4E1A: LDY #1 ;Index to the next byte
4E1C: JSR $42EC ;Get the MSB of next line link
4E1F: STA $25 ;Save
4E21: LDA $1210 ;Get the LSB of the end of BASIC
4E24: STA $24 ;Save as the LSB
4E26: LDA $62 ;Get the MSB of line number address
4E28: STA $27 ;Save
4E2A: DEY ;Index minus 1
4E2B: JSR $42EC ;Get the LSB of the line number from BANK 0
4E2E: CLC
4E2F: SBC $61 ;Subtract 1+ the LSB line address
4E31: EOR #$FF ;Invert
4E33: CLC
4E34: ADC $1210 ;Add to the end of BASIC LSB
4E37: STA $1210 ;Store it back
4E3A: STA $26 ;And in area for line number
4E3C: LDA $1211 ;MSB of end of BASIC
4E3F: ADC #$FF ;To set carry
4E41: STA $1211 ;Store it back
4E44: SBC $62 ;Subtract the MSB of the line number address
4E46: TAX ;Save it in the X register as the counter
4E47: SEC
4E48: LDA $61 ;LSB of line number address
4E4A: SBC $1210 ;Subtract the LSB of the end of BASIC
4E4D: TAY ;Save it in the Y register
4E4E: BCS $4E53 ;If the carry is set, then skip
4E50: INX ;Increment the MSB in the X register

```

```

4E51: DEC   $27           ;Decrement the MSB in $27
4E53: CLC
4E54: ADC   $24           ;Add to LSB of work area
4E56: BCC   $4E5B         ;If the carry is clear, then skip
4E58: DEC   $25           ;Decrement the MSB by one
4E5A: CLC
    
```

```

*-----*
*
* This routine takes all of the program lines that
* follow the line to be deleted and moves them up the
* number of bytes of the length of the deleted line.
* The routine then proceeds to the INSLIN routine.
*
*-----*
    
```

```

4E5B: JSR   $4305         ;Get a byte from the line after the deleted one
4E5E: STA   ($26),Y       ;Store where the deleted line originally was
4E60: INY
4E61: BNE   $4E5B         ;Branch until all of line is moved
4E63: INC   $25           ;MSB
4E65: INC   $27           ;MSB of the placement address
4E67: DEX
4E68: BNE   $4E5B         ;Branch until zero
    
```

```

*****
*
*                               INSLIN
*
*                               INSert LINE
*
* This routine first initializes the BASIC string and
* stack pointers then calls the LNKPRG routine to re-
* link the BASIC program lines. It then checks the end
* of program pointer at $1210/$1211 and generates an
* 'OUT OF MEMORY' error if there isn't enough room to
* insert the new line. If there is enough room, this
* routine inserts the new program line pointed to by
* $3D, $3E and calls the LNKPRG and RUNC routines to
* relink the program lines. It then updates TXTPTR
* to point to the start of program, and exits.
*
* NOTE: THE REGISTERS ARE NOT PRESERVED
*
*****
    
```

---

```

4E6A: JSR   $5238      ;Initialize the BASIC-pointers
4E6D: JSR   $4F4F      ;Update the line links
4E70: LDY   #0         ;Initialize the index
4E72: LDA   ($3D),Y    ;Any more characters in the buffer?
4E74: BNE   $4E79      ;No, then continue
4E76: JMP   $4DD5      ;Yes, then branch back
4E79: CLC
4E7A: LDA   $1210      ;End of BASIC LSB
4E7D: LDY   $1211      ;End of BASIC MSB
4E80: STA   $5C        ;SAVE the LSB
4E82: STY   $5D        ;SAVE the MSB
4E84: ADC   $0D        ;Add the index
4E86: BCC   $4E89      ;If the carry is clear, then skip
4E88: INY
4E89: CLC
4E8A: ADC   #$04       ;Add 4 for links and line number
4E8C: BCC   $4E8F      ;If the carry is clear, then skip
4E8E: INY
4E8F: STA   $5A        ;Store the LSB
4E91: STY   $5B        ;And store the MSB
4E93: CPY   $1213      ;Compare MSB + the carry to the BASIC limit MSB
4E96: BCC   $4EA2      ;If less than, then OK
4E98: BNE   $4E9F      ;If greater than but not equal to, then error
4E9A: CMP   $1212      ;If equal to, compare the LSBs
4E9D: BCC   $4EA2      ;If they are less, then continue
4E9F: JMP   $4D3A      ;Error 16 'OUT OF MEMORY'
4EA2: STA   $1210      ;Store the LSB
4EA5: STY   $1211      ;And store the MSB
4EA8: SEC
4EA9: LDA   $5C        ;Unmodified LSB end of BASIC
4EAB: SBC   $61        ;Subtract the LSB of the line address from it
4EAD: STA   $24        ;And save
4EAF: TAY
4EB0: LDA   $5D        ;Unmodified MSB end of BASIC
4EB2: SBC   $62        ;Subtract the MSB line address from it
4EB4: TAX
4EB5: INX
4EB6: TYA
4EB7: BEQ   $4EDE      ;If zero, then branch
4EB9: LDA   $5C        ;End of BASIC LSB
4EBB: SEC
4EBC: SBC   $24        ;Subtract the LSB of the calculated address
                        ;To make the counter
4EBE: STA   $5C        ;And store it back
4EC0: BCS   $4EC5      ;If the carry is set, then OK
4EC2: DEC   $5D        ;If the carry is clear, decrement the MSB by one
4EC4: SEC

```

```

4EC5: LDA   $5A           ;LSB + index + 4
4EC7: SBC   $24           ;Subtract the address
4EC9: STA   $5A           ;AND SAVE BACK
4ECB: BCS   $4ED6
4ECD: DEC   $5B
4ECF: BCC   $4ED6

```

```

*-----*
*
* This routine takes the program lines after the area
* to be opened for line insertion and moves them down
* the number of bytes in $0D+4 to provide a place to
* put the new program line.
*
*-----*

```

```

4ED1: JSR   $42DD         ;Get byte from $5c,$5d of the last program line
4ED4: STA   ($5A),Y       ;And store it $0d+4 bytes down in memory
4ED6: DEY
4ED7: BNE   $4ED1
4ED9: JSR   $42DD         ;Get the byte from $5c,$5d (y = 0)
4EDC: STA   ($5A),Y
4EDE: DEC   $5D           ;MSB of the address to get from
4EE0: DEC   $5B           ;MSB of the address to store to
4EE2: DEX
4EE3: BNE   $4ED6         ;Continue until the counter is 0
4EE5: LDY   #0
4EE7: LDA   #1
4EE9: STA   ($61),Y       ;At line links $01,$01
4EEB: INY
4EEC: STA   ($61),Y       ;Line link MSB
4EEE: INY                 ;To next byte
4EEF: LDA   $16           ;Get the line number low byte
4EF1: STA   ($61),Y       ;Store line number low
4EF3: LDA   $17           ;Get the high byte of the line number
4EF5: INY                 ;Index to next byte
4EF6: STA   ($61),Y       ;Store high byte of line number
4EF8: CLC
4EF9: LDA   $61           ;Low byte of the line number address
4EFB: ADC   #$04           ;plus 4
4EFD: STA   $61           ;Store as the address to the line
4EFF: BCC   $4F03
4F01: INC   $62           ;If the carry is set, then increment the MSB
4F03: LDY   $0D           ;Index to length
4F05: DEY                 ;minus 1
4F06: LDA   ($3D),Y       ;From line
4F08: STA   ($61),Y       ;Store the line in the allocated area

```



```

4F0A: DEY                               ;Counter to the number of bytes - 1
4F0B: CPY   #$FF
4F0D: BNE   $4F06                       ;Continue until Y = $ff
4F0F: JSR   $4F4F                       ;Update the line links
4F12: JSR   $5254                       ;Reset the TXTPTR to beginning of text
4F15: LDA   $74                         ;Get the offset value for the AUTO command
4F17: ORA   $75                         ;Combine it with the present value
4F19: BEQ   $4F4C                       ;If equal to zero, then branch
4F1B: LDA   $16                         ;Get the line nubmer high byte
4F1D: CLC
4F1E: ADC   $74                         ;Offset value for the AUTO command
4F20: STA   $65                         ;Store
4F22: LDA   $17                         ;Get the high byte of the line number
4F24: ADC   $75                         ;Add to auto value high
4F26: BCS   $4F4C
4F28: CMP   #$FA                       ;Compare it to the highest value allowed
4F2A: BCS   $4F4C                       ;Greater than, then branch
4F2C: STA   $64                         ;If not then store
4F2E: LDX   #$90
4F30: SEC                               ;For positive
4F31: JSR   $8C75                       ;Convert to floating point
4F34: JSR   $8E42                       ;Floating Point Accumulator to ASCII

```

```

*****
*
*                               *
*               LINEXE         *
*
*               LINe EXECute   *
*
* This routine retrieves text starting from location *
* $0101 and transfers it to the keyboard buffer at *
* $034A. It will continue until 255 characters are *
* read or until a $00 is found. The routine then *
* proceeds to the MAIN routine to convert the text to *
* tokens and to execute the line. *
*
*****

```

```

4F37: LDX   #0
4F39: LDA   $0101,X                   ;Get a byte from work area
4F3C: BEQ   $4F44                   ;If equal to zero, then exit
4F3E: STA   $034A,X                   ;Keyboard buffer
4F41: INX
4F42: BNE   $4F39
4F44: LDA   #$1D                     ;CURSOR RIGHT ONE SPACE
4F46: STA   $034A,X                   ;Keyboard buffer
4F49: INX

```

```
4F4A: STX  $D0          ;Index to keyboard buffer queue
4F4C: JMP  $4DC3        ;Convert to token and execute
```

```
*****
*
*                               LNKPRG                               *
*
*                               LiNK PRoGram                         *
*
* This routine first checks to see if the first two                 *
* line links are zeros.  If they are, then the routine             *
* exits.  If they are not zeros, then the routine                   *
* counts through the program lines and updates each of             *
* the line links.                                                  *
*
* NOTE: THE REGISTERS ARE NOT PRESERVED                             *
*
*****
```

```
4F4F: LDA  $2D          ;Point to the start of BASIC text (low byte)
4F51: LDY  $2E          ;Point to the start of BASIC text (high byte)
4F53: STA  $24          ;Into HELP POINTER INDEX 1
4F55: STY  $25
4F57: CLC              ;Clear the Carry flag
4F58: LDY  #0           ;Index pointer
4F5A: JSR  $4305        ;Get the LSB of the first line link
4F5D: BNE  $4F65        ;Not equal to zero, then branch ahead
4F5F: INY              ;Move over to the MSB of the line link
4F60: JSR  $4305        ;Get the MSB
4F63: BEQ  $4F92        ;If it's a zero, then branch
4F65: LDY  #4           ;Skip over the line link and the line number
4F67: INY              ;Move over to the next character
4F68: JSR  $4305        ;Get the character
4F6B: BNE  $4F67        ;Not the end of the line, then branch
4F6D: INY              ;Move over to the next character
4F6E: TYA              ;Save the index pointer
4F6F: ADC  $24          ;Update the previous line link
4F71: TAX              ;Move the value into X register
4F72: LDY  #0           ;Index pointer
4F74: STA  ($24),Y      ;Update the new line link
4F76: TYA              ;Zero the accumulator
4F77: ADC  $25          ;Add the carry to the MSB
4F79: INY              ;Increment the index pointer
4F7A: STA  ($24),Y      ;Save the MSB of the line link
4F7C: STX  $24          ;Save the LSB to the next
4F7E: STA  $25          ;BASIC line link LSB
4F80: BCC  $4F58        ;If it's not the end of the line, branch
```

```

*-----*
*
* This routine updates the end of program pointers
* located at $1210, $1211 from the values stored in
* $24, $25.
*
*-----*
    
```

```

4F82: CLC           ;Clear the carry flag for addition
4F83: LDA   $24     ;Get the end of the
4F85: LDY   $25     ;Program pointers
4F87: ADC   #2      ;Move past the MSB of the last line link
4F89: BCC   $4F8C   ;No carryover from addition, then branch
4F8B: INY           ;Add one to the MSB
4F8C: STA   $1210   ;Save the end of the program
4F8F: STY   $1211   ;Pointers for SAVE etc. commands
4F92: RTS           ;Exit routine
    
```

```

*****
*
*                               INLIN
*
*                               INput a LINE
*
* This routine waits for a key to be pressed then puts
* the value of that key in the input buffer at $0200.
* This continues until the RETURN key is pressed.
*
* NOTE: THE REGISTERS ARE NOT PRESERVED
*
*****
    
```

```

4F93: LDX   #0      ;Index pointer
4F95: JSR   $90E5   ;Get the next character from the keyboard
4F98: CMP   #13     ;Return key pressed?
4F9A: BNE   $4F9F   ;Yes, then branch
4F9C: JMP   $558B   ;Place delimiter at end of buffer
4F9F: STA   $0200,X ;Place in input buffer
4FA2: INX           ;Increment the index pointer
4FA3: CPX   #$A1    ;At the end of the buffer?
4FA5: BCC   $4F95   ;No, then branch
4FA7: JMP   $A5ED   ;STRING TOO LONG ERROR
    
```

```

*****
*
*                               FNDFOR
*
*
    
```

```

*                               FIND FOR                               *
*                               *                                     *
*          Stack routine for GOSUB, FOR-NEXT, and DO                 *
*                               *                                     *
* This routine is for the FOR-NEXT, GOSUB, and DO com- *
* mands. The routine is entered with the value of the *
* token to be searched for in the accumulator. The rou- *
* tine then transfers the pseudo stack pointer from *
* $7D/$7E to $3F/$40 and checks to see if the value *
* is $09FF which is the highest address value of the *
* stack. If it is equal to $09FF, then the stack is *
* empty and the routine exits. If the stack pointer *
* value is not $09FF, then the routine compares the *
* last value on the stack with the $81, the token value *
* for FOR. If it is equal, then the routine checks the *
* range of the address of the FOR variable. If it is *
* v$FF00 or greater, then the routine is exited. If the *
* value of the FOR variable is less than $FF00, then *
* the routine adds 18 to the stack pointer, checks the *
* range of the stack pointer and exits the routine. If *
* the last value on the stack is not the token value *
* for FOR and the value to be searched for is not *
* pointed to by the stack pointer, the routine adds 5 *
* to the stack pointer, checks the range of the stack *
* pointer and exits the routine. *
*
*****

```

```

4FAA: STA  $02          ;Save the character to be searched for
4FAC: JSR  $5047       ;Transfer the stack pointer from $7D/$7E TO
                        ;$3F/$40
4FAF: LDA  $3F          ;Compare the LSB of the stack pointer
4FB1: CMP  #$FF         ;To the LSB of the highest value ($09FF)
4FB3: BNE  $4FBB       ;If not $FF, then branch
4FB5: LDA  $40          ;If it is $FF, then check the MSB
4FB7: CMP  #$09        ;Compare to MSB of highest address allowed
                        ;($09XX)
4FB9: BEQ  $4FFB       ;If it is equal to 9, then exit
4FBB: STA  $FF03       ;If not, then enable BANK 14
4FBE: LDY  #0
4FC0: LDA  $02          ;Get the value to be searched for
4FC2: CMP  #$81        ;Is it the FOR command ?
4FC4: BNE  $4FE1       ;Not FOR, so look for another character
4FC6: CMP  ($3F),Y     ;If it is, compare it to the value on the stack
4FC8: BNE  $4FFD       ;If it is not equal, then exit
4FCA: LDY  #2          ;Index to the MSB of the 'FOR' variable
4FCC: LDA  $4C         ;MSB

```

```

4FCE:  CMP   #$FF           ;Compare to highest allowed value ($FF00)
4FD0:  BEQ   $4FFD          ;If equal, then exit
4FD2:  CMP   ($3F),Y        ;If not equal, compare to MSB address to 'FOR'
4FD4:  BNE   $4FDD          ;If not equal, then branch
4FD6:  DEY                       ;If equal, then decrement counter
4FD7:  LDA   $4B             ;Get the LSB of FOR-NEXT variable address
4FD9:  CMP   ($3F),Y        ;Compare to LSB of 'FOR' variable
4FDB:  BEQ   $4FFD          ;If equal, then exit
4FDD:  LDX   #$12           ;If not, then add 18 to the stack pointer
4FDF:  BNE   $4FEF          ;Unconditional branch
4FE1:  LDA   ($3F),Y        ;Get byte from the stack
4FE3:  CMP   $02            ;Compare to the search value
4FE5:  BEQ   $4FFD          ;If equal, then exit
4FE7:  LDX   #$12           ;
4FE9:  CMP   #$81           ;If not equal then compare to 'FOR' token
4FEB:  BEQ   $4FEF          ;If equal, then add 18 to the stack pointer
4FED:  LDX   #$05           ;If not, then add 5 for 'GOSUB'
4FEF:  TXA                       ;
4FF0:  CLC                       ;
4FF1:  ADC   $3F            ;Add the value to the LSB of the stack pointer
4FF3:  STA   $3F            ;Save it
4FF5:  BCC   $4FAF          ;If the carry is clear, then skip
4FF7:  INC   $40            ;If not, then increment the MSB
4FF9:  BNE   $4FAF          ;And check the range of the address
4FFB:  LDY   #1             ;
4FFD:  RTS                       ;Exit

```

```

*****
*
*                               CHKSTK
*
*                               Check STack space
*
* This routine is ensure there is enough room on the
* pseudo stack for the number of bytes in memory.
* If there is not enough room on the stack, an 'OUT OF
* MEMORY' error will occur. If there is enough room,
* the pseudo stack pointer is decremented by the value
* that is in the accumulator. The only routines that
* use $4FFE as entry points are GOSUB, FOR and DO.
*
*****

```

```

4FFE:  EOR   #$FF           ;Invert number of bytes to free on stack
5000:  SEC                       ;
5001:  ADC   $7D             ;Add A + 1 to LSB of stack address
5003:  STA   $7D             ;Store it as the stack pointer LSB

```



```

5038: PLA                ;LSB of the address back
5039: CPY   $36          ;Check again
503B: BCC   $5043        ;If the Y register is less than $36, then exit
503D: BNE   $5044        ;If the Y register is greater than $36, then
                        ;'OUT OF MEMORY' error
503F: CMP   $35          ;If the accumulator is equal to, then check
                        ;The LSB of the string pointer
5041: BCS   $5044        ;If it is greater than, 'OUT OF MEMORY' error
5043: RTS
5044: JMP   $4D3A        ;OUT OF MEMORY error
    
```

```

*****
*
* Transfer the pseudo stack pointer from $7D, $7E to
* $3F, $40. (Top of run time Stack - Utility Pointer)
*
*****
    
```

```

5047: LDA   $7D
5049: STA   $3F
504B: LDA   $7E
504D: STA   $40
504F: RTS
    
```

```

*****
*
* Transfer the pseudo stack pointer from $3F, $40 to
* $7D, $7E. (Utility Pointer - Top of runtime Stack)
*
*****
    
```

```

5050: LDA   $3F
5052: STA   $7D
5054: LDA   $40
5056: STA   $7E
5058: RTS
    
```

```

*****
*
* Add the value in the Y register to the pseudo
* stack pointer. (TOS = TOS + Y register)
*
*****
    
```

```

5059: TYA                ;Transfer the value into the accumulator
505A: CLC                ;Clear the carry for addition
505B: ADC   $7D          ;Add the value to the pseudo stack pointer LSB
    
```

```

505D: STA  $7D      ;Save it
505F: BCC  $5063   ;If no overflow resulted, then branch
5061: INC  $7E      ;Add one to the MSB
5063: RTS                ;Exit

*****
*
*                               FNDLIN
*
*                               FiND LIne Number
*
* This routine will search through a program trying to
* match up the line number in $16, $17 (which is in
* integer format). If a match is found, locations $61,
* $62 will contain the address which points to the line
* link for that line. The carry flag will be cleared if
* no match was found. On entry to this routine, loca-
* tions $16, $17 contain the line number to be searched
* for. On exit from this routine, locations $61, $62
* will point to the line link.
*
*****

5064: LDA  $2D      ;LSB of the start of BASIC TEXT
5066: LDX  $2E      ;MSB
5068: LDY  #1       ;Index pointer
506A: STA  $61      ;Save the LSB of the start of BASIC TEXT
506C: STX  $62      ;Save the MSB
506E: JSR  $42EC    ;Get a byte from the line (MSB of line link)
5071: BEQ  $509E    ;If it's the end of the line flag ($00), exit
5073: INY                ;Increment the index pointer
5074: INY                ;Twice, to get to the line number
5075: JSR  $42EC    ;Get the MSB of the line number
5078: STA  $79      ;Save it
507A: LDA  $17      ;Get current line number we are searching for
507C: CMP  $79      ;Same line number?
507E: BCC  $509F    ;No, less than line number we are searching for
5080: BEQ  $5085    ;Same line number, then branch
5082: DEY                ;Index pointer - 1
5083: BNE  $5093
5085: DEY                ;Decrement the index point to the LSB
5086: JSR  $42EC    ;Of the line number and get the LSB
5089: STA  $79      ;Save it
508B: LDA  $16      ;Get current line number we are searching for
508D: CMP  $79      ;See if they are the same
508F: BCC  $509F    ;If less than, then exit
5091: BEQ  $509F    ;The same, then exit leaving the line link in

```



```
                                     ;$61, $62
5093:  DEY                                 ;Greater than, so try again
5094:  JSR   $42EC                         ;Get the MSB of the link to the next line
5097:  TAX                                 ;Save it in the X register
5098:  DEY                                 ;Back up to the LSB of the link to the next line
5099:  JSR   $42EC                         ;And get it
509C:  BCS   $5068                         ;Branch back to search for the next line
509E:  CLC                                 ;Flag, for line number not found
509F:  RTS                                 ;Exit
```

```
*****
*
*                                     *
*                               LINGET *
*                                     *
*                               LINE GET *
*                                     *
* This routine converts an ASCII line number to a two *
* byte integer number in the LSB/MSB format.  On entry *
* to this routine, the accumulator holds the first *
* character of the line number, TXTPTR points to that *
* byte and the carry is clear.  Also location $0A will *
* contain a value of $00 if no valid line number is *
* found.  The range of the line number is also checked *
* by this routine, and a 'SYNTAX ERROR' will occur if *
* the line has a line number with a value that is not *
* within the limits of 0 - 63999.  This routine exits *
* with CHRGET pointing to the first character after *
* the line number and locations $16, $17 contain the *
* LSB and MSB of the line number.
*
*****
```

```
50A0:  LDX   #0
50A2:  STX   $0A      ;Clear the temporary work area
50A4:  STX   $16      ;Clear address of the line number
50A6:  STX   $17      ;In low/high byte order ($16/$17)
50A8:  BCS   $50E1    ;If the carry flag is set exit the routine
50AA:  INC   $0A      ;Set Flag to indicate the valid line # found
50AC:  SBC   #$2F     ;The carry is clear, so subtract $30 to make a
                      ;True number ( $31 - $30 = Nominal number 1 )
50AE:  STA   $09      ;Save it as the LSB
50B0:  LDA   $17      ;Save the MSB
50B2:  STA   $24      ;Temporarily
50B4:  CMP   #$19     ;Compare to the highest allowable value
50B6:  BCC   $50BB    ;If less than, then continue
50B8:  JMP   $796C    ;If >= to, then 'SYNTAX ERROR'
50BB:  LDA   $16      ;Get the LSB of the line number
```

```

50BD: ASL                ;Multiply it by two
50BE: ROL    $24        ;Preserve the carry bit
50C0: ASL                ;Multiply the LSB by two again (4)
50C1: ROL    $24        ;Preserve the carry bit
50C3: ADC    $16        ;Add to the original value
                    ;( total multiplication factor of 5 )

50C5: STA    $16
50C7: LDA    $24        ;Add the carry to the
50C9: ADC    $17        ;MSB
50CB: STA    $17
50CD: ASL    $16        ;Multiply the LSB by two again (10)
50CF: ROL    $17        ;Add the carry, if any
50D1: LDA    $16        ;Add the digit from the line number being read
50D3: ADC    $09        ;To the LSB of the final number
50D5: STA    $16        ;Save it
50D7: BCC    $50DB      ;No overflow, then branch
50D9: INC    $17        ;Increment the MSB by one
50DB: JSR    $0380      ;Get next character
50DE: JMP    $50A8      ;And convert it
50E1: RTS

*****
*
*          BASIC LIST COMMAND
*
* Command syntax: LIST <first line#> - <last line#>
*
* This routine will call FRMTO to get the parameters
* LIST FROM and TO.
*
*****

50E2: JSR    $5EFB      ;Check for LIST XXX-XXX etc.
50E5: LDY    #$01       ;Place index pointer on MSB of the line link
50E7: JSR    $42EC      ;Get the MSB of the line link
50EA: BNE    $50F2      ;If it is not zero, then branch
50EC: DEY
                    ;If the MSB is equal to zero, then check the LSB
50ED: JSR    $42EC      ;Get the LSB of the line link
50F0: BEQ    $5120      ;If the LSB is equal to zero, then
                    ;Print a <cr> or linefeed and exit the routine
                    ;(end of program)

50F2: JSR    $4BB5      ;Check the STOP key and abort if key is pressed
50F5: JSR    $5598      ;Print a carriage return
50F8: LDY    #$02       ;Index to the line number
50FA: JSR    $42EC      ;Get the LSB of the line number
50FD: TAX
                    ;Save it in the X register
50FE: INY
                    ;Move up to the MSB

```

```

50FF: JSR   $42EC      ;Get the MSB of the line number
5102: CMP   $17        ;Is it the last line number to be listed
5104: BNE   $510A      ;If not, then continue
5106: CPX   $16        ;If the MSB matches, then check the LSB
5108: BEQ   $510C      ;If it is equal, then continue
510A: BCS   $5120      ;If it is greater than, then exit
510C: JSR   $5123      ;LIST the remainder of the line
510F: LDY   #0         ;Index to the line link
5111: JSR   $42EC      ;Get the LSB of the address to the next line
5114: TAX                      ;Save it in the X register
5115: INY                      ;Move to the MSB
5116: JSR   $42EC      ;Get the MSB of the address to the next line
5119: STX   $61        ;Save the two values
511B: STA   $62        ;To LIST the next line
511D: JMP   $50E5      ;Continue to loop
5120: JMP   $5598      ;Print a <cr> and a linefeed
5123: LDY   #3         ;Flag for LIST
5125: STY   $4B        ;Save in LIST pointer (next char. to be read)
5127: STY   $11        ;GARBFL
5129: JSR   $8E32      ;Output line number
512C: LDA   #$20        ;Print a space to the screen
512E: LDY   $4B        ;LSTPNT
5130: AND   #$7F        ;Make char. lower case (not a shifted character)
5132: JSR   $560C      ;Print the character
5135: CMP   #'"'        ;Is it a quote?
5137: BNE   $513F      ;If not, then branch
5139: LDA   $11        ;Get flag for LIST
513B: EOR   #$FF        ;Remove flag
513D: STA   $11
513F: INY                      ;Index to the next byte
5140: BEQ   $5120      ;If equal to zero, then exit
5142: BIT   $55        ;Test flag for LIST ($00)
5144: BPL   $5149      ;If LIST, then skip
5146: JSR   $59AC      ;If HELP, then print reverse
                        ;(40 column) or underline (80 column)
5149: JSR   $42EC      ;Get the next character
514C: BEQ   $518B      ;If it is equal to zero, then exit
    
```

```

*****
*
*                               QPLOP
*
*       Transform the tokens to BASIC text (LIST)
*
* This routine changes the BASIC tokens back to BASIC
* text and equivalent commands. It is vectored through
*
    
```

```

* $0306 (IQPLOP). This makes it possible to wedge into *
* the routine to list custom BASIC commands.          *
*                                                       *
*****

```

```

514E: JMP    ($0306)    ;Normal entry point $5151
5151: BPL    $5132      ;Not a token, then ouput it (PRINT)
5153: CMP    #$FF      ;Is it π?
5155: BEQ    $5132      ;Yes, then output it
5157: BIT    $11        ;Quote mode?
5159: BMI    $5132      ;Yes, then output the text character
515B: CMP    #$FE      ;Token for added BASIC COMMAND
515D: BEQ    $518C      ;If so, then branch
515F: CMP    #$CE      ;Token for added BASIC FUNCTION
5161: BEQ    $51A6      ;If so, then branch
5163: TAX                    ;Place the TOKEN in the accumulator
5164: STY    $4B        ;Renumber the character pointer
5166: LDA    #$44      ;Set up scan for SINGLE BASIC TOKEN COMMANDS
5168: LDY    #$17      ;Starting at $4417 and place it in INDEX1
516A: STA    $25        ;Save the table in INDEX1
516C: STY    $24
516E: LDY    #0        ;Index the pointer
5170: DEX                    ;Decrement the token value by one
5171: BPL    $5182      ;Greater than 128?
5173: LDA    ($24),Y    ;Get a character from the table
5175: PHA                    ;Temporarily save it onto the stack
5176: INC    $24        ;Increment the table by one
5178: BNE    $517C      ;Read in 256 bytes yet, if not, then continue
                    ;(branch)

517A: INC    $25
517C: PLA                    ;Get the character from the table back
517D: BPL    $5173      ;End of command word yet or end of table flag
517F: BMI    $5170      ;Try next token value
5181: INY                    ;Increment the text pointer by one
5182: LDA    ($24),Y    ;Output the command word from the list
5184: BMI    $512E      ;One character at a time and output it
5186: JSR    $560C      ;Continue until the shifted character is found
5189: BNE    $5181      ;Continue until bit 7 is set (shifted) or until
518B: RTS                    ;The flag $00, which indicates an end to table

```

```

*****
*
*                               LSTABC
*
*                               LiST Added Basic Commands
*
*

```

```

* This routine is called by the aforementioned routine *
* to list the dual token BASIC commands starting with *
* $FE. On entry to the routine, the accumulator holds *
* the value $FE and the Y register contains the      *
* index to the second token minus 1.                *
*                                                    *
*****

```

```

518C: TAX          ;Place token ($FE) into the X register
518D: INY          ;Increment the index to the next byte
518E: JSR $42EC    ;Get the byte
5191: BEQ $5132    ;If it is zero, then output it
5193: STY $4B      ;If it is not, then save the index to the last
                ;Character that was read
5195: CMP #2       ;BANK command?
5197: BCC $51C0    ;If less than, then branch
5199: CMP #$27     ;Reached the end of the ADDED BASIC COMMANDS?
519B: BCS $51C0    ;If greater than, then branch
519D: ADC #$7E     ;Add 128 to create the pointer to the keyword
                ;And set bit 7 as a flag for a token
519F: TAX          ;Save it in the X register
51A0: LDY #$09     ;Address of the ADDED BASIC COMMANDS ($4609)
51A2: LDA #$46
51A4: BNE $516A    ;Branch back to print the keyword

```

```

*****
*                                                    *
*                               LSTABF                *
*                                                    *
*                               LiST Added Basic Functions
*                                                    *
* This routine is called by the aforementioned routine *
* to list the dual token BASIC commands starting with *
* $CE. On entry to the routine, the accumulator holds *
* the value $CE and the Y register contains the      *
* index to the second token minus 1.                *
*                                                    *
*****

```

```

51A6: TAX          ;Place the token ($CE) in the X register
51A7: INY          ;Index to the next byte
51A8: JSR $42EC    ;Get the second token
51AB: BEQ $5132    ;If it is equal to zero, then output it
51AD: STY $4B      ;If it is not, then save the index
51AF: CMP #2       ;POT function?
51B1: BCC $51C0    ;If less than, then print the value in the
                ;Accumulator instead of the keyword

```

```

51B3: CMP    #$0B      ;Reached the end of the ADDED BASIC FUNCTIONS?
51B5: BCS    $51C0     ;If greater than, then print
                    ;the value in the accumulator
51B7: ADC    #$7E      ;Add 126 to create the pointer to the keyword
51B9: TAX                     ;Save it to the X register
51BA: LDY    #$C9      ;Address of the ADDED BASIC FUNCTIONS ($46C9)
51BC: LDA    #$46
51BE: BNE    $516A     ;Branch back to print the keyword
51C0: CPX    #$FE      ;Was the first token an $FE?
51C2: BNE    $51C7     ;If not, then branch
51C4: LDX    #0
51C6: .BYTE $2C        ;Mask to fall through to $51C9
51C7: LDX    #$FF
51C9: SEC                     ;Flag to escape printing keyword

```

```

*****
*
*
*          ESCLST
*
*          ESCape LiST
*
* On entry to this routine, if the carry is set,
* the value in the accumulator is printed. If the
* carry is clear, the value in the accumulator is
* treated as a token, and the appropriate BASIC
* is printed. To list a string, set $24, $25
* to the address of the string which has the
* last letter shifted, clear the carry, and call
* this routine.
*
*
*****

```

```

51CA: JMP    ($030E)   ;Normally goes to $51CD
51CD: BCS    $51D3
51CF: LDY    #0        ;Clear the index
51D1: BEQ    $5182     ;Output the keyword that is pointed to by $24,
                    ;$25
51D3: JMP    $5132     ;Print the character that is in the accumulator

```

```

*****
*
*          BASIC NEW COMMAND
*
* Command syntax:  NEW
*
* This routine places two zeros starting at the
* address pointed to by TXTTAB which normally points
*
*****

```

```

* to $1C01. By putting a zero in these two bytes which *
* normally hold the line link address of the first pro- *
* gram line, an end of program is indicated to the *
* BASIC interpreter. After this has been completed, the *
* routine resets TEXT TOP, located at locations $1210 *
* and $1211 to point to the address stored at TXTTAB *
* plus 2, normally $1C03. This routine then proceeds to *
* execute a CLR command. *
* *
* NOTE: Here are a couple of quick ways to unNEW a *
* BASIC program in the 128: *
* *
* Enter: POKE DEC("1C01"),1:RENUMBER *
* *
* Or how about this one: *
* *
* Enter: POKE DEC("1C01"),1:DELETE 2-1 *
* *
* AND ONE LAST ONE *
* *
* Enter: POKE DEC("1C01"),1:SYS DEC("4F4F"); *
* SYS DEC("4F82") *
* *
* NOTE: There are numerous ways of unNEWing your *
* program, but, be very careful. Many so-called *
* unNEW programs do not save the TEXT TOP point- *
* ers, thereby stopping you from saving or *
* editing the program just unNEWed. *
* *
*****

```

```

51D6: BEQ    $51D9    ;If the byte after the NEW command is zero,
                    ;or a colon ";", then continue
51D8: RTS
51D9: LDA    #0      ;Clear the accumulator
51DB: TAY
51DC: STA    ($2D),Y ;Reset the start
51DE: INY
51DF: STA    ($2D),Y ;Back to TXTTAB
51E1: STA    $116F   ;Turn TRACE MODE off
51E4: LDA    $2D     ;Reset the END of the
51E6: CLC
51E7: ADC    #$02    ;TXTTAB + 2
51E9: STA    $1210   ;LSB
51EC: LDA    $2E
51EE: ADC    #0      ;Add the carry, if any, and store as
51F0: STA    $1211   ;The MSB

```

```

51F3: JSR   $5254      ;Set the program pointer back to the
                    ;Start of the program
51F6: LDA   #0         ;Load the accumulator with $00 to execute a CLR
                    ;Command

*****
*
*           BASIC CLR COMMAND
*
* Command syntax: CLR
*
* This routine will close all the open I/O channels
* which will reset the I/O to the keyboard and the
* screen. The routine then resets the string and
* pseudo stack pointers to their default values and
* also resets various other pointers for commands such
* as TRAP. The routine will also call the RESTORE
* routine to reset the data pointer.
*
*****

51F8: BNE   $524F      ;Useless code unused by BASIC
51FA: JSR   $927B      ;Close all open I/O channels, return system I/O
                    ;to the keyboard/screen
51FD: LDY   #0         ;Set the error flag to force a
51FF: STY   $7A        ;new DS$ to be obtained
5201: DEY
5202: STY   $120C      ;Set line number to be executed on error, and
5205: STY   $1209      ;the line number of the last error to $FFFF
5208: STY   $120A
520B: STY   $1208      ;Set last error number for TRAP command to $FF
520E: LDA   $39        ;Get the pointer to the end of string memory
5210: LDY   $3A
5212: STA   $35        ;and save it as the start of string memory
5214: STY   $36
5216: LDA   #$FF       ;Clear the pseudo stack
5218: LDY   #$09       ;of all entries by setting
521A: STA   $7D        ;the Top of Stack pointer (TOS) to $09FF
521C: STY   $7E
521E: LDA   $2F        ;Get LSB of the start of BASIC variables address
5220: LDY   $30        ;Get MSB of the start of BASIC variables address
5222: STA   $31        ;And save it as the start and end of
5224: STY   $32        ;Of BASIC arrays
5226: STA   $33
5228: STY   $34
522A: LDX   #$03
522C: LDA   $5250,X    ;Get a byte from the PRINT USING table

```



```

522F: STA $1204,X ;and place it in the print using pointers
5232: DEX ;Do four characters
5233: BPL $522C ;Space, comma, decimal point, and dollar sign
5235: JSR $5AE1 ;Perform a BASIC RESTORE command
5238: LDX #$1B ;Clear the temp. string stack
523A: STX $18 ;of all entries
523C: PLA ;Get the return address off the stack
523D: TAY ;of the routine that called this routine
523E: PLA ;and save it in the Y-register/accumulator then
523F: LDX #$FA ;Reset the stack pointer to clear the stack of
5241: TXS ;all addresses and information
5242: PHA ;Return the address
5243: TYA ;of the routine that called this routine back
5244: PHA ;to the stack
5245: LDA #0
5247: STA $1203 ;Erase the line number for CONT command
524A: STA $12 ;Clear the FOR/NEXT pointer
524C: STA $03DF ;Clear the overflow pointer for FAC1
524F: RTS ;Exit

```

```

*****
*
*          DEFAULT VALUES FOR PRINT USING COMMAND          *
*
*****

```

```

5250: TXT ' ,.,$' ;Space, comma, decimal point, dollar sign

```

```

*****
*
*          RUNC          *
*
* This routine sets the TXTPTR to point to the start *
* of BASIC text. This is done so that the next *
* character read in by CHRGET/CHRGOT will be the first *
* character in the BASIC text. (TXTPTR = TXTTAB-1) *
*
*****

```

```

5254: CLC
5255: LDA $2D ;Get the LSB of the Basic start (TXTTAB)
5257: ADC #$FF ;Subtract one from the LSB of TXTTAB
5259: STA $3D ;Save as LSB of TXTPTR
525B: LDA $2E ;Get MSB of BASIC start (TXTTAB)
525D: ADC #$FF ;Subtract one if the carry is clear
525F: STA $3E ;Save as the MSB of TXTPTR
5261: RTS ;Exit

```

```

*****
*
*          BASIC RETURN COMMAND          *
*
* Command syntax:  RETURN                *
*
* This routine will pull the address off the stack of *
* the routine that called this routine, finds the GOSUB*
* address on the pseudo stack and uses these values to *
* set the current line and current address (TXTPTR).  *
*
*****

```

```

5262: PLA          ;Pull the return address of the calling routine
5263: PLA          ;off the stack
5264: LDA  #8D      ;Token for GOSUB
5266: JSR  $4FAA    ;Search for the last GOSUB on the stack
5269: BEQ  $5270    ;If found, branch to pull the return parameters
526B: LDX  #12      ;'RETURN WITHOUT GOSUB' error
526D: JMP  $4D3C    ;Error routine

```

```

-----*
*
*          Get the RETURN parameters from the GOSUB *
*          entry on the pseudo stack                *
*
*   BYTE      DESCRIPTION          LOW/HIGH        *
*   ----      -
*
*   4          RETURN ADDRESS      HIGH           *
*   3          RETURN ADDRESS      LOW             *
*   2          LINE NUMBER         HIGH           *
*   1          LINE NUMBER         LOW            *
*   0          $8D - (TOKEN FOR GOSUB)           *
*
*
*-----*

```

```

5270: JSR  $5050    ;Save the end of stack pointer in $3F/$40
                    ;LOW/HIGH
5273: LDY  #5       ;Index pointer to number of bytes on the stack
5275: JSR  $5059    ;Add the Y register to the end of stack pointer
5278: DEY          ;Subtract one to begin with return address
5279: LDA  ($3F),Y  ;Get the MSB of the return address
527B: STA  $3E      ;Save it as the pointer to BASIC TEXT MSB
527D: DEY          ;Subtract one from the index pointer
527E: LDA  ($3F),Y  ;Get the LSB of the return address
5280: STA  $3D      ;Save it as the pointer to BASIC TEXT LSB

```

```

5282: DEY                ;Subtract one to point to current line number
5283: LDA  ($3F),Y      ;Get the current line number MSB and add one to
5285: JSR  $A83B        ;it and save it as current line # MSB Y=Y+1
                        ;If line number equals $FF, then set the program
                        ;mode to the direct mode, end the program
5288: LDA  ($3F),Y      ;Get the current line number and
528A: STA  $3B          ;save it as the LSB
528C: JMP  $528F        ;Search for a colon or $00 ignoring everything
                        ;between GOSUB and the value

```

```

*****
*
*          BASIC DATA STATEMENT
*
* Statement syntax: DATA item1, item2, etc.
* Example: DATA 1,2,3, HELLO, BYE: SCNCLR
*
* This routine searches for the next occurrence of a
* colon or the end of line flag ($00) skipping
* everything in between. It works much like the REM
* command, except you can follow the last DATA item
* with a colon and a valid statement (as in example).
* The main use of the DATA statement is to hold
* data for the READ command.
*
*****

```

```

528F: JSR  $52A2        ;Search for next statement or chain flag ':'

```

```

*-----*
*
* The Y register contains the offset in bytes to the
* next statement. This routine points TXTPTR to the
* next statement in the line, by adding the value
* in the Y-register to TXTPTR
*
*-----*

```

```

5292: TYA                ;Save pointer
5293: CLC                ;Clear the carry flag for addition
5294: ADC  $3D          ;Add it to the program pointer
5296: STA  $3D
5298: BCC  $529C        ;No overflow
529A: INC  $3E          ;Overflow, so increment the MSB by one
529C: RTS                ;Exit

```

```

*****
*
*           BASIC REM COMMAND
*
* Command syntax:  REM <comments>
*
* This routine will search for the end of line flag
* ($00). Then the routine adds the offset of the
* search to TXTPTR so that TXTPTR will point to the
* LSB of the line link of the next line.
*
*****

```

```

529D: JSR  $52A5      ;Search for the statement terminator flag ($00)
                    ;End of line flag
52A0: BEQ  $5292      ;Add offset in bytes to the next line to TXTPTR

```

```

*****
*
*   FIND THE OFFSET TO THE CHARACTER SPECIFIED BELOW
*
* This routine will search for the next occurrence of
* the character and if it is found, location $09 will
* hold the number of characters it had to count over
* from TXTPTR to find or match the characters.
*
*****

```

```

52A2: LDX  #' : '      ;Search for a colon ' : '
52A4: .BYTE $2C        ;Mask to fall through to $52A7
52A5: LDX  #0          ;Search for the statement terminator

```

```

-----*
*
*   If you enter the routine at this point, the X
*   register must hold the character to be searched for.
*   If this routine should encounter the end of line
*   terminator before the character being searched for
*   is found, the routine will abort the search.
*
-----*

```

```

52A7: STX  $09          ;Save the character to be searched for
52A9: LDY  #0           ;Clear the OFFSET pointer
52AB: STY  $0A
52AD: LDA  $0A          ;Switch the locations of the INDEX pointer
52AF: LDX  $09          ;and the character to be searched for

```

```

52B1: STA  $09      ;Character to be searched for
52B3: STX  $0A      ;Offset pointer
52B5: JSR  $03C9    ;Get a byte from RAM BANK 0
52B8: BEQ  $529C    ;Is it the terminator flag? If so, then branch
52BA: CMP  $0A      ;Is it the character we are looking for?
52BC: BEQ  $529C    ;If it is, then exit
52BE: INY                ;Move the pointer to the next character
52BF: CMP  #'"'      ;Was the last character we got a quote mark?
52C1: BNE  $52B5    ;No, then test the next character
52C3: BEQ  $52AD    ;Switch them back and try again

```

```

*****
*
*
*          BASIC IF COMMAND
*
* Statement syntax: IF <expression>
*
* This routine first evaluates the expression after
* the 'IF' statement and stores the result of the
* expression ( $FF = TRUE, $00 = FALSE ) in location
* $63. It then checks for a 'THEN' or a 'GOTO' after
* the expression. If the result of the expression is
* false, the routine checks if the command following
* the 'THEN' statement is 'BEGIN'. If it is, the
* routine searches for a 'BEND' command and sets
* TXTPTR to the statement following the 'BEND' and
* executes it. If a 'BEGIN' was not found, the routine
* searches for an 'ELSE' command on the same line.
*
* NOTE: If the syntax of 'IF/THEN/BEGIN' is used, the
* 'ELSE' command must be after the first 'BEND'
* command or the 'ELSE' will not be executed.
* If there is not an 'ELSE' command after the
* 'BEND' command, everything on the same line
* as the 'BEND' is ignored until an 'ELSE' com-
* mand or the end of line flag ($00) is found.
*
*****

```

```

52C5: JSR  $77EF    ;Evaluate the expression after 'IF'
52C8: JSR  $0386    ;Get the first character after the expression
52CB: CMP  #$89     ;Is it the token for 'GOTO'?
52CD: BEQ  $52D4    ;Yes, then skip the check for 'THEN'
52CF: LDA  #$A7     ;Token for 'THEN'
52D1: JSR  $795E    ;Search for 'THEN', if not found, 'SYNTAX ERROR'
52D4: LDA  $63     ;Get the result ($FF=TRUE, $00=FALSE)

```

```

52D6: BNE    $52FE    ;If the result is TRUE, then branch to execute
                    ;The statement after the 'THEN' or 'GOTO'
52D8: JSR    $0386    ;Get the character after the 'THEN' or 'GOTO'
52DB: CMP    #$FE     ;Is it the first part of a dual token command?
52DD: BNE    $52EA    ;No, then skip to find a colon or $00
52DF: INY                    ;If it is, then check the second byte
52E0: JSR    $03C9    ;Get the byte after the $FE
52E3: CMP    #$18     ;Token for 'BEGIN'?
52E5: BNE    $52EA    ;If not, then find a colon or $00
52E7: JSR    $5320    ;Search for a 'BEND' and set TXTPTR to it
52EA: JSR    $528F    ;Search for a colon or end of line
52ED: LDY    #0       ;Index to the next byte
52EF: JSR    $03C9    ;Check the next character
52F2: BEQ    $529D    ;If equal to zero, then execute a REM COMMAND
52F4: JSR    $0380    ;If not, then get the same character
52F7: CMP    #$D5     ;Compare it to the 'ELSE' token
52F9: BNE    $52EA    ;If not equal, then search until an 'ELSE' or
                    ;An end of line terminator is found
52FB: JSR    $0380    ;If an 'ELSE' is found, then get the first
                    ;Character after the 'ELSE'

```

```

*-----*
*
* This routine evaluates the character or token that
* is currently pointed to by TXTPTR and executes it.
*
*-----*

```

```

52FE: JSR    $0386    ;Get the last character
5301: BEQ    $531A    ;If it is equal to zero or a colon, then exit
5303: BCS    $5308    ;If it is a character, then check for 'BEGIN'
5305: JMP    $59DB    ;If it is a digit, then execute a 'GOTO'
5308: CMP    #$FE     ;Compare the first character to a dual token
530A: BNE    $531A    ;If it is not, then exit
530C: INY                    ;If it is, increment index to the second token
530D: JSR    $03C9    ;Get the second token
5310: CMP    #$18     ;Token for 'BEGIN'?
5312: BNE    $531A    ;No, then exit
5314: JSR    $0380    ;Update the TXTPTR
5317: JSR    $0380    ;Get the character
531A: JSR    $0386    ;Get the char that TXTPTR's currently
531D: JMP    $4B3F    ;pointng to and execute it

```

```

*-----*
*
* This routine searches for 'BEND' and updates TXTPTR *
* to point to the token for 'BEND'. *
* *
*-----*

```

```

5320: JSR   $0380      ;Get the next character
5323: BNE   $534C      ;If not a ':' or statement terminator, branch
5325: CMP   #' :'      ;Is it the chain flag ':' ?
5327: BEQ   $5320      ;If it is, then branch get the next character
5329: BIT   $7F        ;Check to see if were in DIRECT mode
532B: BPL   $5377      ;If we are, then branch to 'BEND NOT FOUND'
532D: LDY   #2         ;Index pointer
532F: JSR   $03C9      ;Get the next character (TXTPTR)
5332: BEQ   $5377      ;End of the BASIC LINE? If so, then branch
5334: INY                   ;Increment the INDEX pointer
5335: JSR   $03C9      ;Get the next character (TXTPTR+INDEX)
5338: STA   $3B        ;Save it as the current BASIC line number LSB
533A: INY                   ;Increment the INDEX pointer
533B: JSR   $03C9      ;Get the line number MSB
533E: STA   $3C        ;Save it as the current BASIC line number MSB
5340: TYA                   ;Place INDEX pointer in the accumulator
5341: CLC                   ;Clear the carry for addition
5342: ADC   $3D        ;Add the TXTPTR LSB to the INDEX pointer
5344: STA   $3D        ;Move the BASIC TXTPTR to where we left off
5346: BCC   $5320      ;No carry, then branch
5348: INC   $3E        ;Add one to the BASIC TXTPTR MSB
534A: BNE   $5320      ;Get the next character
534C: CMP   #' "'      ;Is it a quote?
534E: BNE   $5357      ;No, then branch
5350: JSR   $537C      ;Search for the ending quote or $00
5353: BEQ   $5325      ;Check if a quote or $00
5355: BNE   $5320      ;and continue with the loop
5357: CMP   #$8F      ;Token for REM?
5359: BNE   $5361      ;No, then branch
535B: JSR   $529D      ;Then execute a REM COMMAND
535E: JMP   $5329      ;Update TXTPTR
5361: CMP   #$FE      ;Dual token BASIC COMMAND?
5363: BNE   $5320      ;No, then keep searching
5365: JSR   $0380      ;Yes, then get the second token
5368: CMP   #$19      ;Token for BEND?
536A: BEQ   $5376      ;Yes, then branch to exit routine
536C: CMP   #$18      ;Token for BEGIN?
536E: BNE   $5320      ;No, then keep searching
5370: JSR   $5320      ;Yes, then
5373: JMP   $5320      ;Keep searching for 'BEND'

```

```
5376: RTS
5377: LDX #37 ;'BEND NOT FOUND' error
5379: JMP $4D3C
```

```
*-----*
*
* This routine is used to search a line for a quote *
* mark or an end of line terminator. If a quote mark *
* is found, get the next character after it and exit *
* the routine. If it's the end of line flag ($00) *
* then the routine exits. *
* *
*-----*
```

```
537C: LDY #0
537E: INC $3D ;Increment TXTPTR LSB by one
5380: BNE $5384 ;If not zero, then skip ahead
5382: INC $3E ;Increment MSB by one
5384: JSR $03C9 ;Get a char. from the buffer or the BASIC text
5387: BEQ $5390 ;If end of line flag or statement chain flag,
;Branch
5389: CMP #'"' ;Is it a quote ?
538B: BNE $537E ;If not, then continue searching
538D: JMP $0380 ;If it is, then get the next character and exit
5390: RTS ;Exit
```

```
*****
*
* BASIC ELSE BEGIN COMMAND *
*
* This routine is executed only if the expression in *
* an 'IF' statement is true and there is an 'ELSE' *
* command after the 'IF' command on the same line. *
*
* Basically, what this routine does is it first checks *
* to see if the command after the 'ELSE' command is *
* a 'BEGIN' command and if it is the routine searches *
* for a 'BEND' command, does a 'REM' command and exits *
* the routine. If the command after the 'ELSE' *
* command is not a 'BEGIN' command, then the routine *
* does a 'REM' command only and exits. *
*
*-----*
```



```

5391: CMP   #$FE           ;Dual TOKEN BASIC COMMAND?
5393: BNE   $53A0          ;No, then skip to the next BASIC line
5395: INY                   ;Move index pointer to next char. in BASIC text
5396: JSR   $03C9          ;Get the next character from BASIC text
5399: CMP   #$18           ;Token for 'BEGIN'?
539B: BNE   $53A0          ;No, then skip to the next BASIC line
539D: JSR   $5320          ;Search for a BEND command
53A0: JMP   $529D          ;JUMP to REM command to skip the remainder of
                        ;this line down to the next BASIC line

```

```

*****
*
*           BASIC ON COMMAND
*
* Command syntax: ON exp GOTO line #1 [,line #2,...]
*                 ON exp GOSUB line #1 [,line #2,...]
*
* The ON command will evaluate the expression
* following the ON command and then get the next
* value after the expression to see if it is the token
* for GOTO or GOSUB. If neither of these tokens are
* found, then a SYNTAX ERROR will result. If the
* token for GOTO or GOSUB is found, then the
* expression is used as an index to count over to get
* the correct line number to GOTO or GOSUB to.
*
*****

```

```

53A3: JSR   $87F4          ;Calculate the expression which will be used as
                        ;an index value.
53A6: PHA                   ;Save the token after the expression
53A7: CMP   #$8D           ;Token for GOSUB command?
53A9: BEQ   $53B2          ;Yes, then branch
53AB: CMP   #$89           ;Token for GOTO command?
53AD: BEQ   $53B2          ;Yes, then branch
53AF: JMP   $796C          ;Not GOSUB/GOTO, then 'SYNTAX' error
53B2: DEC   $67            ;Decrement the index value
53B4: BNE   $53BA          ;Not zero, then index over to the line number
53B6: PLA                   ;Get GOSUB/GOTO token back
53B7: JMP   $4B59          ;Then GOSUB/GOTO
53BA: JSR   $0380          ;Get the next character
53BD: JSR   $50A0          ;Get the line number to GOSUB/GOTO
53C0: CMP   #','          ;Equal to a comma?
53C2: BEQ   $53B2          ;Yes, then branch
53C4: PLA                   ;Get the token back
53C5: RTS                   ;Exit

```

```

*****
*
*          BASIC LET COMMAND
*
* Command syntax: [LET] variable = expression
*
* This routine first looks for the variable that is
* specified after the LET command. After this has
* been done, the routine checks to see if the value of
* the character following the variable is the equal
* sign. If it is not, then a 'SYNTAX ERROR' results.
* If the character is an equal sign, the routine evalu-
* tates the expression after the '=' and assigns the
* result to the variable specified after the LET.
*
*****

```

```

53C6: JSR   $7AAF      ;Looks for variable, and if not found
                    ;prepares new descriptor for this variable name
53C9: STA   $4B        ;Save the address
53CB: STY   $4C        ;of the variable's descriptor in RAM BANK 1
53CD: LDA   #'='       ;Checks for an equals sign after the variable
53CF: JSR   $795E      ;If not found, then 'SYNTAX ERROR'
53D2: LDA   $10        ;Save the integer flag
53D4: PHA                ;Onto the stack
53D5: LDA   $0F        ;And save the VALTYP flag
53D7: PHA                ;Onto the stack
53D8: JSR   $77EF      ;Evaluate the expression after the equals sign
53DB: PLA                ;Get the VALTYP flag back
53DC: ROL                ;If string, set the carry else clear the carry
53DD: JSR   $77DE      ;Check on correct type
53E0: BNE   $5404      ;Assign a string to the variable
53E2: PLA                ;Get the integer flag back
53E3: BPL   $53FA      ;If zero, then assign a floating point number
                    ;to the variable: If not zero assign an integer

```

```

*****
*
*          ASSIGN AN INTEGER VALUE TO A VARIABLE
*
* This routine rounds off the floating point number
* that's in FAC1 and then converts the value to integer*
* format and places the value into the variable des-
* criptor. For more information see section 1.3. On
*
*****

```

```

* entry locations $4B, $4C point to the descriptor's *
* address starting at $0400 in RAM BANK 1. *
* *
*****

```

```

53E5: JSR $8C47 ;Round off the floating point accumulator (FAC1)
53E8: JSR $84B4 ;Change FAC1 to integer format
53EB: LDY #0 ;Zero the index pointer
53ED: LDA $66 ;Get the MSB value of the integer number
53EF: STA $FF04 ;Switch, to put the value in BANK 1
53F2: STA ($4B),Y ;Place MSB of the integer value into descriptor
53F4: INY ;Increment the index pointer
53F5: LDA $67 ;Get the LSB value of the integer number
53F7: STA ($4B),Y ;Place LSB of the integer value into descriptor
53F9: RTS ;Exit the routine

```

```

*****
*
* ASSIGN A FLOATING POINT NUMBER TO A VARIABLE *
*
* This routine assigns a floating point number to the *
* variable whose address is stored in locations $4B, *
* $4C. *
*
* On entry to this routine $4B,$4C must point to the *
* address of where you want the floating point number *
* (FAC1) to be moved TO. *
*
*****

```

```

53FA: LDX $4B ;Get the LSB and
53FC: LDY $4C ;The MSB of the descriptor's address
53FE: STA $FF04 ;Switch, to put the value in BANK 1
5401: JMP $8C00 ;Transfer the Floating point ACcumulator (FAC1)
;To the variable address (descriptors)

```

```

*****
*
* ASSIGN A STRING VARIABLE *
*
* This routine assigns a string to the variable whose *
* address is stored in locations $4B,$4C. *
*
*****

```

```

5404: PLA ;Remove the integer flag from the stack
5405: LDY $4C ;Check the MSB of the string to be assigned

```

```

5407: CPY    #$03      ;to see if it's TI$ and
5409: BNE    $547D     ;If it is not TI$ then branch to see if it's DS$

```

```

*-----*
*
*          ASSIGN A STRING VARIABLE TO TI$
*
*-----*

```

```

540B: JSR    $8781     ;Delete the string from the temp. string stack
540E: CMP    #$06      ;If the length of the string is not 6 characters
5410: BNE    $5450     ;Branch to generate an 'ILLEGAL QUANTITY' error
5412: LDY    #0        ;Zero the index pointer
5414: STY    $63      ;Clear the string descriptor LSB temp. pointer
5416: STY    $68      ;Clear the string address MSB temp. pointer
5418: STY    $72      ;Reset the current character being processed
541A: JSR    $5448     ;Get an ASCII char. from the string and ensure
                    ;It's a digit 0 - 9 then add it to FAC1
541D: JSR    $8B17     ;Multiply FAC1 by 10
5420: INC    $72      ;Add one to the current char. being processed
                    ;Counter
5422: LDY    $72      ;Get the current char. being processed counter
5424: JSR    $5448     ;Get an ASCII char. from the string and ensure
                    ;It's a digit 0 - 9 then add it to FAC1
5427: JSR    $8C38     ;Round off FAC1 and move it into FAC2 (ARG)
542A: TAX                    ;Save the exponent in the X-register
542B: BEQ    $5432     ;If the exponent equals zero then branch
542D: INX                    ;Add one to the exponent
542E: TXA                    ;Move the exponent into the Acc.
542F: JSR    $8B22     ;FAC1 = FAC1 + FAC2 * 10
5432: LDY    $72      ;Get the current char. being processed counter
5434: INY                    ;Add one to move the index over to next char.
5435: CPY    #6        ;All six characters done yet?
5437: BNE    $5418     ;No, then branch to continue processing them
5439: JSR    $8B17     ;FAC1 = FAC1 * 10
543C: JSR    $8CC7     ;Convert FAC1 into a four byte signed integer
543F: LDX    $66      ;Get MID value for TI$
5441: LDY    $65      ;Get HI  value for TI$
5443: LDA    $67      ;Get LOW value for TI$
5445: JMP    $FFDB     ;Call SETTIM to set the jiffy clock

```

```

*-----*
*
* Check to ensure that the next character that is read
* in is an ASCII digit zero thru nine and if it is then
* convert it to an integer value and add it to FAC1
* If the character is not an ASCII digit then it will
*
*-----*

```

```

* generate an 'ILLEGAL QUANTITY ERROR' message.      *
*                                                       *
*-----*

```

```

5448: JSR   $03B7      ;Get a character from the string that is pointed
                    ;To by $24,25 in RAM BANK 1
544B: JSR   $0390      ;Use QNUM to query the char. and clear carry
                    ;Flag if it's an ASCII digit zero thru nine.
544E: BCC   $5453      ;Branch past the error message if it's a digit
5450: JMP   $7D28      ;Generate an 'ILLEGAL QUANTITY ERROR' message
5453: SBC   #$2F        ;Subtract $30 to convert from ASCII to NUMERIC
5455: JMP   $8DB0      ;Add the numeric value in the Acc. to FAC1

```

```

*-----*
*                                                       *
* ASSIGN A STRING TO A VARIABLE THAT'S NOT TI$ OR DS$ *
*                                                       *
*-----*

```

```

5458: PLA           ;Pull the MSB of string's address off the stack
5459: INY           ;Move the index pointer 1 byte past LSB value
545A: CMP   $36      ;Check to see if string's address is below the
545C: BCC   $5476      ;Allocated area in RAM and if so branch
545E: BNE   $5468      ;If MSBs are not equal, branch past LSB test
5460: DEY           ;Move the index pointer back over to the LSB
5461: JSR   $42E7      ;Get the LSB of the string's address
5464: CMP   $35      ;Less then FRETOP? If so then
5466: BCC   $5476      ;Branch past the test for the start of BASIC
                    ;variables
5468: LDY   $67        ;Check to ensure we are not intruding at the top
546A: CPY   $30        ;Of the variable storage area and if not branch
546C: BCC   $5476      ;To assign the move of the descriptor into the
                    ;Variable storage area from the temp. descriptor
546E: BNE   $5494      ;String storage area in RAM BANK 1
5470: LDA   $66        ;Check to see if the LSBs match and
5472: CMP   $2F        ;If they do not then branch
5474: BCS   $5494      ;
5476: LDA   $66        ;Get the address of the string
5478: LDY   $67        ;that is in memory then move the address of the
547A: JMP   $54B2      ;variable,descriptor behind the string

```

```

*-----*
*                                                       *
* This routine is used to see if the string to be    *
* assigned it for DS$ and if not it will call the    *
* routine which assign a normal string, if it is DS$ *
* then it will check location $7A to see if a serial *

```

```

* bus operation has occurred since the last time DS$ *
* was assigned, and if not exits the routine but if $7A *
* indicates that a serial bus operation has occurred *
* then a new descriptor is formed in the temp. string *
* descriptors. If no access has occurred to the serial *
* bus since the last time DS$ was assigned the $7A will *
* contain a #$40, if access has occurred the location *
* $7A will contain a #$00. *
* *
* NOTE: This routine does NOT actually read in the disk *
* drive error channel; that is handled by the *
* routines in ISVAR ($7978). *
* *
*-----*

```

```

547D: LDY   #$02           ;Move the index pointer over to the MSB of the
547F: JSR   $42E7         ;string's address and get that value in the Acc.
5482: CMP   $7C           ;Check to see if MSB of string to be assigned
5484: BNE   $545A         ;has the same MSB value as DS$ and if not branch
5486: PHA                   ;Save the MSB value onto the stack
5487: DEY                   ;Move the index over to the LSB value
5488: JSR   $42E7         ;Get the value and check to see
548B: CMP   $7B           ;If string's LSB is the same as DS$'s and if it
548D: BNE   $5458         ;does not then branch to passing it to variable
548F: LDA   $7A           ;Check to see if DS$ needs to be updated, if so
5491: BEQ   $5458         ;branch to assign the new string to DS$
5493: PLA                   ;Get the MSB value of the string's address back

```

```

*-----*
* This routine is used to take a string from the *
* descriptor and create room for it in the string *
* area in RAM BANK 1, if there is not enough room for *
* the string in memory an 'OUT OF MEMORY ERROR ' will *
* result. If enough room exists then the string is moved*
* to that location and the zero page pointers are *
* updated. *
*-----*

```

```

5494: LDY   #0             ;Move the index pointer back over to the length
5496: JSR   $42E7         ;of the string and get that value in the Acc.
5499: JSR   $8688         ;Allocate room for the string in the string
                        ;storage area
549C: LDA   $52           ;Move the string descriptor from $52,$53
549E: LDY   $53
54A0: STA   $70           ;Into $70,$71 for the next routine
54A2: STY   $71

```

```

54A4: JSR   $874E      ;Move string descriptor into the space allocated
                        ;above and then update FRESPEC.
54A7: LDA   $70        ;Get address of which temp. string descriptor
54A9: LDY   $71        ;that was being used.
54AB: JSR   $87E0      ;Delete string from the string descriptor since
                        ;the string has been moved into the string
                        ;storage area
    
```

```

*-----*
*
*   Move the string descriptor into the variable
*   descriptor in RAM BANK1
*
*-----*
    
```

```

54AE: LDA   #$63      ;Load address of which 0 page address holds the
54B0: LDY   #0        ;Address to the temp. string descriptor
54B2: STA   $52        ;Move the address of the string descriptor into
54B4: STY   $53        ;$24,$25 and $52,$53 for the various routines
54B6: STA   $24
54B8: STY   $25
54BA: JSR   $87E0      ;Delete string from the string descriptor since
                        ;The string has been moved into the variable
                        ;storage area.
54BD: JSR   $54F6      ;Ensure enough room for the variable
                        ;and that it is not the same address as DS$
54C0: BCC   $54D0      ;If the string was in RANGE the branch
54C2: LDY   #0
54C4: LDA   $4B        ;Get the LSB value of the descriptor
54C6: STA   $FF04      ;Same as BASIC BANK 15 command but wth RAM BANK1
54C9: STA   ($24),Y    ;Save it at the end of the string + 1
54CB: INY
54CC: LDA   $4C        ;Get the MSB value of the descriptor
54CE: STA   ($24),Y    ;Save it at the end of the string + 2
54D0: LDA   $4B        ;Move the variable storage pointer into
54D2: STA   $24        ;$24,$25
54D4: LDA   $4C
54D6: STA   $25
54D8: JSR   $54F6      ;Ensure enough room for the variable
                        ;and that it is not the same address as DS$
54DB: BCC   $54E9      ;If string was in range and not DS$ branch to
                        ;The string descriptor pointer after the string.
    
```

```

*-----*
*
*   This section of code places the length of the string*
*   and the flag for 'OUT OF RANGE' to inform the garbage *
    
```

```

* collection routine the length of this string and that *
* it was 'OUT OF RANGE' and now free for use. This set *
* of flags will also be set if you do something like *
* A$="C-128" then a$="C-129". The old string (C-128) *
* have an 05 FF indicating that the string's length is 5*
* and the string space in memory is now ready for use. *
*
*-----*

```

```

54DD: DEY                ;Move the index pointer back to the MSB
54DE: LDA  #FFF          ;Flag for string was out of range
54E0: STA  $FF04         ;Same as BASIC BANK 15 command but w/ RAM BANK1
54E3: STA  ($24),Y      ;Place flag for string out of range in place of
                        ;The MSB
54E5: DEY                ;Move index pointer over to the LSB position
54E6: TXA                ;Move the string LENGTH into the X register
54E7: STA  ($24),Y      ;Place LENGTH of the string into the LSB pointer
54E9: LDY  #$02         ;Place length, LSB, MSB of the string into the
54EB: LDA  #$52         ;Variable storage area
54ED: JSR  $03AB        ;Get a byte from $52,$53
54F0: STA  ($4B),Y      ;And save it in the variable storage area
54F2: DEY
54F3: BPL  $54EB        ;Continue looping until all 3 bytes are moved
54F5: RTS                ;Exit the routine

```

```

*****
*
* This routine checks to ensure there is enough room in*
* memory to place a string.If there is not enough room *
* the ACC. will contain the length of the string and the*
* carry flag will be cleared. If there is enough room in*
* memory the address of the free space will be in $24, *
* $25 and the carry flag will be set. If the address *
* should happen to be the same as for DS$ then the 1st *
* statement holds true. On entry to this routine, *
* the ACC. holds the length of the string *
* that you need room for. *
*
*****

```

```

54F6: LDY  #0            ;Zero the index pointer
54F8: JSR  $03B7         ;Get a byte from $24,$25 which point to string
                        ;Descriptor
54FB: PHA                ;Save the length of the string onto the stack
54FC: BEQ  $5537         ;If the length is zero branch to exit routine
54FE: INY                ;Move the index pointer over to the LSB of the
                        ;String's address

```



---

```

54FF: JSR   $03B7   ;Get the LSB
5502: TAX                       ;Save it in the X-register
5503: INY                       ;Move the index pointer over to the MSB of the
                               ;String's address
5504: JSR   $03B7   ;Get the MSB
5507: CMP   $3A     ;If the string's address is lower then $FFXX
5509: BCC  $5511   ;Then branch to bypass the test on the LSB
550B: BNE  $5537   ;If it was not less then $FFxx then it must be
                               ;Higher then $FFXX so set the string 'OUT OF
                               ;RANGE FLAG'
550D: CPX   $39     ;Check to see if the LSB of string's address is
550F: BCS  $5537   ;Greater than allowed and if so branch to the
                               ;String 'OUT OF RANGE FLAG'
5511: JSR   $03B7   ;Get the MSB address of the string AGAIN!
5514: CMP   $36     ;Check to see if it's
5516: BCC  $5537   ;Greater than allowed and if so branch to the
                               ;String 'OUT OF RANGE FLAG'
5518: BNE  $551E   ;Branch past test on the LSB if it's not equal
                               ;Indicating thats it's in range
551A: CPX   $35     ;Check to see if it's below the bottom of
                               ;Current string memory is
551C: BCC  $5537   ;less than allowed; if so branch to the
                               ;String 'OUT OF RANGE FLAG'
551E: CMP   $7C     ;Check to see if it's the same address as DS$
5520: BNE  $5526   ;If MSBs don't match then at this point it's
                               ;In range and not DS$ so branch to set the 'IN
                               ;RANGE'flag and-
5522: CPX   $7B     ;If MSB matched address for DS$ then check the
5524: BEQ  $5537   ;LSBs and if they're equal branch to set the
                               ;String 'OUT OF RANGE FLAG'
5526: STX   $24     ;Save the address of the string in $24,$25
5528: STA   $25
552A: PLA                       ;Pull the length of the string off the stack
552B: TAX                       ;And save it in the X-register
552C: CLC                       ;Clear the carry flag for addition
552D: ADC   $24     ;Add the length of the string to the address of
552F: STA   $24     ;The string
5531: BCC  $5535   ;If no overflow occurred branch
5533: INC   $25     ;Add one to the MSB of the string's address
5535: SEC                       ;Set the carry flag to indicate string was not
                               ;DS$ and was 'IN RANGE'
5536: RTS                       ;Exit the routine
5537: PLA                       ;Pull length of the string back off the stack
5538: CLC                       ;Clear the carry flag to indicate the string was
5539: RTS                       ;'OUT OF RANGE' then exit the routine

```

```

*****
*
*          BASIC PRINT# COMMAND
*
* Command syntax: PRINT# file number [,print list]
*
* This routine will first call the BASIC CMD routine
* at $5540 to redirect the output to the logical file
* number specified after the pound sign. The routine
* then calls the BASIC PRINT routine to print any
* text. After performing the PRINT command, the
* output file is closed and the system is reset to the
* standard I/O configuration of keyboard and screen.
*
*****

```

```

553A: JSR   $5540      ;Perform BASIC CMD and PRINT commands
553D: JMP   $5658      ;Jump to close the CMD output file number

```

```

*****
*
*          BASIC CMD COMMAND
*
* Command syntax: CMD file number [,variable]
*
* This routine will convert the ASCII value of the
* number that follows the CMD command to a numeric
* value in order to get the logical file number. If
* the keyboard was selected, then an error will
* result. If a valid output device was selected, then
* the routine checks to see if the next character
* is a comma and if it is, then this routine will
* print the variable to the output device. If there is
* no comma, then a return or a linefeed is printed
* depending on the value of the logical file number.
* If the logical file number is less than 128, then a
* carriage return is printed. However, if the logical
* file number is greater than 128, then a linefeed as
* well as a carriage return is printed.
*
*****

```

```

5540: JSR   $87F4      ;Convert ASCII to numeric and place it in
                    ;The X-register (Logical file number)
5543: BEQ   $554A      ;If no parameters after the ')' then branch to
                    ;Open the output file
5545: LDA   #' , '     ;See if there is a comma after

```

```

5547: JSR   $795E      ;The token for CMD. If so, skip it
554A: PHP                               ;Save the processor status register
554B: STX   $15        ;Save logical file number for screen prompting
554D: JSR   $90EB      ;Set output device
5550: PLP                               ;Get the processor status back
5551: JMP   $555A      ;Jump to the BASIC PRINT command
    
```

```

*-----*
*                PRTSTR                *
*                *                       *
*                PRINT STRING           *
*                *                       *
* This routine is used by the print command as the main *
* loop; it will first print the string to the current *
* output device then get the character after the *
* variable and fall through to the BASIC PRINT COMMAND. *
*                *                       *
*-----*
    
```

```

5554: JSR   $55E5      ;Print the string
5557: JSR   $0386      ;Get the last character
    
```

```

*****
*                BASIC PRINT COMMAND    *
*                *                       *
* Command syntax: PRINT [variable][' ' ';' ][variable] *
*                PRINT [" <text to be printed> "] *
*                *                       *
* This routine first checks to see if there is *
* anything after the PRINT statement to print. If *
* not, the routine will print a carriage return and or *
* a line feed. If there is something after the PRINT *
* statement, then this routine checks to see if it is *
* the value for USING, TAB(), or SPC() and then goes to *
* execute the routines that handle whichever function *
* is specified. If the value after the PRINT *
* statement is none of the above, then the routine *
* checks to see if the value is a comma. If it is a *
* comma, then this routine will print the text, *
* print over ten spaces, and restore the routine to *
* the check that is made after the PRINT USING check. *
* The routine will then test for a semicolon and if *
* it is found, the semicolon is skipped and the value *
* of the variable after the semicolon is printed. *
*                *                       *
*****
    
```

```

555A: BEQ   $5598   ;End of statement, then print return
555C: CMP   #$FB    ;Is the character after PRINT
555E: BNE   $5563   ;The token for 'USING'? If not, branch
5560: JMP   $9520   ;Go to PRINT USING routine
5563: BEQ   $55A8   ;No character, then exit routine
5565: CMP   #$A3    ;Token for 'TAB(' ?
5567: BEQ   $55B9   ;If so, then carry still set go to TAB(
5569: CMP   #$A6    ;Token for 'SPC(' ?
556B: CLC                      ;Set flag for SPC(
556C: BEQ   $55B9   ;Then go to TAB( / SPC( command routine
556E: CMP   #','    ;Check to see if there is a comma
5570: BEQ   $55A9   ;If so, branch to set the cursor position to ten
                    ;Columns over from current cursor position
5572: CMP   #';'    ;Check to see if there is a semicolon
5574: BEQ   $55D4   ;If so, then skip it
5576: JSR   $77EF   ;Evaluate the expression
5579: BIT   $0F     ;Type flag $00 = NUMERIC, $FF = STRING
557B: BMI   $5554   ;Branch if it's a string and print it.
557D: JSR   $8E42   ;Change floating point accumulator
                    ;To ASCII string (FOUT)
5580: JSR   $869A   ;Get the string parameters
5583: JSR   $55E5   ;Print the string
5586: JSR   $5600   ;Print a space
5589: BNE   $5557   ;Continue

```

```

*-----*
*
* This routine places an end of line flag ($00) in the
* input buffer at $0200 plus the value in the
* X register and will print a return if the channel
* ($15) is set to the standard I/O configuration which
* is the keyboard and screen. The Y register will hold
* a #$01 and the X register will hold a $FF which is
* the MSB and LSB of the address that points to the
* start of the input buffer minus one.
*
*-----*

```

```

558B: LDA   #$00      ;Input buffer
558D: STA   $0200,X   ;Place statement terminator ($00) in the buffer
5590: LDX   #$FF      ;Set pointer to the input buffer
5592: LDY   #$01      ;To $01FF
5594: LDA   $15       ;Number of output devices
5596: BNE   $55A8   ;If the current output device is not the screen,
                    ;Then exit the routine

```

```

*-----*
*
* Print a return to the current output device and then
* check to see if the logical file number is 128 or
* greater. If it is greater than 128 then print a
* linefeed.
*
*-----*

```

```

5598: LDA    #$0D      ; <CR>
559A: JSR    $560C     ;Print it
559D: BIT    $15       ;Check the logical file number
559F: BPL    $55A6     ;Smaller than 128?
55A1: LDA    #$0A     ;Linefeed
55A3: JSR    $560C     ;Print it

```

```

*-----*
*
* If the routine is entered at this point, the
* character in the accumulator is reversed (mirrored).
*
*-----*

```

```

55A6: EOR    #$FF     ;Restore the char. back to its original value
55A8: RTS

```

```

*-----*
*
* This routine will take the current cursor position
* and print over ten spaces from that position. This
* routine is used by the BASIC PRINT command to print
* ten spaces when variables are separated by a comma
* in a print statement.
*
*-----*

```

```

55A9: SEC                ;Set the carry flag
55AA: JSR    $928D       ;To get the current cursor position
55AD: TYA                ;Get cursor position (column)
55AE: SEC                ;Set the carry for subtraction
55AF: SBC    #$0A       ;Subtract 10 from the current cursor position
55B1: BCS    $55AF       ;If overflow, then branch
55B3: EOR    #$FF       ;Make the value positive
55B5: ADC    #$01       ;Then add one
55B7: BNE    $55CF       ;Print the spaces

```

```

*****
*
*           TAB( AND SPC( COMMANDS
*
* Command syntax: TAB( <number of columns to tab> )
*                 SPC( <number of columns to space> )
*
* If the carry is set, then execute the TAB( command.
* If the carry is clear, then execute the SPC( command.*
* This routine is called by the PRINT routine.
*
*****

55B9: PHP           ;Save the flag
55BA: SEC           ;Set the carry flag
55BB: JSR   $928D   ;Get the cursor position
55BE: STY   $0B     ;Save it
55C0: JSR   $87F1   ;Get the byte value
55C3: CMP   #' ) '  ;Close parenthesis
55C5: BNE   $55DA   ;If it's not, then 'SYNTAX ERROR'
55C7: PLP           ;Get the flag back
55C8: BCC   $55D0   ;Carry not set? Then it must be a 'SPC('
                    ;Command
55CA: TXA           ;Not a 'SPC(', then it's a 'TAB(' command
55CB: SBC   $0B     ;Place TAB value in ASCII and compare to cursor
55CD: BCC   $55D4   ;Position value less than position, then ready
55CF: TAX           ;Place ASCII value back into the X register
55D0: INX           ;Increment the pointer by one
55D1: DEX           ;Decrement the pointer by one
55D2: BNE   $55DD   ;Print a space
55D4: JSR   $0380   ;Get the next character
55D7: JMP   $5563   ;Restart the loop
55DA: JMP   $796C   ;'SYNTAX ERROR'
55DD: JSR   $5600   ;Print a space or a question mark
55E0: BNE   $55D1   ;Not done, so continue

*****
*
*           STROUT - STRing OUTPUT
*
* The accumulator and the Y register hold the LSB and
* the MSB of the address of the string to be
* printed. NOTE: The string must be located in
* RAM BANK 0 and the string must end with a zero
* or a quote mark.
*
*****

```

```

55E2: JSR   $869A      ;Search for the string, if not found, create it
55E5: JSR   $8781      ;Delete a temporary string
55E8: TAX                      ;Place the string length into the pointer
55E9: LDY   #0          ;Zero the index pointer
55EB: INX                      ;Add one to the length for the next decrement
55EC: DEX                      ;Decrement the string length of the string
55ED: BEQ   $55A8      ;Done? If so, then exit the routine
55EF: JSR   $03B7      ;Nope, then get the next character
55F2: JSR   $560C      ;Print the character
55F5: INY                      ;Increment the index pointer
55F6: CMP   #$0D       ;Is the character a carriage return?
55F8: BNE   $55EC      ;No, then keep printing
55FA: JSR   $55A6      ;Invert value
55FD: JMP   $55EC      ;Back to the loop

```

```

*****
*
* This routine checks the logical file number in the
* channel location ($15) and if it does not indicate
* the current I/O configuration as being the keyboard
* and screen, the routine prints a space. If it is
* the standard I/O configuration, then the routine
* prints a cursor right.
*
*****

```

```

5600: LDA   $15          ;If the input device is the keyboard, then
5602: BEQ   $5607      ;Print a cursor right

```

```

*-----*
*
* Enter this routine here to print a space.
*
*-----*

```

```

5604: LDA   #32          ;Print a space
5606: .BYTE $2C          ;Skip to cursor right

```

```

*-----*
*
* Enter this routine here to print a cursor right.
*
*-----*

```

```

5607: LDA   #29          ;Print a cursor right
5609: .BYTE $2C          ;Skip to $560C

```

```

*-----*
*
*   Enter this routine here to print a question mark.
*
*-----*

560A: LDA  #'?'      ;Print a question mark

*-----*
*
*   Print the character in the accumulator and test the
*   STOP key.  If the STOP key is depressed, then exit
*   routine with a 'BREAK' message.
*
*-----*

560C: JSR  $90DF      ;Output character and check for the STOP key
                        ;If the key was depressed, 'BREAK'
560F: AND  #$FF       ;Place the value that was originally printed
                        ;in the ACC.
5611: RTS             ;Exit routine

*****
*
*           BASIC GET COMMAND
*
*   Command syntax:  GET variable list
*
*   This routine first checks to see if you are in DIRECT*
*   mode or PROGRAM mode.  If you are in DIRECT mode, the *
*   'ILLEGAL DIRECT' results.  If you are in PROGRAM mode,*
*   the routine checks the first character following the *
*   GET command and stores the character value in loca- *
*   tion $77.  If the value was for the pound sign (#) or *
*   for the 'KEY' token, the routine branches to the pro- *
*   per routines for these commands.  If neither value is *
*   found, then a standard GET command is executed.
*
*****

5612: JSR  $84D9      ;If in the direct mode, then 'SYNTAX ERROR'
5615: STA  $77        ;Save the character after the GET token for use
                        ;By the READ routine (checks token for KEY).
5617: CMP  #'#'       ;Is it GET# ?
5619: BEQ  $5625      ;Yes, then branch to GET #
561B: CMP  #$F9       ;Token for KEY as in GETKEY?
561D: BNE  $5635      ;No, then branch, it's a standard GET

```



```

*-----*
*
*                BASIC GETKEY COMMAND
*
* Command syntax:  GETKEY  variable
*
* This routine will skip the token for KEY ($F9) and
* call the standard GET routine below.
*
*-----*
    
```

```

561F: JSR  $0380      ;Get the next character
5622: JMP  $5635      ;Set up the buffer end, and GET flag,
                    ;Read in the variable
    
```

```

*-----*
*
*                BASIC GET# COMMAND
*
* Command syntax:  GET#  file number, variable
*
* This routine will skip the token for '#' and gets
* the logical file number and places it in the channel
* at location ($15). The routine then falls through
* to the standard GET routine.
*
*-----*
    
```

```

5625: JSR  $0380      ;Skip # char. and get the logical file number
5628: JSR  $87F4      ;Get the byte value for the character
562B: LDA  #','       ;Check to see if it's a comma and if it is, then
562D: JSR  $795E      ;Skip it and get the variable, if there was
5630: STX  $15        ;No comma, then 'SYNTAX ERROR',
                    ;If it is a comma then save the file #
5632: JSR  $90FD      ;Set up the file for input
    
```

```

*-----*
*
*                STANDARD GET COMMAND
*
* This routine puts a delimiter of $00 at $0201 which
* is one byte after where the value that is received
* by the GET or the GETKEY routines will be stored.
* This is the reason why you can only enter one
* character in response to a GET or GETKEY command.
*
*-----*
    
```

```

5635: LDX #1 ;Set the pointer to the start of the input
5637: LDY #2 ;Buffer-1 ($0201)
5639: LDA #0 ;Set the delimiter up for the GET
563B: STA $0201 ;Command
563E: LDA #$40 ;Load the accumulator with the flag for GET
5640: JSR $56B2 ;Set the flag and read in the information
5643: LDX $15 ;Get the current input device
5645: BNE $565A ;If not the keyboard (0), make it the keyboard
5647: RTS
    
```

```

*****
*
*
*          BASIC INPUT# COMMAND
*
* Command syntax: INPUT# file number, variable
*
* This routine takes the ASCII value of the character
* following the pound sign, converts it to integer for-
* mat, and puts the value in location $15 to be used as
* the current input file number. The routine then opens
* the file for input by calling the BASIC CHKIN routine
* and then calling the standard INPUT routine to read
* in the characters until a carriage return has been
* found. The routine then resets the current input file
* number to the default value of $00.
*
*
*****
    
```

```

5648: JSR $87F4 ;Convert ASCII to numeric and place it in X-reg.
564B: LDA #',' ;Check to see if it's a comma
564D: JSR $795E ;If it is, then skip it, if not, 'SYNTAX ERROR'
5650: STX $15 ;Save as the current CHKIN device
5652: JSR $90FD ;Set the file up for input
5655: JSR $5671 ;Input without text such as INPUT#3,A$,B$
5658: LDA $15 ;USELESS INSTRUCTION - HAS NO EFFECT OR USE!
565A: JSR $926F ;Restore system to default I/O devices 0 and 3
565D: LDX #0 ;Make the keyboard the current input device
565F: STX $15 ;For screen prompting
5661: RTS ;Exit
    
```

```

*****
*
*
*          BASIC INPUT COMMAND
*
* Command syntax:INPUT ["text";] variable,[variable etc]
*
*
    
```

```

* This routine will check to see if there is any text *
* to be printed and if there is, then it prints the *
* text and checks to make sure that a semicolon *
* follows. If there is not a semicolon, then it is a *
* 'SYNTAX ERROR'. If there is a semicolon and a *
* valid variable, then this routine will call $56B0 to *
* set the flag for INPUT. The routine then falls *
* through to the rest of the READ routine. *
* *
*****

```

```

5662: CMP #'"' ;Is there a quote mark after token for input?
5664: BNE $5671 ;Branch if there is not one. (No text to print)
5666: JSR $7913 ;There is a quote mark, so set TXTPTR to point
;To the address of the string.
5669: LDA #';' ;Is there a semicolon after the string?
566B: JSR $795E ;If not then 'SYNTAX ERROR'
566E: JSR $55E5 ;Print the string to the screen
5671: JSR $84D9 ;Are we in direct mode, If so, 'SYNTAX ERROR'
5674: LDA #',' ;Comma
5676: STA $01FF ;Place in INPUT buffer
5679: JSR $569C ;Print the prompt and get the input
567C: LDA $15 ;Get the current input device and branch if it's
567E: BEQ $568D ;The keyboard. If not check ST
5680: JSR $9251 ;Read ST if the input is other than the keyboard
5683: AND #2 ;Time out on RS-232 or time out of 64ms?
5685: BEQ $568D ;No, Then branch
5687: JSR $5658 ;Time out has occured so close I/O channels
568A: JMP $528F ;Move the TXTPTR to the end of the statement
568D: LDA $0200 ;Get the character
5690: BNE $56B0 ;Branch if there is a character there
5692: LDA $15 ;Get the current input device
5694: BNE $5679 ;Branch if it's not the keyboard
5696: JSR $52A2 ;Search for a colon
5699: JMP $5292 ;Increment TXTPTR to the byte after the colon

```

```

*****
* *
* This routine checks if the current input device is *
* the keyboard and if it is, the routine prints a *
* question mark and a space before getting an input. *
* If the input device is not the keyboard, then the *
* routine will accept input until a carriage return is *
* received. *
* *

```

```

* If this routine is entered at $56A6, then no      *
* question mark or space will be printed before   *
* accepting an input. The input is stored starting *
* at $0200 and a zero is put at the end of the input. *
*                                                  *
*****

```

```

569C: LDA    $15          ;Get the current input device
569E: BNE    $56A6        ;Branch if it's not the keyboard
56A0: JSR    $560A        ;If it is the keyboard, print a question mark
56A3: JSR    $5604        ;Print a space after the question mark
56A6: JMP    $4F93        ;Get input line and store it in $0200 until a
                          ;Carriage return has been entered

```

```

*****
*
*              BASIC READ COMMAND
*
* Command syntax:  READ variable
*
* The READ routine will get the pointer to the next
* DATA element from $43, $44 (DATPTR). This is the
* address of the next DATA item. The flag is then set
* for READ and the routine falls through to the
* routine below.
*
*****

```

```

56A9: LDX    $43          ;Get the pointers to next DATA element
56AB: LDY    $44

```

```

*-----*
*
*   Enter this routine here to set the flag for READ.
*
*-----*

```

```

56AD: LDA    #$98          ;Flag for READ
56AF: .BYTE $2C          ;Mask to fall through to $56B2

```

```

*-----*
*
*   Enter this routine here to set the flag for INPUT.
*
*-----*

```

```

56B0: LDA    #0           ;Flag for INPUT

```

```

*-----*
*
*   This routine is used by the INPUT, READ, GET, and
*   GETKEY routines.
*
*-----*

```

```

56B2: STA $13 ;Save flag for input type:
; $00=INPUT, $40=GET, $98=READ
56B4: STX $45 ;Save the current DATA element address
56B6: STY $46
56B8: JSR $7AAF ;Check for a variable after the READ command,
; If not found, then create it
56BB: STA $4B ;Save its location
56BD: STY $4C
56BF: LDX #1 ;Index pointer
56C1: LDA $3D,X ;Save the TXTPTR to
56C3: STA $4D,X ;$4D, $4E
56C5: LDA $45,X ;And put the data address
56C7: STA $3D,X ;Into TXTPTR
56C9: DEX ;Decrement the index pointer
56CA: BPL $56C1 ;Not done; branch until loop is executed twice
56CC: JSR $0386 ;Get a character from BASIC text
56CF: BNE $5702 ;Increment BASIC text pointer and get next char.
56D1: BIT $13 ;Check for the type of input
56D3: BVC $56EF ;Is it the flag for GET? If not, then branch
56D5: LDA $77 ;Get the character following the GET command
56D7: CMP #$F9 ;Is it the token for KEY?
56D9: BNE $56E3 ;No, then branch
56DB: JSR $9109 ;Get a byte from current input device (GETIN)
56DE: TAX ;Save it in the X register
56DF: BEQ $56DB ;If it is equal to zero, then loop until
; A key is pressed
56E1: BNE $56E6 ;If the value is not zero, then branch
56E3: JSR $9109 ;Get a byte from current input device (GETIN)
56E6: STA $0200 ;Save it to the input buffer
56E9: LDX #$FF ;Address - 1 of the
56EB: LDY #$01 ;Input buffer
56ED: BNE $56FE ;Unconditional branch

```

```

*-----*
*
*   This routine INPUTs data from the keyboard or READs
*   data from the BASIC text. The routine stores the re-
*   sult into the variable.
*
*-----*

```

```

56EF: BPL   $56F4      ;Branch if it is the flag for input
56F1: JMP   $57CA      ;Search for the next data line
56F4: LDA   $15        ;Check the current input device
56F6: BNE   $56FB      ;If it is not the keyboard, then print a space
56F8: JSR   $560A      ;If it is the keyboard, print a question mark
56FB: JSR   $569C      ;Print a question mark and a space and get INPUT
                        ;from the keyboard. After INPUT is done, place
                        ;the delimiter ($00) at the end of the INPUT.

56FE: STX   $3D        ;Save the address to the input buffer
5700: STY   $3E        ;In TXTPTR ($01FF)
5702: JSR   $0380      ;Get the next character
5705: BIT   $0F        ;Is it a string ($FF), or numeric ($00)
5707: BPL   $573A      ;Branch if it is numeric
5709: BIT   $13        ;Check the GET flag ($40)
570B: BVC   $5716      ;If not there, then branch
570D: INX                   ;LSB + 1
570E: STX   $3D        ;Save in TXTPTR
5710: LDA   #0         ;
5712: STA   $09        ;Clear the search character
5714: BEQ   $5722      ;Unconditional branch

```

```

*-----*
*
* This routine is used if the input type is INPUT or
* READ to check for 'REDO FROM START' errors. On entry
* the accumulato holds the value to be tested.
*
*-----*

```

```

5716: STA   $09        ;Save the value
5718: CMP   #'"'       ;Is it a quote?
571A: BEQ   $5723      ;If it is, then skip
571C: LDA   #':'       ;Value for a colon (:)
571E: STA   $09        ;Save it
5720: LDA   #','       ;Value for a comma (,)
5722: CLC
5723: STA   $0A        ;Save it
5725: LDA   $3D        ;Get TXTPTR LSB
5727: LDY   $3E        ;Get TXTPTR MSB
5729: ADC   #0         ;Add the carry bit, if any
572B: BCC   $572E      ;If the carry is clear, then skip
572D: INY                   ;Increment the MSB
572E: JSR   $86A0      ;Set up the string and allocate the space for it
5731: JSR   $791F      ;Save the string pointer as TXTPTR
5734: JSR   $5405      ;Assign the value to the string
5737: JMP   $5744      ;Check to see if there is another variable to
                        ;Process as in INPUT A$,B$...if so, restart loop

```

```
*****
*
* This routine assigns a numeric or floating point
* number (depending on the INTFLG) to the variable and
* then checks the last character of the input and if
* it is not the end of the input flag #$00 it restarts
* the loop to process the next variable.
*
*****
```

```
573A: LDX #0
573C: JSR $8D22 ;Change ASCII to floating point number
573F: LDA $10 ;Get the INTFLG
5741: JSR $53E3 ;Assign the value to the variable
5744: JSR $0386 ;Get the last character
5747: BEQ $5784 ;If equal to zero, then branch
5749: CMP #$2C ;Is it a comma?
574B: BEQ $5784 ;Yes, then branch
574D: LDA $13 ;Get the flag for INPUT, GET, or READ
574F: BEQ $575B ;If it is zero then it's INPUT, branch
5751: BMI $5757 ;Branch, if it is READ
5753: LDX $15 ;Screen/keyboard current input/output?
5755: BNE $575F ;No, then 'FILE DATA' error
```

```
-----*
*
* TYPMIST
*
* TYPE MISmatch Text
*
* This routine will display the error message 'TYPE
* MISMATCH' and then fall through to the routine
* below.
*
-----*
```

```
5757: LDX #22 ;'TYPE MISMATCH' error
5759: BNE $5761 ;Jump to the error message handler
575B: LDA $15 ;Screen/keyboard current input/output?
575D: BEQ $5764 ;Yes, then 'REDO FROM START' error
```

```
-----*
*
* FILEDAT
*
* FILE Data Text
*
-----*
```

```

* This routine will display the error message 'FILE
* DATA' and then fall through to the routine below.
*
*-----*

575F: LDX #24 ;'FILE DATA' error
5761: JMP $4D3C ;Jump to error message routine

*-----*
*
* REDOT
*
* REDO from start Text
*
* This routine will display the error message 'REDO
* FROM START'then fall through to the routine below.
*
*-----*

5764: JSR $9281 ;Print the following text
5767: TXT '?redo from start'
5777: .BYTE $0D,$00 ;<CR>,Delimiter

*****
*
* This routine saves the line number to CONTINUE to in
* TXTPTR and exits. This routine is usually used to
* re-execute the INPUT line after a 'REDO FROM START'
* error message has been generated.
*
*****

5779: LDA $1202 ;Get the LSB
577C: LDY $1203 ;And the MSB of the line number for CONT
577F: STA $3D ;And save it in TXTPTR
5781: STY $3E
5783: RTS ;Exit

*****
*
* This routine will get the input buffer address and
* save it in INPTR. It then restores TXTPTR to its
* original address and checks for an end of line flag.
*
*****

5784: LDX #$01

```



```

5786: LDA  $3D,X      ;Get the address that TXTPTR is pointing to
5788: STA  $45,X      ;And save it as the INPTR (input buffer address)
578A: LDA  $4D,X      ;Get the old TXTPTR address and
578C: STA  $3D,X      ;Store it to TXTPTR
578E: DEX
578F: BPL  $5786      ;Loop until
5791: JSR  $0386      ;Two bytes are done
5791: JSR  $0386      ;Get current char. that TXTPTR is pointing to
5794: BEQ  $579C      ;If it is equal to zero, then branch
5796: JSR  $795C      ;Check for a comma and
                    ;If it is not there, 'SYNTAX ERROR'
5799: JMP  $56B8      ;Return to the loop
579C: LDA  $45        ;Get the vector pointer to the DATA address
579E: LDY  $46
57A0: LDX  $13        ;Get the flag for INPUT, GET, and READ
57A2: BPL  $57A9      ;If it is not the flag for READ, then branch
57A4: STA  $43        ;Save as the current DATA address
57A6: STY  $44        ;And
57A8: RTS           ;Exit the routine

```

```

*****
*
* This routine will check the first character from
* INPTR and if it is not the value $00, the routine
* will check channel $15 for the current I/O
* configuration. If it is the default configuration
* (keyboard/screen), the 'EXTRA IGNORED' message is
* displayed.
*
*****

```

```

57A9: LDY  #0          ;Get the low byte of vector to the INPUT buffer
57AB: LDA  #$45
57AD: JSR  $039F      ;Get a byte from the input buffer
57B0: BEQ  $57C9      ;None there, then exit
57B2: LDA  $15        ;If the current input device
57B4: BNE  $57C9      ;Is not the keyboard, then exit

```

```

-----*
*
* EXIGNT
*
* EXtra IGnored Text
*
* This routine prints the 'EXTRA IGNORED' error
* message to the screen and exits the routine.
*
-----*

```

```

57B6: JSR   $9281      ;Print the following text and exit
57B9: TXT   '?extra ignored'
57C7: .BYTE $0D,$00    ;<CR>, end of text marker
57C9: RTS                      ;Exit

```

```

*****
*
* This routine will search until it finds the next
* DATA line. It then updates TXTPTR to point to that
* line. If no DATA is found, then the routine
* generates an 'OUT OF DATA' error message and exits.
*
*****

```

```

57CA: JSR   $52A2      ;Search for a colon or a value of $00 and return
                          ;With the offset in the Y register
57CD: INY                      ;Increment the offset by one
57CE: TAX                      ;Transfer the value into the X register
57CF: BNE   $57E6      ;If the value found was $00, then branch
57D1: LDX   #$0D        ;For 'OUT OF DATA' error
57D3: INY                      ;Offset two bytes past the colon
57D4: JSR   $03C9      ;Get the byte
57D7: BEQ   $5819      ;Generate error if a value of $00
57D9: INY                      ;Increment the Y register to the next byte
57DA: JSR   $03C9      ;Get the next byte
57DD: STA   $41         ;Save it as the current DATA line number LSB
57DF: INY
57E0: JSR   $03C9      ;Get the next byte
57E3: INY
57E4: STA   $42         ;And save it as the current DATA line number MSB
57E6: JSR   $5292      ;Update TXTPTR
57E9: JSR   $0386      ;Get a byte
57EC: TAX                      ;Put it into the X register
57ED: CPX   #$83       ;Is it the token for DATA
57EF: BNE   $57CA      ;If not, then keep looping
57F1: JMP   $5702      ;If it is, then branch back to READ the data

```

```

*****
*
*          BASIC NEXT COMMAND
*
* Command syntax:  NEXT [variable]
*
* This routine will search the pseudo stack for the
* next occurrence of 'FOR' and take the starting
* variable and add the step value to the FOR variable.
* The routine then compares the result to the 'TO'
* value and if the loop is completed, the stack
* entries that 'FOR' used will be removed from the
* pseudo stack. If the loop is not finished, then the
* pointers to the current statement and TXTPTR are
* updated which will cause the program to continue at
* the next statement after the 'FOR' statement.
*
*****

```

```

57F4: BNE    $5809    ;If there is a variable after 'NEXT', branch
57F6: LDY    #$FF     ;Flag for no variable after 'NEXT' statement
57F8: BNE    $580E    ;Branch
57FA: LDY    #$12     ;For 18 bytes
57FC: JSR    $5059    ;Add the value to the pseudo stack pointer
57FF: JSR    $0386    ;Get the variable after the 'NEXT' token
5802: CMP    #','     ;Is there a comma?
5804: BNE    $5877    ;No, then no more variables
5806: JSR    $0380    ;Get next variable after the comma
5809: JSR    $7AAF    ;Look for the variable, if not found, create it
580C: STA    $4B     ;Save the variable address
580E: STY    $4C     ;
5810: LDA    #$81     ;Token for 'FOR'
5812: JSR    $4FAA    ;Search for 'FOR' on the stack
5815: BEQ    $581C    ;Branch if found
5817: LDX    #10     ;If not, then
5819: JMP    $4D3C    ;'NEXT WITHOUT FOR' error
581C: JSR    $5050    ;Transfer the pseudo stack pointer from $3F, $40
                    ;To $7D, $7E
581F: LDA    $3F     ;Add three
5821: CLC                    ;To the
5822: ADC    #$03     ;Stack pointer LSB to point to the 'STEP' value
5824: LDY    $40     ;Get the stack pointer MSB
5826: BCC    $5829    ;If there is no overflow, then skip
5828: INY                    ;If there is an overflow, increment MSB by one
5829: JSR    $8BD4    ;Transfer the 'STEP' value to the floating
                    ;Point accumulator

```

---

```

582C: LDY    #8           ;Pointer to the sign of the 'STEP'
                          ;($00 = positive, $FF = negative)
582E: LDA    ($3F),Y     ;Get the sign of the 'STEP' value
5830: STA    $68         ;Save it
5832: LDY    #$01       ;Pointer to LSB of the address to the 'FOR'
                          ;Variable
5834: LDA    ($3F),Y     ;Get the LSB
5836: PHA                    ;Save it on to the stack
5837: TAX                    ;And in to the X register
5838: INY                    ;Index to the MSB of the address to the 'FOR'
                          ;Variable
5839: LDA    ($3F),Y     ;Get the MSB
583B: PHA                    ;Save it on to the stack
583C: TAY                    ;And in to the Y register
583D: TXA                    ;Move the LSB to the accumulator
583E: JSR    $8845       ;Transfer 'FOR' variable value to floating point
                          ;Accumulator
5841: PLA                    ;Get the MSB back
5842: TAY                    ;Put it into the Y register
5843: PLA                    ;Get the LSB back
5844: TAX                    ;Put it into the X register
5845: STA    $FF04       ;Enable RAM BANK 1, BASIC, KERNAL, character ROM
5848: JSR    $8C00       ;Transfer floating point accumulator to stack
584B: LDA    $3F         ;Add nine
584D: CLC                    ;To the
584E: ADC    #$09        ;Stack pointer LSB to get to the 'TO' value
5850: LDY    $40         ;If there is no overflow, then skip
5852: BCC    $5855       ;If there is an overflow, then increment
5854: INY                    ;The stack pointer MSB by one
5855: STA    $FF03       ;Enable BANK 14
5858: JSR    $8C87       ;Compare 'STEP' value to 'TO' value
                          ;$FF = STEP larger,$01 = STEP smaller,$00 = STEP
                          ;Equal
585B: LDY    #$08        ;Pointer to 'STEP' value
585D: SEC                    ;Compares floating point accumulator to the end
585E: SBC    ($3F),Y     ;Value of the loop
5860: BEQ    $57FA       ;If equal, then we are done, so branch
5862: LDY    #$11        ;Pointer to the LSB of the address of the first
                          ;Statement in the loop
5864: LDA    ($3F),Y     ;Get the LSB
5866: STA    $3D         ;Save it to TXTPTR LSB
5868: DEY                    ;Decrement the index to the MSB of the first
                          ;Statement
5869: LDA    ($3F),Y     ;Get the MSB
586B: STA    $3E         ;Save it to TXTPTR MSB
586D: DEY                    ;Decrement the index to MSB of the line number
                          ;Of the first statement

```

```

586E: LDA ($3F),Y ;Get the MSB of the line number
5870: STA $3C ;Save it to the CURLIN MSB
5872: DEY ;Decrement the index to the LSB of the line
;Number of the first statement
5873: LDA ($3F),Y ;Get the LSB of the line number
5875: STA $3B ;Save it to the CURLIN LSB
5877: RTS ;Exit

```

```

*-----*
*
* The DIM command will reenter here if there is more
* then one variable to dimension such as:
* DIM A(100),B(100),C(100)
* After the DIM command has processed the A(100)
* it will reenter here to process the ,B(100) and the
* ,C(100). When this entry point is used the ACC. will
* hold the character ',' and if it does not a
* 'SYNTAX ERROR' will result.
*
*-----*

```

```

5878: JSR $795C ;Checks on comma if found, then skip it and if
;Not it will produce a 'SYNTAX ERROR' message

```

```

*****
*
* BASIC DIM COMMAND
*
* Command syntax: DIM variable (subscripts)
*
* This routine will call PTRGET ($7AB4) for each
* variable that is DIMensioned. If an array element
* is defined BEFORE a DIM statement, the system will
* set the default value DIM value of 10 to be
* assigned. For instance, if you defined C(1)=12
* before your DIM statements, then the system would
* automatically perform a DIM C(10). Just as a
* reminder, when you DIMension an array, remember the
* system considers C(0) as a legal entry.
*
*****

```

```

587B: TAX ;Place variable name in X-register.
587C: JSR $7AB4 ;Dimension variable
587F: JSR $0386 ;Get last character
5882: BNE $5878 ;If one there, then process the next variable
5884: RTS ;Exit

```

```

*****
*
*          BASIC SYS COMMAND
*
* Command syntax:  SYS address [,A][,B][,C][,D]
*
* Optional parameters:
*
*           A = accumulator
*           B = X register
*           C = Y register
*           D = Status register
*
* EXAMPLE:  SYS 32800,123,45,6
*
* NOTE:  When you use the SYS command, you will need
*        to specify which bank number you wish to SYS
*        to such as BANK1:SYS32768.  This particular
*        SYS will call a machine language routine in
*        RAM BANK 1 with the address of 32768.
*        On power up the default BANK is 15
*
* NOTE:  The SYS command can no longer pass string
*        parameters onto a machine language routine
*        like you could with the C-64 as this routine
*        has been changed for the 128.
*
*****

```

```

5885: JSR   $8812      ;Evaluate number and get address of the 'SYS'
                    ;And store the LSB/MSB to $16, $17
5888: LDA   $16        ;Get the MSB of the address to 'SYS' to
588A: STA   $04        ;Save the MSB
588C: LDA   $17        ;Get the LSB of the address to 'SYS' to
588E: STA   $03        ;Save the LSB
5890: LDA   $03D5      ;Get the BANK number for the 'SYS' to 'goto'
5893: STA   $02        ;Save current BANK number
5895: JSR   $9E1E      ;Puts value found after comma into X register
5898: BCC   $589C      ;Branch if comma and/or value not found
589A: STX   $06        ;Save specified accumulator value
589C: JSR   $9E1E      ;Puts value found after comma into X register
589F: BCC   $58A3      ;Branch if comma and/or value not found
58A1: STX   $07        ;Save specified X register value
58A3: JSR   $9E1E      ;Puts value found after comma into X register
58A6: BCC   $58AA      ;Branch if comma and/or value not found
58A8: STX   $08        ;Save specified Y register value

```

```

58AA: JSR   $9E1E      ;Puts value found after comma into X register
58AD: BCC   $58B1      ;Branch if comma and/or value not found
58AF: STX   $05        ;Save specified SR register value
58B1: JMP   $FF6E      ;JSRFAR-JSR to any bank, return to calling bank

```

```

*****
*
*                                     *
*           BASIC TRACE ON COMMAND   *
*
* Command syntax:  TRON              *
*
* This routine places the flag for the TRace ON mode *
* in location $116F (TRCFLG) to inform the interpreter *
* that it can begin printing the line number currently *
* being executed to the screen in brackets.          *
*
* EXAMPLE:  [10] [20] ...           *
*
*****

```

```

58B4: LDA   #$FF      ;Flag for trace mode on
58B6: .BYTE $2C      ;Mask to fall thru to $58B9

```

```

*****
*
*                                     *
*           BASIC TRACE OFF COMMAND  *
*
* Command syntax:  TROFF            *
*
* This routine places the flag for TRace OFF in *
* location $116F (TRCFLG) to inform the interpreter *
* not to print the line number currently being *
* executed.                                       *
*
*****

```

```

58B7: LDA   #0        ;Flag for trace mode off
58B9: STA   $116F     ;Trace mode
58BC: RTS                ;Exit

```

```

*****
*
*
*           BASIC RREG COMMAND
*
* Command syntax:  RREG A [,B] [,C] [,D]
*
* This command is used to read the value of the 8502
* registers after a SYS to a machine language program.
*
* Optional parameters:
*
*           A = accumulator
*           B = X register
*           C = Y register
*           D = status register
*
* NOTE: The variables can be any name, but must be
*       numeric.
*
* With this command it now makes it easier to pass
* numerical data back to BASIC from machine language by
* just loading the values to be passed into the
* registers and doing an RTS.
*
*****

```

```

58BD: LDA    #0           ;Zero out the index pointer and
58BF: STA    $0D          ;Save it in COUNT
58C1: JSR    $0386        ;Get the character after the command
58C4: BEQ    $58FD        ;If there is none, then exit routine
58C6: CMP    #','         ;Compare character to comma
58C8: BEQ    $58EB        ;If it is a comma, then skip it
58CA: JSR    $7AAF        ;Search for variable and create var. in memory
                          ;If necessary
58CD: STA    $4B          ;Save the address of the variable descriptor
58CF: STY    $4C          ;Which is located in RAM BANK 1
58D1: LDA    $0F          ;Check VALTYP to be sure the variable is numeric
58D3: BNE    $58FE        ;If it's the flag for string ($FF), then branch
                          ;To produce a 'TYPE MISMATCH' error
58D5: LDY    $0D          ;Get the counter
58D7: LDA    $0006,Y      ;Get the register pointed to by the counter
58DA: CPY    #$03         ;See if we need to get the Status Register
58DC: BNE    $58E0        ;No, then skip
58DE: LDA    $05          ;Get Status Register value
58E0: TAY                    ;Transfer the value to the Y register as the LSB
58E1: LDA    #0           ;Put a zero in the accumulator for the MSB
58E3: JSR    $793C        ;Convert the value to floating point notation
58E6: LDA    $10          ;Get the flag for integer

```



```

58E8: JSR   $53E3      ;And transfer the value to the variable
58EB: INC   $0D        ;Increment the counter by one
58ED: LDA   $0D
58EF: CMP   #$04        ;See if counter is at maximum for 4 registers
58F1: BCS   $58FD      ;Yes, then exit the routine
58F3: JSR   $0386      ;Get the last accessed character
58F6: BEQ   $58FD      ;If it is a zero or a colon, exit
58F8: JSR   $0380      ;Get the next character
58FB: BNE   $58C1      ;And continue
58FD: RTS
58FE: JMP   $77E7      ;'TYPE MISMATCH' error message
    
```

```

*****
*
*          BASIC MID$ COMMAND          *
*
* Function syntax: MID$ (PAR1, PAR2[, PAR3]) = PAR4 *
*
* Where:   PAR1 = The existing string you wish to *
*           insert PAR4 into.                  *
*           PAR2 = The number of characters from the *
*           left of PAR1 where you wish to *
*           insert PAR4.                        *
*           PAR3 = The number of characters from the *
*           left of PAR4 you wish to place in *
*           PAR1.                               *
*           PAR4 = The existing string you wish to *
*           have inserted into PAR1            *
*
* EXAMPLE: A$="WHAT'S THIS":B$="THAT":MID$(A$,8,4)=B$ *
*
* NOTE: A$ would equal "WHAT'S THAT" in this example. *
*       Also this command's not available on the C-64. *
*
*****
    
```

```

5901: JSR   $7959      ;Search for opening parenthesis '('
5904: JSR   $7AAF      ;Search for variable and create the variable in
                    ;Memory if necessary
5907: STA   $4B        ;Save the LSB
5909: STY   $4C        ;And MSB of the address of the variable
590B: JSR   $77DD      ;Make sure the variable is a string, if not,
                    ;Then 'TYPE MISMATCH' error
590E: JSR   $8809      ;Search for a comma, numeric range, and
                    ;Convert value to an integer
5911: DEX
5912: STX   $78        ;Save the starting position
    
```

---

```

5914:  CMP  #' ) '      ;Closing parenthesis
5916:  BEQ  $591C
5918:  JSR  $8809        ;Check for comma, numeric range, and convert to
                    ;Integer (gets the length, optional)
591B:  .BYTE $2C        ;Mask to fall through to $591E
591C:  LDX  #$FF        ;No second parameter flag
591E:  STX  $77          ;Save the second parameter value
5920:  JSR  $7956        ;Check for closing parenthesis ' ) '
5923:  LDA  #$B2        ;Token for '='
5925:  JSR  $795E        ;Search this BASIC line for the token
5928:  JSR  $77EF        ;Evaluate expresssion
592B:  JSR  $77DD        ;Ensure variable is a string, if not,
                    ;'TYPE MISMATCH' error
592E:  LDY  #$02        ;Index
5930:  LDA  #$4B        ;Pointer to address of string
5932:  JSR  $03AB        ;Get MSB of the address of the string
5935:  STA  $005D,Y      ;Save the MSB
5938:  JSR  $42E7        ;Get a byte from the second string's descriptor
593B:  STA  $0060,Y      ;Save
593E:  DEY
                    ;Until five bytes
593F:  BPL  $5930        ;Are finished
5941:  SEC
                    ;Set the carry for subtraction
5942:  LDA  $61          ;Get the LSB of the second string
5944:  SBC  $78          ;Minus the offset into the first string
5946:  STA  $61          ;Store it back
5948:  BCS  $594C        ;Carry set? OK
594A:  DEC  $62          ;Carry clear, decrement the MSB by one
594C:  LDA  $77          ;Value of length to concatenate
594E:  CMP  $60          ;Compare to the length of the second string
5950:  BCC  $5954        ;If less than, then skip
5952:  LDA  $60          ;Get the length of the second string
5954:  TAX
                    ;Transfer it to the X register
5955:  BEQ  $596F        ;If equal to zero, then exit
5957:  CLC
                    ;Clear the carry for addition
5958:  ADC  $78          ;Add length of the second string to the offset
                    ;Into the first string
595A:  BCS  $5972        ;If overflow, then 'ILLEGAL QUANTITY' error
595C:  CMP  $5D          ;Compare to the length of the first string
595E:  BCC  $5962        ;If less than, OK
5960:  BNE  $5972        ;'ILLEGAL QUANTITY' error
5962:  LDY  $78          ;Get the offset into first string in Y register
5964:  LDA  #$61          ;Pointer to the address of the string
5966:  JSR  $03AB        ;Get a byte from the string
5969:  STA  ($5E),Y      ;Store the value in the string
596B:  INY
                    ;Increment to the next byte
596C:  DEX
                    ;Decrement the counter
596D:  BNE  $5964        ;Continue

```

```
596F: JMP $8781 ;FRMEVL (FoRMula EVaLuation)
5972: JMP $7D28 ;'ILLEGAL QUANTITY' error
```

```
*****
*
*          BASIC AUTO COMMAND          *
*
* Command syntax:  AUTO [increment value] *
*
* This routine will set a flag to inform the BASIC *
* interpreter to begin to automatically number the *
* BASIC program lines that you are entering, starting *
* with the next line number that is entered. The *
* routine will then increment the value of the line *
* number by the increment value that you specified *
* after the AUTO command. If the AUTO command is used *
* without a line increment value, then the AUTO *
* function is turned off. *
*
*****
```

```
5975: JSR $84F0 ;Check for DIRECT mode
5978: JSR $50A0 ;Convert ASCII offset to two byte integer number
                    ;In $16, $17
597B: LDA $16 ;Get the LSB of the offset
597D: STA $74 ;Save it
597F: LDA $17 ;Get the MSB of the offset
5981: STA $75 ;Save it
5983: JMP $4D37 ;Print 'READY' to the screen
```

```

*****
*
*                               BASIC HELP COMMAND
*
* Command syntax:  HELP
*
* This routine is for debugging. It will display the
* BASIC line that caused an error. It will also denote
* the portion of the BASIC line that caused the error
* by printing that portion in reversed characters in
* the forty column mode and by underlining the
* portion that caused the error in the 80-column mode.
*
* NOTE: The HELP command is not always that accurate.
*       For an example enter the following:
*
*       10 RUN"JUNK",B0,D0,U8
*
* Then run the program which will produce an error
* because of the ,B0 but when you enter HELP you
* will not see anything listed. There are several
* bugs with this command, but the problem is with
* separate routines which generate the syntax error*
* message and not with the HELP routine.
*
*****

```

```

5986: LDX    $1208    ;Last error # (ERRNUM)
5989: INX
598A: BEQ    $59A9    ;If equal to 0, exit ($FF = no error, therefore
                    ;$FF + 1 = $00)
598C: LDA    $1209    ;Get the LSB
598F: LDY    $120A    ;And the MSB of the line number of the error
5992: STA    $16      ;Save
5994: STY    $17      ;Them
5996: JSR    $5064    ;Search for the line number that is stored in
                    ;$16, $17
5999: BCC    $59A9    ;If the line number is not found, then exit
599B: ROR    $55      ;Shift the carry into $55 to set the HELP flag
599D: JSR    $5598    ;Print a carriage return or a linefeed
59A0: LDX    $16      ;Get the LSB
59A2: LDA    $17      ;And the MSB of the line number
59A4: JSR    $5123    ;And LIST that line
59A7: LSR    $55      ;Shift the HELP flag out
59A9: JMP    $5598    ;And print a carriage return or linefeed and
                    ;Exit

```

```
*****
*
* This routine is entered here with the Y register
* containing the offset value of the number of
* characters to the part of the BASIC program line
* that contains the error.
*
*****
```

```
59AC: LDX $62 ;Get the MSB of the address of the line
59AE: TYA ;Transfer the offset to the accumulator
59AF: CLC ;Clear the carry for addition
59B0: ADC $61 ;Add the offset to the LSB of the line's address
59B2: BCC $59B5 ;No overflow, then skip
59B4: INX ;Overflow, increment the MSB of line's address
59B5: CPX $120F ;Is the MSB of current address equal to the MSB
;Of the address of where the error is
59B8: BNE $59CE ;If it is not the same, then exit
59BA: CMP $120E ;Is the LSB of current address equal to the LSB
;Of the address of where the error is
59BD: BCC $59CE ;If it is less than, then exit
59BF: BEQ $59CE ;If it is equal, then exit
59C1: LSR $55 ;Shift the HELP flag out
59C3: LDA #$12 ;For the reverse mode on
59C5: BIT $D7 ;If we are in the 40 column mode
59C7: BPL $59CB ;Yes, this is the 40 column mode
59C9: LDA #$02 ;For the underline mode if we are in the 80
;Column mode
59CB: JMP $560C ;Output the value and exit
59CE: RTS ;Exit
```

```
*****
*
* BASIC GOSUB COMMAND
*
* Command syntax: GOSUB line number
*
* This routine pushes various information onto the
* stack (see $5A1D) then calls the GOTO routine below.*
* The routine then jumps to the BASIC interpreter loop.*
*
*****
```

```
59CF: JSR $5A1D ;Save the program and BASIC TEXT pointers
59D2: JSR $0386 ;Get the last character again
59D5: JSR $59DB ;Do a BASIC GOTO command
59D8: JMP $4AF6 ;Jump to the INTERPRETER LOOP
```

```

*****
*
*          BASIC GOTO COMMAND          *
*
* Command syntax:  GOTO  line number  *
*
* This routine will start at the current program line *
* and scan for the line number that it is supposed to *
* 'GOTO' to.  If the line number is smaller than the *
* line number it is on, the routine will begin its *
* scan with the first line in the program.  It will *
* continue to search until the target line number is *
* found.  Then the pointers for the current line *
* number and the TXTPTR are updated to point to the *
* target line number so that program line will be the *
* next to be executed.
*
*****

```

```

59DB: JSR  $50A0      ;Convert ASCII to ADDRESS FORMAT
59DE: LDA  $0A        ;Valid line number?
59E0: BEQ  $5A1A      ;No, then 'SYNTAX' error
59E2: JSR  $52A5      ;Search for the end of the GOTO command
59E5: SEC
59E6: LDA  $3B        ;Subtract the line number we are on
59E8: SBC  $16        ;From the line number we are to GOTO
59EA: LDA  $3C        ;And see if the line number we are to
59EC: SBC  $17        ;GOTO is less than the line number we are on
59EE: BCS  $59FB      ;If it is less than the line number
                    ;We are on, then branch
59F0: TYA
                    ;Place the pointer to the end of the
                    ;Command in the accumulator
59F1: SEC
                    ;Set the carry flag for addition
59F2: ADC  $3D        ;Add the pointer to the end of the command
59F4: LDX  $3E        ;To the low byte of the text pointer
59F6: BCC  $59FF      ;If no overflow, then search for the line number
59F8: INX
                    ;Overflow, then increment the high byte by one
59F9: BCS  $59FF      ;Search for the line number that is in $3D, $3E
59FB: LDA  $2D        ;Get the address of the
59FD: LDX  $2E        ;start of BASIC text
59FF: JSR  $5068      ;Search for the line number in $16, $17
                    ;and place its line link in $61, $62
5A02: BCC  $5A15      ;If the line number is not found, then ERROR
5A04: LDA  $61        ;Get the pointer to address of the line number
5A06: SBC  #1         ;Move the pointer back one to
5A08: STA  $3D        ;Point to the LSB
5A0A: LDA  $62        ;of this line's line

```

```

5A0C: SBC #0 ;Link and place it
5A0E: STA $3E ;in TXTPTR
5A10: BIT $7F ;Make sure we are in the PROGRAM mode
5A12: BPL $5A81 ;If we are in the DIRECT mode, then
;Put us in the PROGRAM mode
5A14: RTS ;Exit the routine
5A15: LDX #17 ;'UNDEF'D STATEMENT' error
5A17: JMP $4D3C ;Jump to error message routine
5A1A: JMP $796C ;'SYNTAX' error
    
```

```

*****
*
* This routine places the current TXTPTR to be used
* as the return address, the current line number, and
* the token for GOSUB ($8D) into the pseudo stack
* in following order:
*
*
* BYTE DESCRIPTION LSB/MSB
* ----
* 4 RETURN ADDRESS MSB
* 3 RETURN ADDRESS LSB
* 2 LINE NUMBER MSB
* 1 LINE NUMBER LSB
* 0 TOKEN FOR GOSUB ($8D) N/A
*
*****
    
```

```

5A1D: LDA #5 ;Make sure there is enough room for 5 bytes
5A1F: JSR $4FFE ;On the pseudo stack
5A22: LDY #4 ;Set index pointer to point to BYTE 4
5A24: LDA $3E ;Get TXTPNTR MSB
5A26: STA ($7D),Y ;Save it as the return address MSB
5A28: DEY ;Decrement the index pointer
5A29: LDA $3D ;Get return address LSB
5A2B: STA ($7D),Y ;Save it on the stack
5A2D: DEY ;Decrement the index pointer
5A2E: LDA $3C ;Get the current line number MSB
5A30: STA ($7D),Y ;Save it onto the stack
5A32: DEY ;Decrement the index pointer
5A33: LDA $3B ;Get the current line number LSB
5A35: STA ($7D),Y ;Save it onto the stack
5A37: DEY ;Decrement the index pointer
5A38: LDA #$8D ;BASIC token for GOSUB
5A3A: STA ($7D),Y ;Save it onto the stack
5A3C: RTS ;Exit routine
    
```

```
*****
*
* If the statement 'GOTO' or 'GO TO' is found,
* then this routine will send you to the BASIC
* GOTO routine. If this statement is not found, then
* this routine will check for the 'GO64' command. If
* it is the 'GO64' command and you are in the DIRECT
* mode, the question 'ARE YOU SURE?' will be asked.
* As long as the first character of the response is a
* 'Y' then the 128 will be configured to the 64 mode.
* If the 'GO 64' command is not found, then a 'SYNTAX
* ERROR' will result.
*
*****
```

```
5A3D: JSR    $0380    ;Get the next character
5A40: CMP    #$A4     ;Is it the token for TO as in GO TO ?
5A42: BNE    $5A4A    ;No, then branch
5A44: JSR    $0380    ;Yes it is, so get the next character
5A47: JMP    $59DB    ;Jump to BASIC GOTO COMMAND
5A4A: JSR    $87F4    ;Convert string to numeric value & check range
5A4D: CPX    #64     ;GO 64?
5A4F: BEQ    $5A54    ;If so, then branch
5A51: JMP    $796C    ;'SYNTAX' error
5A54: JSR    $A7E1    ;Ask question ARE YOU SURE? and await reply
5A57: BNE    $5A5F    ;Answered with 'Y'? If not, then branch
5A59: JSR    $A845    ;Switch to BANK 15
5A5C: JMP    $FF4D    ;Jump to 64 mode
5A5F: RTS                    ;Exit routine
```

```
*****
*
*
*          BASIC CONT COMMAND
*
* Command syntax:  CONT
*
* This routine will enable you to restart the BASIC
* program as long as it was stopped by the RUN/STOP
* key combination or the STOP or END commands by
* resetting the TXTPTR and the current line number to
* where you left off. The routine will also turn off
* the COLLISION, and AUTO modes.
*
* NOTE: Now that the variables are stored in RAM
*       BANK 1, you can restart a program even after
*       you have edited it without fear of destroying
```



```

*           the values of the variables that are stored in *
*           memory by simply using a GOTO command to the *
*           line that you wish to start at.             *
*                                                       *
*****

```

```

5A60: BNE   $5A9A      ;Exit routine if any parameters are after 'CONT'
5A62: BIT   $7F        ;Check to see if we are in the RUN mode
5A64: BMI   $5A9A      ;Branch if we are in the PROGRAM mode
5A66: LDX   #26        ;Error number for 'CAN'T CONTINUE'
5A68: LDY   $1203      ;Is the CONTinue command blocked?
5A6B: BNE   $5A70      ;If it's not blocked, then branch
5A6D: JMP   $4D3C      ;Jump to error message routine
5A70: LDA   $1202      ;CONT. not blocked, so get the pointer to
5A73: STA   $3D        ;Where we left off and restart there
5A75: STY   $3E
5A77: LDA   $1200      ;Get the previous line number we we are on
5A7A: LDY   $1201
5A7D: STA   $3B        ;And make it the current line number
5A7F: STY   $3C
5A81: LDA   #$80       ;Flag for PROGRAM mode
5A83: STA   $7F        ;Set flag to running
5A85: ASL
; * 2 = 00 with carry flag set
5A86: STA   $74        ;Turn off the AUTO
5A88: STA   $75        ;Command pointer
5A8A: STA   $127F      ;INTVAL
5A8D: STA   $F6        ;Turn AUTO INSERT MODE OFF
5A8F: LDX   #2         ;Index pointer
5A91: STA   $1276,X    ;Clear the interrupt storage area
5A94: DEX
;For a the COLLISION COMMAND
5A95: BPL   $5A91      ;Continue to loop until done
5A97: JMP   $FF90      ;Disable all of the KERNAL/BASIC messages
5A9A: RTS             ;Exit routine

```

```

*****
*
*                               BASIC RUN COMMAND
*
* Command syntax:  RUN
*                  RUN line number
*                  RUN " <filename> ", DRIVE#, DEVICE#
*
* This routine gives three options, to RUN a program
* starting with the first line number, to RUN a
* program starting at a specified line number, and to
* load and automatically RUN a program.
*
*****

```

```

5A9B: BEQ   $5AB5      ;End of statement then standard RUN
5A9D: BCC   $5ABB      ;If its a valid line number branch to GOTO

```

```

-----*
*
*                               BASIC RUN a program COMMAND
*
* Syntax:  RUN "filename" [, OPT1] [ON] [, OPT2]
*          RUN (string variable) [, OPT1] [, OPT2]
*
* This routine will set the flag ($40) for the
* auto load and run function to location $7F. Then
* the routine calls the BASIC DLOAD routine at $A1A7
* to load the specified program, sets the system state
* to the PROGRAM mode, prints a RETURN to the current
* output device, and then jumps to the BASIC
* interpreter loop to RUN the program.
*
* Optional parameters:
*
* OPT1 = Drive number which must follow the character
*       D
*
* OPT2 = Device number which must follow the character
*       U which stands for the Unit number.
*
* NOTE: Even though it is not documented by Commodore
* you can assign the file name to a string
* variable as long as you place the variable in
* parentheses;also the optional parameters can be
* any order you wish but they CANNOT be assigned
* to a variable.
*

```

```

* Example: A$="SUSAN GRAY":RUN (A$),U8,D0 *
*                               RUN "SUSAN GRAY",D0,U8 *
*                               *
* Both of the aforementioned examples will do the exact *
* same thing and that is to load a program called *
* SUSAN GRAY from the disk drive with the device number *
* of eight and internal drive number 0. *
* *
*-----*

```

```

5A9F: LDA    #64          ;Flag for auto load and run function
5AA1: STA    $7F          ;Set the flag for the BASIC LOAD routine
5AA3: JSR    $A1A7        ;Call the BASIC DLOAD command
5AA6: JSR    $5A81        ;Set system state to PROGRAM mode
5AA9: JSR    $51F3        ;Set TXTPNTR back to $1C00, and do a CLR CMD
5AAC: JSR    $4F4F        ;Relink BASIC LINE LINKS
5AAF: JSR    $5598        ;Print return or linefeed depending on output
                               ;Device
5AB2: JMP    $4AF6        ;Jump to the INTERPRETER LOOP

```

```

*-----*
*                               *
*           BASIC STANDARD RUN COMMAND *
*                               *
* Command syntax:  RUN *
*                               *
* This routine will set the system pointers to the *
* start of the first line number and call the SETMSG *
* routine to set the flag for the PROGRAM mode. *
* *
*-----*

```

```

5AB5: JSR    $5A81        ;Set system state to PROGRAM mode
5AB8: JMP    $51F3        ;Set TXTPTR to the program start then CLR

```

```

*-----*
*
*          BASIC RUN  line number  COMMAND
*
* Command syntax:  RUN  line number
*
* This routine is similar to the routine above except
* this routine RUNs the program starting with the
* line number that is specified after the RUN command.
* The difference between this command and a GOTO to
* the same line number is that this command will
* perform a CLR command before running the program.
*
*-----*

```

```

5ABB: JSR   $51FA      ;Do a BASIC CLR COMMAND
5ABE: JSR   $0386      ;Get the last BASIC TEXT character again
5AC1: JSR   $59DB      ;BASIC GOTO COMMAND
5AC4: JSR   $5A81      ;Set the system state to the PROGRAM mode
5AC7: JMP   $4AF6      ;Jump to the INTERPRETER LOOP

```

```

*****
*
*          BASIC RESTORE COMMAND
*
* Command syntax:  RESTORE  [line number]
*
* If no line number is indicated after the RESTORE
* command, then the READ pointer is reset to the
* beginning of the BASIC program.  If a line number is
* specified, then the READ pointer is reset to the
* end of the line flag before that line number.  If the
* line that is specified after the RESTORE command is
* not found, then an 'UNDEF'D STATEMENT' error will
* result.
*
*-----*
*****

```

```

5ACA: BEQ   $5AE1      ;Is there a line number? If not, then
                    ;Skip to the normal RESTORE

```

```

*-----*
*
*          RESTORE LINE NUMBER
*
*-----*

```

```

5ACC: JSR   $8812      ;Convert ASCII to the line number format
5ACF: STY   $16       ;Save as the current line number
5AD1: STA   $17
5AD3: JSR   $5064     ;Search for the line number stored in $16, $17
5AD6: BCS   $5ADB     ;If it's a valid line number, then branch
5AD8: JMP   $5A15     ;'UNDEF'D STATEMENT' error
5ADB: LDA   $61       ;Get the LSB
5ADD: LDY   $62       ;And the MSB of the line address
5ADF: BCS   $5AE6     ;Unconditional branch
    
```

```

*-----*
*                                           *
*           STANDARD RESTORE COMMAND           *
*                                           *
*-----*
    
```

```

5AE1: SEC           ;Set the carry flag for subtraction
5AE2: LDA   $2D     ;Get the pointers to the start of BASIC TEXT
5AE4: LDY   $2E
5AE6: SBC   #1      ;LSB minus one (normally $1C00)
5AE8: BCS   $5AEB   ;No overflow, then branch ($2D > $FF)
5AEA: DEY           ;Low byte was less than $FE, so subtract one
                    ;From $2E
5AEB: STA   $43     ;Make it the start of the BASIC DATA addresses
5AED: STY   $44
5AEF: RTS           ;Exit routine
    
```

```

*-----*
*                                           *
*           RNMTAB                               *
*                                           *
*           ReNuMber data TABLE                 *
*                                           *
* This data table is for the RENUMBER routine which *
* uses this table to search through the BASIC program *
* for the tokens which are followed by a line number *
* which will need renumbering.                   *
*                                           *
*-----*
    
```

TOKEN VALUE      COMMAND THE TOKEN REPRESENTS

```

5AF0: .BYTE $89,$8A,$8D ;GOTO, RUN, GOSUB
5AF3: .BYTE $A7,$8C,$D6 ;THEN, RESTORE, RESUME
5AF6: .BYTE $D7,$D5     ;TRAP, ELSE
    
```

```

*****
*
*           BASIC RENUMBER ROUTINE           *
*
* Command syntax:  RENUMBER [par1] [,par2] [,par3] *
*
* NOTE:  par1 = the new starting line number *
*        par2 = increment to add to the line numbers *
*        par3 = optional line number to start *
*              renumbering at *
*
* DEFAULT:  RENUMBER 10,10 *
*
*****

```

```

5AF8: JSR  $84F0      ;If in PROGRAM mode, then 'SYNTAX' error
5AFB: LDA  #0
5AFD: LDX  #10      ;Default start, increment parameters
5AFF: STX  $1170     ;Start at line 10
5B02: STA  $1171     ;$0010
5B05: STX  $1172     ;And increment by 10
5B08: STA  $1173     ;$0010
5B0B: STA  $5C      ;Clear the block transfer
5B0D: STA  $5D      ;Pointers
5B0F: JSR  $0386     ;Get last character back <par1>
5B12: BEQ  $5B68     ;Default, then branch
5B14: JSR  $50A0     ;Convert ASCII to address format <par1>
5B17: LDA  $0A      ;Valid starting line number?
5B19: BEQ  $5B25     ;No, then get <par2>
5B1B: LDA  $16      ;Get the LSB
5B1D: LDX  $17      ;And the MSB of the value of the line number
                    ;To start renumbering at
5B1F: STA  $1170     ;Save them
5B22: STX  $1171     ;For the renumber routine
5B25: JSR  $9E06     ;Get <par2> in the Y register (LSB) and the
                    ;Accumulator (MSB)
5B28: BCC  $5B38     ;If there is no value, then branch
5B2A: STY  $1172     ;Save them
5B2D: STA  $1173     ;For the renumber routine
5B30: ORA  $1172     ;Check if increment value
5B33: BNE  $5B38     ;Is zero
5B35: JMP  $7D28     ;And generate 'ILLEGAL QUANTITY' error if equal
                    ;To zero
5B38: JSR  $9E06     ;Get <par3> in the Y register (LSB) and the
                    ;Accumulator (MSB)
5B3B: BCC  $5B68     ;If there is no value, then branch
5B3D: STY  $5C      ;Save the LSB

```

---

```

5B3F: STY   $16           ;And the MSB
5B41: STA   $5D           ;Of the line number
5B43: STA   $17           ;In <par3>
5B45: JSR   $5064         ;Search for line number in $16, $17 and return
                               ;The address where it was found in $61, $62

5B48: LDA   $61           ;Save the LSB
5B4A: LDX   $62           ;And MSB of the line address
5B4C: STA   $5A           ;For range
5B4E: STX   $5B           ;Checking
5B50: LDA   $1170        ;Save the LSB and
5B53: LDX   $1171        ;The MSB of <par1>
5B56: STA   $16           ;For the
5B58: STX   $17           ;Following routine
5B5A: JSR   $5064         ;Search for line number in $16, $17 and return
                               ;The address where it was found in $61, $62

5B5D: SEC                               ;Set the carry for subtraction
5B5E: LDA   $61           ;Get the LSB of the line address for <par1>
5B60: SBC   $5A           ;Subtract the LSB of the line address of <par3>
5B62: LDA   $62           ;Get the MSB of the line address of <par1>
5B64: SBC   $5B           ;Subtract the MSB of the line address of <par3>
5B66: BCC   $5B35         ;'ILLEGAL QUANTITY' error if <par3> is < <par1>
5B68: JSR   $5D68         ;Transfer <par1> to $65, $64 (LSB, MSB)
5B6B: JSR   $5D9C         ;Get the LSB of the address to the next line
5B6E: INY                               ;Increment to the next byte
5B6F: JSR   $03C9         ;Get the MSB of the address to the next line
5B72: BEQ   $5BAE         ;If equal to zero, end of the program, branch
5B74: INY                               ;Increment to the next byte
5B75: JSR   $03C9         ;Get the LSB of the line number
5B78: SEC                               ;Set the carry for subtraction
5B79: SBC   $5C           ;Subtract the LSB of <par3>
5B7B: INY                               ;Increment to the next byte
5B7C: JSR   $03C9         ;Get the MSB of the line number
5B7F: SBC   $5D           ;Subtract the MSB of <par3>
5B81: BCS   $5B8A         ;Branch if we are at the correct line number
5B83: JSR   $5B9D         ;Make TXTPTR = address of the next line
5B86: BNE   $5B74         ;If not the end of the program flag $00, keep ;
                               ;Searching

5B88: BEQ   $5BAE         ;If it is the end of the program, then branch
5B8A: JSR   $5B9D         ;Make TXTPTR = address of the next line
5B8D: BEQ   $5BAE         ;If it is the end of the program, then branch
5B8F: JSR   $5D89         ;Add increment value to the starting line number
5B92: BCS   $5B98         ;Carry set? Then the line number is too large
5B94: CMP   #$F9         ;Is the line number too large?
5B96: BCC   $5B8A         ;If not, then branch
5B98: LDX   #38           ;'LINE NUMBER TOO LARGE' error
5B9A: JMP   $4D3C         ;Jump to error message routine

```

```

*****
*
* Get the link to the next line and update TXTPTR
* to that address.
*
*****

5B9D: LDY #0 ;Index pointer to the LSB of the line link
5B9F: JSR $03C9 ;Get the LSB of the line link of the next lin
5BA2: TAX ;Move the character into the X register
5BA3: INY ;Increment the index pointer to the MSB
5BA4: JSR $03C9 ;Get the MSB of the line link to the next line
5BA7: BEQ $5BAD ;If equal to zero, then exit
5BA9: STX $3D ;Save the address of
5BAB: STA $3E ;The next line as TXTPTR
5BAD: RTS ;Exit
5BAE: LDA #$01 ;Set the flag so the routine will not
5BB0: STA $77 ;Perform the renumber command
5BB2: LDA $1210 ;Save the LSB
5BB5: LDX $1211 ;And the MSB
5BB8: STA $3F ;Of the start
5BBA: STX $40 ;Of BASIC text
5BBC: JSR $5BFB ;Check the line references & if there is enough
;Memory
5BBF: DEC $77 ;Erase the flag to prevent renumbering
5BC1: JSR $5BFB ;Start renumbering the lines
5BC4: JSR $5D99 ;Move TXTPTR to MSB of the address of the next
;Line and get the MSB
5BC7: BEQ $5BF8 ;If it is the end of the program, then relink
;The program lines and exit
5BC9: JSR $5D9C ;Get the LSB of the line number
5BCC: STA $16 ;Save it
5BCE: INY ;Increment to the next byte
5BCF: JSR $03C9 ;Get the MSB of the line number
5BD2: SEC ;Set the carry for subtraction
5BD3: SBC $5D ;Subtract the MSB of <par3>
5BD5: BCC $5BF0 ;If less than the starting line number,
;Then go to the next line
5BD7: BNE $5BDF ;If larger than, then branch
5BD9: LDA $16 ;If equal to, then
5BDB: SBC $5C ;Check the LSB of <par3>
5BDD: BCC $5BF0 ;If it is less than, then go to the next line
5BDF: LDA $64 ;Get the MSB of the line number
5BE1: STA ($3D),Y ;Save it in the program line
5BE3: DEY ;Index to the LSB of the line number
5BE4: LDA $65 ;Get the LSB of the line number
5BE6: STA ($3D),Y ;Save it in the program line

```



```

5BE8: JSR   $5D9C      ;Increment TXTPTR by one
5BEB: JSR   $5D80      ;Update replacement line number and search for
                    ;The end of the line
5BEE: BEQ   $5BC4      ;Unconditional branch

*-----*
*
* Find the end of the line flag (#$00) and update
* TXTPTR to point to that address.
*
*-----*

5BF0: JSR   $5D9C      ;Increment TXTPTR by one
5BF3: JSR   $5D83      ;Search for the end of the line
5BF6: BEQ   $5BC4      ;Unconditional branch

*-----*
*
* Relink the BASIC lines and exit the routine.
*
*-----*

5BF8: JMP   $5EE5      ;Relink the lines and update end of program
                    ;Pointers
5BFB: JSR   $5254      ;Set TXTPTR to the start of the BASIC text
5BFE: JSR   $5D99      ;Move TXTPTR to MSB of the address of the next
                    ;Line and get the MSB
5C01: BNE   $5C06      ;Branch if it is not the end of the program
5C03: JMP   $5D68      ;Exit
5C06: JSR   $5D9C      ;Get the LSB of the line number
5C09: STA   $4B        ;Save it
5C0B: JSR   $5D9C      ;Get the MSB of the line number
5C0E: STA   $4C        ;Save it
5C10: JSR   $5D9C      ;Get the next character from the line
5C13: CMP   #$22       ;Is it a quote?
5C15: BNE   $5C22      ;If not, then branch
5C17: JSR   $5D9C      ;Get a character from the line
5C1A: BEQ   $5BFE      ;If end of the line, branch to the next line
5C1C: CMP   #$22       ;Is it a quote?
5C1E: BNE   $5C17      ;No, then keep searching
5C20: BEQ   $5C10      ;If not a token, then skip the character
5C22: TAX                      ;Transfer a byte to the X register
5C23: BEQ   $5BFE      ;If equal to zero, then branch to the next line
5C25: BPL   $5C10      ;If it is not a token, then skip the character
5C27: LDX   #$08       ;Index to the token table
5C29: CMP   $5AEF,X    ;Compare token to the tokens in the RNMTAB table
5C2C: BEQ   $5C56      ;If equal, then branch

```

---

```

5C2E: DEX                                ;If not,
5C2F: BNE $5C29                          ;Then check all eight of token values in table
5C31: CMP #$CB                            ;Token for GO?
5C33: BNE $5C40                          ;No, then branch
5C35: JSR $0380                          ;Get the next character from the BASIC text
5C38: BEQ $5BFE                          ;If it's the end of the BASIC line or ':', then
;Branch
5C3A: CMP #$A4                          ;Is it the token for TO?
5C3C: BEQ $5C56                          ;If it is, then branch
5C3E: BNE $5C10                          ;If it's not, then skip the character
5C40: CMP #$FE                          ;Dual token flag?
5C42: BNE $5C10                          ;No, then skip the character
5C44: JSR $0380                          ;Get next character from BASIC text
5C47: BEQ $5C35                          ;End of the line or the ':'? If so, branch
5C49: CMP #$17                          ;Token for COLLISION ?
5C4B: BNE $5C10                          ;No, then skip the character
5C4D: JSR $0380                          ;Get the next character from BASIC text
5C50: BEQ $5C35                          ;End of the line or the ':'? If so, then branch
5C52: CMP #','                          ;Is it a comma?
5C54: BNE $5C4D                          ;No, then keep searching
5C56: LDA $3D                            ;Get the pointer to the
5C58: STA $1200                          ;Next character of the BASIC text and save them
5C5B: LDA $3E                            ;As the
5C5D: STA $1201                          ;Previous line number
5C60: JSR $0380                          ;Get the next character
5C63: BCS $5C13                          ;Branch if it is a character
5C65: JSR $50A0                          ;Convert ASCII line number to integer
5C68: JSR $5D19                          ;Check GOTO, GOSUB, etc. reference
5C6B: LDA $1200                          ;Get the pointer to the previous line
5C6E: STA $3D                            ;Number from the temporary storage location
5C70: LDA $1201                          ;And save as
5C73: STA $3E                            ;TXTPTR
5C75: JSR $0380                          ;Get the next character from the BASIC text
5C78: LDA $3D                            ;Check the LSB of TXTPTR
5C7A: BNE $5C7E                          ;Not equal to zero, then branch
5C7C: DEC $3E                            ;If the LSB equals zero, then decrement the MSB
5C7E: DEC $3D                            ;And the LSB by one
5C80: LDX #$FF                          ;Index to end of buffer
5C82: LDA $77                            ;Check renumber flag
5C84: BEQ $5CC7                          ;If renumbering allowed the change lines
5C86: JSR $5C8F                          ;Check length of line number in buffer to length
;Of the line number in the BASIC program
5C89: CMP #','                          ;Is the next character in the program a comma
5C8B: BEQ $5C56                          ;Yes - branch to get the line number following
;The ','
5C8D: BNE $5C13                          ;If not branch back to the main loop
5C8F: INX                                ;Increment the index pointer

```

```

5C90: LDA    $0101,X    ;Get a character from replacement line number
5C93: BEQ    $5CB4      ;If no more characters then exit the routine
5C95: JSR    $0380      ;Check for an ASCII digit in the BASIC line.
5C98: BCC    $5C8F      ;If found branch
5C9A: INC    $3F         ;Add one to the end of program address
5C9C: BNE    $5CA0
5C9E: INC    $40
5CA0: SEC
5CA1: LDA    $3F         ;And compare to the result of the addition to
5CA3: SBC    $1212      ;HIGHEST address available to a BASIC program
5CA6: LDA    $40
5CA8: SBC    $1213
5CAB: BCS    $5CC4      ;And if larger then 'OUT OF MEMORY' error
5CAD: INX
5CAE: LDA    $0101,X    ;And get the next character
5CB1: BNE    $5C9A      ;then continue the renumbering process
5CB3: RTS
5CB4: JSR    $0380      ;Get the next character in the BASIC line
5CB7: BCS    $5CC3      ;Exit if it is a character
5CB9: LDA    $3F         ;If it's a digit then subtract
5CBB: BNE    $5CBF      ;One from the end of program address
5CBD: DEC    $40
5CBF: DEC    $3F
5CC1: BCC    $5CB4      ;Continue searching line until a non-digit
                          ;Character is found
5CC3: RTS

```

```

*-----*
*
*   Generate an 'OUT OF MEMORY' error
*
*-----*

```

```

5CC4: JMP    $4D3A      ;'OUT OF MEMORY' error

```

```

*****
*
*   This routine is used to place the new ASCII line
*   number into the BASIC line .
*
*****

```

```

5CC7: INX          ;Increment the index pointer
5CC8: LDA    $0101,X ;Get the next digit from the buffer
5CCB: BEQ    $5CF9      ;Branch if no digits left
5CCD: PHA          ;Save digit onto stack
5CCE: JSR    $5D9C      ;Get next character from BASIC line

```

```

5CD1:  CMP    #' : '
5CD3:  BCS    $5CE1      ;Branch if it's a letter
5CD5:  CMP    #$20      ;Is it a space?
5CD7:  BEQ    $5CE1      ;Yes, branch
5CD9:  SEC
5CDA:  SBC    #'0'      ;Subtract $30
5CDC:  SEC              ;then
5CDD:  SBC    #$D0      ;Subtract $D0 to test for a digit 0-9
5CDF:  BCC    $5CF1      ;If it's a digit, branch to store in BASIC line
5CE1:  JSR    $5DA7      ;Move TXTPTR to INDEX1, TXTTOP to INDEX2 & clear
                    ;2 offset pointers
5CE4:  INC    $6D        ;Add 1 to the offset
5CE6:  JSR    $5DDF      ;Insert space in memory for char. of line number
5CE9:  INC    $1210      ;And increment
5CEC:  BNE    $5CF1      ;TXTTOP by one
5CEE:  INC    $1211
5CF1:  PLA              ;Get digit of replacement line number
5CF2:  LDY    #0
5CF4:  STA    ($3D),Y    ;Store it in the BASIC line
5CF6:  INX              ;Index to next character
5CF7:  BNE    $5CC8      ;Continue
5CF9:  JSR    $0380      ;Get the next character from the BASIC line
5CFC:  BCS    $5C89      ;Branch if it's a character
5CFE:  JSR    $5DA7      ;Move TXTPTR to INDEX1, TXTTOP to INDEX2 & clear
                    ;2 offset pointers
5D01:  DEC    $6D        ;Subtract 1 from the offset
5D03:  JSR    $5DC6      ;Delete digit from line if the replacement line
                    ;number was smaller than the old line number
5D06:  LDA    $1210      ;And subtract 1 from TXTTOP
5D09:  BNE    $5D0E
5D0B:  DEC    $1211
5D0E:  DEC    $1210
5D11:  JSR    $0386      ;Get character from line
5D14:  BCC    $5CFE      ;Delete it if it's a digit
5D16:  JMP    $5C89      ;And continue

```

```

*****
*
* This routine checks if the referenced line number in *
* a GOTO ,GOSUB etc. command exists and generates an *
* 'UNRESOLVED REFERENCE' error if it doesn't . *
*
*
*****

```

```

5D19:  JSR    $5D68      ;Put starting line number into $64,$65 MSB,LSB
5D1C:  JSR    $5D99      ;Skip over 1 character and get second character
5D1F:  BNE    $5D2E      ;Branch if it's not the end of the program

```

```

5D21: LDX  #39          ;'UNRESOLVED REFERENCE' error number
5D23: LDA  $4B          ;Save the LSB
5D25: STA  $3B          ;and the MSB of the
5D27: LDA  $4C          ;Referenced line number that wasn't found
5D29: STA  $3C          ;As the current BASIC line number
5D2B: JMP  $4D3C        ;And generate the error
5D2E: JSR  $5D9C        ;Get the LSB of the line number
5D31: STA  $5A          ;Save
5D33: CMP  $16          ;Compare to the current line number
5D35: BNE  $5D5E        ;Branch if they are not the same
5D37: JSR  $5D9C        ;Get MSB of the line number
5D3A: STA  $5B          ;Save
5D3C: CMP  $17          ;Compare to current line number
5D3E: BNE  $5D63        ;Branch if not the same
5D40: SEC
5D41: SBC  $5D          ;Is the LSB of the line number the same as
5D43: BCC  $5D4D        ;The LSB of [PAR3] ? Branch to next line if less
5D45: BNE  $5D55        ;Convert line number to ASCII & exit if larger
5D47: LDA  $16          ;Is the MSB of the line number the same as
5D49: SBC  $5C          ;The MSB of [PAR3] ?
5D4B: BCS  $5D55        ;Convert line number to ASCII & exit if larger
5D4D: LDA  $16
5D4F: STA  $65          ;Save the LSB
5D51: LDA  $17          ;And the MSB
5D53: STA  $64          ;Of the current line number
5D55: LDX  #$90
5D57: SEC
5D58: JSR  $8C75        ;Convert line number in $64,$65 (MSB,LSB) to
                          ;Floating Point format and store in FAC1
5D5B: JMP  $8E42        ;Convert line number to an ASCII string at $0101

```

```

*****
*
* This routine is entered with TXTPTR pointing to the
* LSB of the line number in a BASIC line .
* It then checks if the line number were on is the
* same as the one specified in [PAR3] ,if they aren't
* the same ,the starting line number,[PAR1], is saved
* at $64,$65 and TXTPTR is reset to the start of BASIC
*
*****

```

```

5D5E: JSR  $5D9C        ;Get the MSB of the line number
5D61: STA  $5B          ;Save
5D63: JSR  $5D75        ;Are we on the line number specified in [PAR3]?
5D66: BEQ  $5D1C        ;If we are then branch
5D68: LDA  $1170        ;If not then save the LSB

```

```

5D6B: STA   $65           ;and the MSB
5D6D: LDA   $1171        ;Of the starting line number
5D70: STA   $64
5D72: JMP   $5254        ;Reset TXTPTR to the start of BASIC text, exit

```

```

*****
*
* Check if we're on the same line number as specified *
* in [PAR3]. Move to the next BASIC line if not.      *
*
*****

```

```

5D75: LDA   $5A
5D77: SEC
5D78: SBC   $5C
5D7A: LDA   $5B
5D7C: SBC   $5D
5D7E: BCC   $5D83

```

```

*****
* This routine adds the increment value that was      *
* specified in the RENUMBER command to the current   *
* replacement line number and updates TXTPTR to the next*
* BASIC line                                          *
*****

```

```

5D80: JSR   $5D89        ;Add incre. value to get next line number to add
5D83: JSR   $5D9C        ;Get a character from the BASIC line
5D86: BNE   $5D83        ;Loop until the end of line flag $00 is found
5D88: RTS

```

```

*****
*
* This routine adds the increment value that was      *
* specified in the RENUMBER command to the current   *
* line number.                                        *
*
*****

```

```

5D89: LDA   $65           ;Get the LSB of the current line number
5D8B: CLC
5D8C: ADC   $1172        ;Add the increment value to current line number
5D8F: STA   $65           ;And save it back
5D91: LDA   $64           ;Get the MSB of the current line number
5D93: ADC   $1173        ;Add the increment value to current line number
5D96: STA   $64           ;And save it back
5D98: RTS

```

```
*****
*
*   Increment TXTPTR by two and get a character.
*
*****
```

```
5D99: JSR   $5D9C      ;Increment TXTPTR by one and get the character
```

```
*****
*
*   If the routine is entered here, then increment
*   TXTPTR by one and get the character.
*
*****
```

```
5D9C: LDY   #0
5D9E: INC   $3D      ;Add one to TXTPTR
5DA0: BNE   $5DA4
5DA2: INC   $3E
5DA4: JMP   $03C9    ;Get a character from the BASIC line and exit
```

```
*****
*
*   Move TXTPTR to $24, $25, TXTTOP to $26, $27 and set
*   2 offset pointers at $0D , $6D to $00
*
*****
```

```
5DA7: LDA   $3D
5DA9: STA   $24
5DAB: LDA   $3E
5DAD: STA   $25
5DAF: LDA   $1210
5DB2: STA   $26
5DB4: LDA   $1211
5DB7: STA   $27
5DB9: LDY   #0
5DBB: STY   $0D
5DBD: STY   $6D
5DBF: RTS
```

```

*****
*
*                               BMOVE1
*
* This routine moves a specified block of bytes down
* in memory .
*
* Enter with $24,$25 containing the address of the
* start of the block - 1
* $26,$27 containing the address of the end of the
* block - 1
* $0D containing the offset to the first byte to move
* - 1
* And $6D containing the number of bytes to move the
* block - 1
*
*****

```

```

5DC0: INC   $24      ;Add one to
5DC2: BNE   $5DC6    ;the starting address
5DC4: INC   $25      ;of the block
5DC6: LDY   $0D      ;Get the offset to the first byte to move
5DC8: INY                   ;Add one to it
5DC9: JSR   $4305    ;Get the byte
5DCC: LDY   $6D      ;And the offset of where to move it
5DCE: INY                   ;Add one to the offset
5DCF: STA   ($24),Y  ;Save the byte in new area
5DD1: JSR   $5DEE    ;See if the move is completed
5DD4: BNE   $5DC0    ;Continue if not
5DD6: RTS                   ;Exit

```

```

*****
*
*                               BMOVE2
*
* This routine moves a specified block of bytes up
* in memory.
*
* Enter with $24,$25 containing the address of the
* start of the block + 1
* $26,$27 containing the address of the end of the
* block + 1
* $0D containing the offset to the first byte to move
* and $6D containing the number of bytes to move the
* block
*
*****

```



```

5DD7: LDA   $26           ;Subtract one from
5DD9: BNE   $5DDD        ;The starting address
5DDB: DEC   $27           ;of the block
5DDD: DEC   $26
5DDF: LDY   $0D           ;Get the offset to the first byte to move
5DE1: JSR   $03C0        ;Get a byte
5DE4: LDY   $6D           ;and the offset of where to move it to
5DE6: STA   ($26),Y      ;Save byte in new area
5DE8: JSR   $5DEE        ;See if we're done yet
5DEB: BNE   $5DD7        ;Continue if we're not
5DED: RTS                ;Exit

```

```

*****
*
*   Compare the address in $24,$25 to the address in
*   $26,$27 and exit the routine with the zero flag set
*   if they are equal.
*
*****

```

```

5DEE: LDA   $24
5DF0: CMP   $26
5DF2: BNE   $5DF8
5DF4: LDA   $25
5DF6: CMP   $27
5DF8: RTS

```

```

*****
*
*           BASIC FOR COMMAND
*
* Command syntax: FOR variable = PAR1 TO PAR2[step PAR3]*
*
* Parameters 1 = numeric value to start the loop with
*            2 = numeric value to end the loop with
*            3 = numeric value to increment the loop
*            with.
*
* The variable must be a floating point number but the
* remaining variables can be floating point numbers or
* integers.
*
*****

```

```

5DF9: LDA   #$80           ;Flag for INTEGERS and ARRAY elements
5DFB: STA   $12           ;Are not allowed for the loop variable
5DFD: JSR   $53C6        ;Perform a BASIC LET command

```

---

```

5E00: LDA    #81      ;Token for FOR
5E02: JSR    $4FAA    ;Search the pseudo stack for the token
5E05: BEQ    $5E0F    ;If found then branch
5E07: LDA    #18      ;Make sure there is room for
5E09: JSR    $4FFE    ;18 bytes on the stack
5E0C: JSR    $5047    ;Transfer the stack pointers from $7D, $7E to
                    ;$3F, $40
5E0F: JSR    $5050    ;Transfer the stack pointers from $3F/$40 to
                    ;$7D/$7E
5E12: JSR    $52A2    ;Search BASIC TEXT for a colon
5E15: TYA                    ;Place its location into the accumulator
5E16: LDY    #17      ;Index pointer
5E18: CLC                    ;Clear the carry flag for addition
5E19: ADC    $3D      ;Add the location of the colon to TXTPNTR
5E1B: STA    ($7D),Y  ;And place it into the stack
5E1D: LDA    $3E      ;Get the location of the colon MSB
5E1F: ADC    #0       ;Add the carry flag back in
5E21: DEY                    ;Decrement the index pointer
5E22: STA    ($7D),Y  ;Place it into the stack as the MSB
5E24: LDA    $3C      ;Get the current line number MSB
5E26: DEY                    ;Decrement the index pointer
5E27: STA    ($7D),Y  ;Place it into the stack
5E29: LDA    $3B      ;Get the current line number LSB
5E2B: DEY                    ;Decrement the index pointer
5E2C: STA    ($7D),Y  ;Save it onto the stack
5E2E: LDA    #A4      ;Token for TO
5E30: JSR    $795E    ;Checks a match, if not found, 'SYNTAX ERROR'
5E33: JSR    $77DA    ;Numeric variable?
5E36: JSR    $77D7    ;Gets number from floating point accumu. (FAC)
5E39: LDA    $68      ;Get the value from FAC
5E3B: ORA    #$7F     ;Drop sign bit off
5E3D: AND    $64
5E3F: STA    $64
5E41: LDX    #4       ;Index pointer to floating point accumulator 1
                    ;(FAC1) mantissa
5E43: LDY    #13      ;Index pointer to the stack
5E45: LDA    $63,X    ;Get the value from FAC1
5E47: STA    ($7D),Y  ;Save it onto the stack
5E49: DEX                    ;Decrement the index pointer for the mantissa
5E4A: DEY                    ;Decrement the index pointer for the stack
5E4B: BPL    $5E45    ;Continue the transfer until all bytes are
                    ;Transferred
5E4D: LDA    #$9C     ;Get the value to STEP by (+1) in floating point
                    ;Format
5E4F: LDY    #$89
5E51: JSR    $8BD4    ;Move constant (+1) into FAC1
5E54: JSR    $0386    ;Get the last character read by $0380

```

```

5E57: CMP    #$A9      ;Token for STEP?
5E59: BNE    $5E61     ;If not, then branch
5E5B: JSR    $0380     ;Get the next character
5E5E: JSR    $77D7     ;EVALUATE NUMBER
5E61: JSR    $8C57     ;Get sign of the number (+ or -)
5E64: PHA                    ;Place the sign into the stack
5E65: JSR    $8C47     ;Round FAC1
5E68: PLA                    ;Get sign back
5E69: LDY    #8        ;Index pointer to stack
5E6B: LDX    #5        ;Index pointer for FAC1
5E6D: STA    ($7D),Y   ;Place the character into the stack
5E6F: LDA    $62,X    ;Get the exponent
5E71: DEY                    ;Decrement the index pointer for the stack
5E72: DEX                    ;Decrement the index pointer for FAC1
5E73: BPL    $5E6D     ;Continue until done
5E75: LDA    $4C      ;Get variable-name
5E77: STA    ($7D),Y   ;Place it into the stack
5E79: LDA    $4B      ;Get variable-name LSB
5E7B: DEY                    ;Decrement the index pointer for the stack
5E7C: STA    ($7D),Y   ;Place it into the stack
5E7E: LDA    #$81     ;Token value for FOR
5E80: DEY                    ;Subtract one from the index pointer
5E81: STA    ($7D),Y   ;Place the token into the pseudo stack
5E83: RTS                    ;Exit routine
5E84: JMP    $796C     ;'SYNTAX ERROR'

*****
*
*
*          BASIC DELETE COMMAND
*
* Command syntax:  DELETE first line [-last line]
*
* This routine deletes a single line or a range of
* lines.  If the first line number in the range is
* larger than the second line number, the routine
* exits by relinking the program lines.
*
*****

5E87: JSR    $84F0     ;If we are in PROGRAM mode, generate the error
5E8A: JSR    $0386     ;Get first character after the token for DELETE
5E8D: BEQ    $5E84     ;If there are no parameters following the DELETE
                          ;Command, then 'SYNTAX ERROR'

5E8F: JSR    $5EFB     ;Get the parameters
5E92: LDA    $61      ;Get the LSB
5E94: LDX    $62      ;And the MSB of the address of the line
5E96: STA    $26      ;And save

```

---

```

5E98: STX   $27           ;Them
5E9A: JSR   $5064        ;Search for line number in $16, $17
5E9D: BCC   $5EB4        ;If it was found, then branch
5E9F: LDY   #$01         ;Index to the MSB of the line link to next line
5EA1: JSR   $42EC        ;Get the MSB of the address to the next line
5EA4: DEY           ;Decrement to the next byte
5EA5: TAX           ;Save the MSB in the X register
5EA6: BNE   $5EAD        ;If the MSB is not a zero, then branch
5EA8: JSR   $42EC        ;Get the LSB of the address to the next line
5EAB: BEQ   $5EB4        ;If it is equal to zero, then branch
5EAD: JSR   $42EC        ;If not, then get the value again
5EB0: STA   $61          ;Save the LSB
5EB2: STX   $62          ;And the MSB to the address of the next line
5EB4: LDA   $26          ;Get the LSB of the address to the second line
5EB6: SEC           ;Set the carry for subtraction
5EB7: SBC   $61          ;Subtract the first line address LSB from the
                        ;LSB of the second line
5EB9: TAX           ;Save it in the X register
5EBA: LDA   $27          ;Get the MSB of the address to the second line
5EBC: SBC   $62          ;And subtract the first line address MSB from it
5EBE: TAY           ;Save it in the Y register
5EBF: BCS   $5EE5        ;If the second value is smaller than
                        ;The first, then exit
5EC1: TXA           ;Put the LSB of the offset between
                        ;Lines in the accumulator
5EC2: CLC           ;Clear the carry for addition
5EC3: ADC   $1210        ;Add the offset to the TXTTOP LSB
5EC6: STA   $1210        ;And save it
5EC9: TYA           ;Put the MSB of the offset into the accumulator
5ECA: ADC   $1211        ;Add the offset to the TXTTOP MSB
5ECD: STA   $1211        ;And save it
5ED0: LDY   #0          ;
5ED2: JSR   $42EC        ;Get a byte from the line
5ED5: STA   ($26),Y     ;Store it at the address of first line to delete
5ED7: INY           ;Continue
5ED8: BNE   $5ED2        ;Until
5EDA: INC   $62          ;The
5EDC: INC   $27          ;MSB
5EDE: LDA   $1211        ;Of the target address
5EE1: CMP   $27          ;Is equal to the value in the TXTTOP MSB
5EE3: BCS   $5ED2        ;Loop until equal
5EE5: JSR   $4F4F        ;Relink the BASIC line links
5EE8: LDA   $24          ;Get the LSB
5EEA: LDX   $25          ;And MSB of address of the end of the program
5EEC: CLC           ;Clear the carry for addition
5EED: ADC   #$02        ;Add two to the end of program address to point
                        ;It to third 0 byte in the end of program flag

```

```

5EEF: STA $1210 ;Save it as the TXTTOP LSB
5EF2: BCC $5EF5
5EF4: INX ;Add one to the MSB of TXTTOP
5EF5: STX $1211 ;Save it as the TXTTOP MSB
5EF8: JMP $4D37 ;Print 'READY' to the screen

*****
*
* FROMTO
*
* This routine checks on the starting line number to
* the ending line number in such commands as
* LIST [par1] [- par2] and DELETE [par1] [-par2].
*
*****

5EFB: BEQ $5F0F ;If there is no parameter after the token this
;Routine will assume you wish to LIST the entire
;Program, If it is for the DELETE routine, the
;Delete routine itself calls for a syntax error

5EFD: BCC $5F0F ;If line number, then branch
5EFF: CMP #'-' ;Token for subtraction? (through)
5F01: BNE $5F0F ;No, then branch
5F03: LDY #1 ;Index pointer
5F05: JSR $03C9 ;Move over one place and read the next character
5F08: BEQ $5F31 ;If the end of BASIC line or a colon
;Is found, then branch

5F0A: CMP #' ':' ;Colon?
5F0C: BEQ $5F31 ;Yes, then 'SYNTAX ERROR'
5F0E: SEC ;Set the carry
5F0F: JSR $50A0 ;Convert ASCII to address format and
;Store it in $16, $17

5F12: JSR $5064 ;Search for line number in $16, $17
5F15: JSR $0386 ;Get the character again
5F18: BEQ $5F26
5F1A: CMP #$AB ;Token for subtraction? (through)
5F1C: BNE $5F31 ;No, then branch
5F1E: JSR $0380 ;Get the next character
5F21: JSR $50A0 ;Convert ASCII to address format
5F24: BNE $5F31 ;No line number, then 'SYNTAX ERROR'
5F26: LDA $0A ;Valid line number to
5F28: BNE $5F30 ;List to? If so, then exit
5F2A: LDA #$FF ;If there is no line number,
5F2C: STA $16 ;Then set the line number to $FFFF
5F2E: STA $17
5F30: RTS ;Exit routine
5F31: JMP $796C ;'SYNTAX ERROR'

```

```

*****
*
*          BASIC PUDEF COMMAND
*
* Command syntax:  PUDEF  string
*
* NOTE: string = any combination of characters
*           up to four characters
*
* Where: Position 1 (lxxx) = redefinition for a space
*        Position 2 (x2xx) = redefinition for comma
*        Position 3 (xx3x) = redefinition for deciml pt*
*        Position 4 (xxx4) = redefinition for a $
*
* This routine is used to define the characters in
* PRINT USING command and it will redefine any of the
* the four main formatting characters.
*
*****

```

```

5F34: JSR   $877B      ;Get the length of the string
5F37: TAY           ;Transfer it into the Y register
5F38: DEY           ;Decrement it by one to get the actual length
5F39: CPY   #$04      ;If there were less than 5 chars. entered, then
5F3B: BCC   $5F40      ;Branch (if length -1 is < 4 then branch)
5F3D: JMP   $7D28      ;If not, then 'ILLEGAL QUANTITY' error
5F40: JSR   $03B7      ;Get a character from the string
5F43: STA   $FF03      ;Enable BANK 14
5F46: STA   $1204,Y    ;Store each character in the PUDEF Table + Y
5F49: DEY           ;Decrement the string length by one
5F4A: BPL   $5F40      ;Continue looping until all the
                    ;Characters are redefined
5F4C: RTS           ;Exit

```

```

*****
*
*          BASIC TRAP COMMAND
*
* Command syntax:  TRAP  [line number]
*
* This routine will read in what line number the
* program wants to goto when the system finds an
* error. If no line number is specified, then the
* TRAP command is turned off.NOTE: The line number
* can also be a numeric variable.
*
*****

```

```

5F4D: JSR    $84D9      ;Check to see if we are in the DIRECT mode and
                        ;If we are, then 'ILLEGAL DIRECT' error
5F50: JSR    $0386      ;Get the last character back
5F53: BEQ    $5F5C      ;No line number to execute on error
5F55: JSR    $8812      ;Get the line number to execute on error
5F58: STY    $120B      ;Save low byte of the address of the line number
5F5B: .BYTE  $2C        ;Mask to bypass the TRAP off flag.
5F5C: LDA    #$FF       ;Turn off TRAP flag.
5F5E: STA    $120C      ;Save the high byte (or flag) for line number
5F61: RTS

```

```

*****
*
*                               *
*          BASIC RESUME COMMAND *
*                               *
* Command syntax: RESUME [line number] *
*                   RESUME NEXT      *
*                               *
* This routine will try to restart program execution *
* by executing the specified line number. If the *
* RESUME NEXT is used, the system will attempt to *
* restart program execution at the statement following *
* the statement which had the error in it. If neither *
* are requested then the system will re-execute the *
* statement which caused the error to occur. *
*                               *
*****

```

```

5F62: JSR    $84D9      ;Check to see if we are in the DIRECT mode
                        ;And if we are, then 'ILLEGAL DIRECT' error
5F65: LDX    $120A      ;Get the high byte of line the error occurred in
5F68: INX                    ;Increment the address by one to see if an error
                        ;Has occurred (If no error has occurred,
                        ;Then $120A will equal $FF)
5F69: BEQ    $5FDB      ;If no error has occurred, then generate a
                        ;'SYNTAX'error
5F6B: JSR    $0386      ;Get the character after the RESUME token
5F6E: BEQ    $5FB7      ;No parameters, then try to restart the program
                        ;At the statement that caused the error
5F70: BCC    $5FAC      ;Branch if there is a line number after the
                        ;RESUME token,
5F72: CMP    #$82       ;Is it the token for NEXT? (RESUME NEXT)
5F74: BNE    $5FD8      ;No, then branch to generate a 'SYNTAX ERROR'
                        ;message

```

```

*****
*
*                               RESUME NEXT
*
*****

```

```

5F76: JSR   $5FB7      ;Reconfigure the various pointers to attempt
                        ;A restart sequence
5F79: LDY   #0         ;Index pointer
5F7B: JSR   $03C9      ;Get the next byte in the line and see if there
                        ;Is a statement or character after the error
5F7E: BNE   $5FA6      ;If not equal to zero, then branch to skip over
                        ;The error to the next statement
5F80: INY                   ;If it was equal to zero, then check for the end
                        ;Of the program
5F81: JSR   $03C9      ;Get the next byte - LSB of the line link
5F84: BNE   $5F8F      ;Branch if not equal to zero
                        ;(not the end of the program)
5F86: INY                   ;If it was equal to zero, then

```

```

*****
*
*                               RESUME
*
*****

```

```

5F87: JSR   $03C9      ;Check the next byte and if the MSB equals $00,
                        ;It is the end of the program
5F8A: BNE   $5F8F      ;If it was not equal to zero, then branch
5F8C: JMP   $4D37      ;Print 'READY' to screen, it's end of program
5F8F: LDY   #$03        ;Index to LSB of the next line number
5F91: JSR   $03C9      ;Get the LSB of the line number
5F94: STA   $3B        ;Save it
5F96: INY                   ;Increment to the MSB
5F97: JSR   $03C9      ;Get the MSB of the line number
5F9A: STA   $3C        ;And save it
5F9C: TYA                   ;Transfer the index to the accumulator
5F9D: CLC                   ;Clear the carry for addition
5F9E: ADC   $3D        ;Add the offset to the LSB of TXTPTR
5FA0: STA   $3D        ;To point to the first statement in line that
                        ;Follows the line in which the error occurred
5FA2: BCC   $5FA6      ;If no overflow, then skip
5FA4: INC   $3E        ;Increment the MSB
5FA6: JSR   $0380      ;Get the next character in the line
5FA9: JMP   $528F      ;Perform a DATA command to skip over the next
                        ;Statement

```



```
*****
*
*           RESUME LINE NUMBER           *
*
*****
```

```
5FAC: JSR   $8812      ;Evaluate the expression after the RESUME token
                        ;And convert the numeric result to a HEX line
                        ;Number with the result in $16, $17
5FAF: STA   $17       ;Useless code - the MSB was placed here by the
                        ;GET ADDRESS routine at $8812
5FB1: JSR   $5FC6     ;Clear the error number and
5FB4: JMP   $59FB     ;Execute a BASIC GOTO command
```

```
*****
*
* Set up the current line number and line address
* to attempt to restart the program.
*
*****
```

```
5FB7: LDX   #$01      ;Get line number that the error occurred in from
5FB9: LDA   $1209,X   ;Location $1209 (ERRLIN) and make it the current
                        ;Line number we are on
5FBC: STA   $3B,X     ;By placing it in location $3B (CURLIN)
5FBE: LDA   $120E,X   ;Get the address of the byte just before the
                        ;Statement that caused the error
5FC1: STA   $3D,X     ;From location $12CE (ERRTXT) and place it in
                        ;TXTPTR to attempt to restart the program
5FC4: BPL   $5FB9     ;Continue til LSB and MSB have been transferred
```

```
*****
*
* Clear the various pointers in preparation for a
* system restart after an error.
*
*****
```

```
5FC6: LDX   #$FF
5FC8: STX   $1208     ;Clear the last error number (ER - value)
5FCB: STX   $1209     ;Clear the line number that an error
5FCE: STX   $120A     ;Has occurred in ($FFFF indicates no error)
5FD1: LDX   $120D
5FD4: STX   $120C
5FD7: RTS                ;Exit
5FD8: JMP   $796C     ;'SYNTAX ERROR' error
5FDB: LDX   #28       ;'CAN'T RESUME' error
```

5FDD: JMP \$4D3C ;JUMP to the error message routine

```
*****
*
*          BASIC DO COMMAND          *
*
* Command syntax: DO [mod1] statements [exit] *
*                LOOP [mod2]         *
*
* NOTE:  mod1 = optional modifier UNTIL or WHILE *
*        exit = EXIT modifier          *
*        mod2 = optional modifier UNTIL or WHILE *
*
* This routine is similar to NEXT except this routine *
* does not have a specific number of times to loop. In- *
* stead, the routine will test for modifiers as WHILE, *
* UNTIL, and EXIT for a way out of the DO-LOOP. If none *
* of these modifiers are found, the only way out of the *
* DO-LOOP is to press the RUN/STOP key.          *
*
*****
```

```
5FE0: LDY #1 ;Index pointer
5FE2: LDA $003D,Y ;Save the current TXTPTR
5FE5: STA $1214,Y ;In TMPTXT
5FE8: LDA $003B,Y ;Save the current line number
5FEB: STA $1216,Y
5FEE: DEY ;Decrement the index pointer
5FEF: BPL $5FE2 ;Continue until two bytes are done
5FF1: JSR $0386 ;Get the character after the DO token
5FF4: BEQ $6012 ;If end of statement, then branch
5FF6: CMP #$FC ;Token for UNTIL?
5FF8: BEQ $600B ;If so, then branch to the UNTIL command
5FFA: CMP #$FD ;Token for WHILE?
5FFC: BNE $6041 ;If not, then generate a 'SYNTAX' error message.
5FFE: JSR $60DB ;Evaluate the expression after the WHILE token
```

```
*****
*
*          LOOP/WHILE          *
*
* This routine is called if WHILE is found after DO. If *
* the expression after WHILE is true, the DO parameters *
* are saved into the pseudo stack to force the LOOP to *
* continue execution. A false expression ends the loop. *
*
*****
```

```

6001: LDA  $63      ;If the result of the expression
6003: BNE  $6012    ;Was true, then branch
6005: JSR  $0386    ;Get a character from the BASIC text
6008: JMP  $6047    ;JuMP to process the character
    
```

```

*****
*
*          BASIC UNTIL COMMAND
*
* Command syntax: DO UNTIL condition ..... LOOP
*                  DO ..... LOOP UNTIL condition
*
* This command is actually an extension or modifier to
* the DO/LOOP command statement. The UNTIL command
* allows you to exit the DO/LOOP when the specified
* 'condition' is met.
*
*****
    
```

```

600B: JSR  $60DB    ;Evaluate the expression after 'UNTIL'
600E: LDA  $63      ;If the result of the expression
6010: BNE  $6005    ;Was true, then branch
    
```

```

*****
*
* This routine will push the current TXTPTR to be used
* as the return address, the current line number, and
* the token for DO ($EB) onto the pseudo stack in the
* following order:
*
*
* BYTE      DESCRIPTION      LSB/MSB
* ----      -
* 4         RETURN ADDRESS    MSB
* 3         RETURN ADDRESS    LSB
* 2         LINE NUMBER        MSB
* 1         LINE NUMBER        LSB
* 0         TOKEN FOR DO ($EB) N/A
*
*****
    
```

```

6012: LDA  #$05      ;Load the accumulator with the number of bytes
                        ;To free on the pseudo stack for 'DO'
6014: JSR  $4FFE    ;Check on the free stack space and return with
                        ;The stack address in locations $7D, $7E
6017: STA  $FF03    ;Enable BANK 14
601A: LDY  #4       ;Load the Y register with the number of bytes
                        ;Minus one to be put onto the pseudo stack
    
```

```

601C: LDA   $1215      ;Get the MSB of the address of the next
                        ;Statement in the line of the 'DO' command
601F: STA   ($7D),Y   ;Save it onto the pseudo stack
6021: DEY                        ;Move to the next byte
6022: LDA   $1214      ;Get the LSB of the address
6025: STA   ($7D),Y   ;And save it onto the pseudo stack
6027: DEY                        ;Move to the next byte
6028: LDA   $1217      ;Get the MSB of the line number of the 'DO'
602B: STA   ($7D),Y   ;Command and save it onto the pseudo stack
602D: DEY                        ;Move to the next byte
602E: LDA   $1216      ;Get the LSB of the line number of the 'DO'
6031: STA   ($7D),Y   ;Command and save it onto the pseudo stack
6033: DEY                        ;Move to the next byte
6034: LDA   #$EB       ;Token for BASIC DO command
6036: STA   ($7D),Y   ;Save it onto the pseudo stack
6038: RTS                        ;Exit this routine

```

```

*****
*
*                               *
*          BASIC EXIT COMMAND   *
*                               *
* Command syntax: DO...IF expression THEN EXIT...LOOP *
*                               *
* While the UNTIL and WHILE commands must be          *
* stated at the beginning or end of a DO/LOOP          *
* command statement, EXIT allows you to stop the      *
* execution of the loop in the middle.                 *
*                               *
*****

```

```

6039: JSR   $609B      ;Search for 'DO' on the pseudo stack and point
                        ;The pseudo stack pointer to it
603C: JSR   $0386      ;Get the character following the 'EXIT' command
603F: BEQ   $6047      ;If it is the chain flag ':' or the end of line
                        ;Flag, then branch
6041: JMP   $796C      ;Generate a 'SYNTAX' error
6044: JSR   $0380      ;Get the next character in the BASIC text
6047: BEQ   $6060      ;If it is the end of statement or end of line,
                        ;Then branch
6049: CMP   #$EC       ;Is it the token for BASIC LOOP command?
604B: BEQ   $6087      ;If it is, then branch
604D: CMP   #'"'       ;Is it a quote mark?
604F: BEQ   $605B      ;If it is, then branch.
6051: CMP   #$EB       ;Is it the token for BASIC DO command?
6053: BNE   $6044      ;If not, then branch
6055: JSR   $6044      ;Check the command following the 'DO'
6058: JMP   $6005      ;And process it

```

```

605B: JSR   $537C   ;Search for quote mark or end of line terminator
605E: BNE   $6044   ;If it is not the end of line terminator or the
                    ;Chain flag ':', then branch
6060: CMP   #' : '   ;Is it a colon?
6062: BEQ   $6044   ;If it is, then branch
6064: BIT   $7F     ;If it is not a colon, then check to see if
                    ;We are in the DIRECT mode
6066: BPL   $60AA   ;If in DIRECT mode, then generate an error
6068: LDY   #$02    ;Index to the MSB of the next line's line link
606A: JSR   $03C9   ;Get MSB of the line link to the next BASIC line
606D: BEQ   $60AA   ;If it is a zero, then generate an error
606F: INY                   ;Move to the LSB of the line number
6070: JSR   $03C9   ;Get the LSB of the next line number
6073: STA   $3B     ;And save it
6075: INY                   ;Move to the MSB
6076: JSR   $03C9   ;Get the MSB of the next line number
6079: STA   $3C     ;And save it
607B: TYA                   ;Move the index to the accumulator
607C: CLC                   ;Clear the carry for addition
607D: ADC   $3D     ;Add the index to TXTPTR LSB to move it to
                    ;The next line
607F: STA   $3D     ;And save it
6081: BCC   $6044   ;If there was no overflow, branch to continue
6083: INC   $3E     ;If overflow did occur, then increment the MSB
                    ;Of TXTPTR
6085: BNE   $6044   ;And branch to continue skipping over the
                    ;Remaining text
6087: JMP   $528F   ;By calling the BASIC DATA command

```

```

*****
*
*           BASIC LOOP COMMAND
*
* Command syntax: DO .... LOOP
*
* This command marks the end of a DO/LOOP command
* statement. The statements that are inside the loop
* will be executed without end unless an UNTIL, WHILE,
* or EXIT command is used to exit the loop.
*
*****

```

```

608A: BEQ   $60C1   ;If no command exists after the 'LOOP' command,
                    ;Then branch
608C: CMP   #$FD    ;Is the next character the token for 'WHILE'?
608E: BEQ   $60BC   ;If it is, then branch
6090: CMP   #$FC    ;Is the next character the token for 'UNTIL'?

```

```

6092: BNE    $6041    ;If not, then generate a 'SYNTAX' error
6094: JSR    $60DB    ;Evaluate the expression following the 'UNTIL'
6097: LDA    $63      ;If the result of the expression
6099: BEQ    $60C1    ;Was false, then branch
609B: LDA    #$EB     ;Find the token for the
609D: JSR    $4FAA    ;BASIC DO command on the stack
60A0: BNE    $60B7    ;If the token for 'DO' is not on the stack,
        ;Then generate a 'LOOP WITHOUT DO' error
60A2: JSR    $5050    ;Move the pseudo stack pointer
        ;From locations $3F, $40 to locations $7D, $7E
60A5: LDY    #5      ;Load the Y register with the number of bytes
        ;For the 'DO' entry on the pseudo stack
60A7: JMP    $5059    ;Point pseudo stack pointer to the 'DO' token

```

```

*-----*
*
* This routine will take the line number that the DO
* statement is on and make it the Current line number
* to be used by the error message handler routine.
* The routine then falls thru to generate a
* 'LOOP NOT FOUND' error.
*
*-----*

```

```

60AA: LDA    $1216    ;Get the line number that the
60AD: LDX    $1217    ;DO statement is on and
60B0: STA    $3B      ;Make it the
60B2: STX    $3C      ;Current BASIC line number

```

```

*****
*
* GENERATE A 'LOOP NOT FOUND' ERROR MESSAGE
*
*****

```

```

60B4: LDX    #32      ;'LOOP NOT FOUND' error number
60B6: .BYTE $2C      ;Mask to fall through to $60B9

```

```

*****
*
* GENERATE A 'LOOP WITHOUT DO' ERROR MESSAGE
*
*****

```

```

60B7: LDX    #33      ;'LOOP WITHOUT DO' error number
60B9: JMP    $4D3C    ;Jump to the error message routine
60BC: JSR    $60DB    ;Evaluate expression after the 'WHILE' command

```

```

60BF: BEQ    $609B    ;If the expression is false, then branch
60C1: JSR    $609B    ;Search for the 'DO' token on the pseudo stack
                        ;And update the pseudo stack pointer to point to
                        ;That token

60C4: DEY
60C5: LDA    ($3F),Y  ;Get the MSB of the address of the next
                        ;After the 'DO' statement

60C7: STA    $3E      ;And save it as the MSB of TXTPTR
60C9: DEY
60CA: LDA    ($3F),Y  ;Get the LSB of the address of the next
                        ;Statement after the 'DO' statement

60CC: STA    $3D      ;And save it as the LSB of TXTPTR
60CE: DEY
                        ;Move to the MSB of the line number of
                        ;The 'DO' statement

60CF: LDA    ($3F),Y  ;Get the MSB of the line number
60D1: JSR    $A83B    ;And save as the MSB of the current line number
60D4: LDA    ($3F),Y  ;Get the LSB of the line number
60D6: STA    $3B      ;And save as the LSB of the current line number
60D8: JMP    $5FE0    ;Execute the 'DO' statement

```

```

*****
*
* This routine is used to evaluate the expression that *
* is after the 'UNTIL' or 'WHILE' commands. On entry *
* to this routine, TXTPTR must point to the first *
* character before the expression. On exiting the *
* routine, FAC1 will hold a zero if the expression *
* was false.
*
*****

```

```

60DB: JSR    $0380    ;Get the next character of the BASIC text
60DE: JMP    $77EF    ;Evaluate the expression

```

```

*****
*
* This routine is used by the BASIC KEY command to *
* define a programmable key.
*
*****

```

```

60E1: JSR    $87F4    ;Get the KEY number into the X register
60E4: DEX
60E5: CPX    #8        ;Is between 1 and 7,
60E7: BCC    $60EC    ;Then branch
60E9: JMP    $7D28    ;If the KEY number is 0 or greater than 7, then
                        ;Generate an 'ILLEGAL QUANTITY' error

```

```

60EC: STX   $77           ;Save the KEY number
60EE: JSR   $795C        ;Check for a comma and if it is not found, then
                        ;Generate a 'SYNTAX' error
60F1: JSR   $877B        ;Evaluate the KEY definition, store address in
                        ;Locations $24,$25 and length in the accumulator
60F4: TAY
60F5: LDA   #1           ;Move definition length into the Y register
60F7: STA   $26          ;Set the BANK number of the KEY definition
                        ;To RAM BANK 1
60F9: LDA   #$24         ;Get the pointer to the KEY definition address
60FB: LDX   $77          ;Get the KEY number (-1) into the X register
60FD: INX
                        ;And restore it to its original value
60FE: JSR   $A845        ;Enable BANK 15
6101: JSR   $FF65        ;Define the specified KEY
6104: BCS   $6107        ;If there's not enough room for the definition,
                        ;Then generate an 'OUT OF MEMORY' error
6106: RTS
6107: JMP   $4D3A        ;Exit this routine
                        ;Generate an 'OUT OF MEMORY' error

```

```

*****
*
*
*           BASIC KEY COMMAND
*
* Command syntax:  KEY
*                  KEY key number, string
*
* This command allows you to define the function keys
* F1 - F8. For instance, if you wish to define F1 to
* print the current system error message when that key
* is depressed, you would enter the following
* statement:  KEY 1, "PRINT ERR$(ER)". If the 'KEY'
* command is used by itself, the current key
* definitions will be displayed to the screen.
*
* This routine first checks to see if a key is to be
* defined or if the key definitions are to be
* displayed. This routine also reveals the use of the
* two commands 'ON' and 'OFF', one of which (OFF) is
* unimplemented. Apparently, there was supposed to
* have been two extra commands 'KEY ON' and 'KEY OFF'
* which would have enabled or disabled the KEY
* definitions. However, neither of these commands
* was ever implemented. If either command is entered,
* an 'UNIMPLEMENTED COMMAND' error will result.
*
*****

```



---

```

610A: BEQ   $6121      ;If the KEY command was used by itself, then
                        ;Branch to output the current KEY definitions
610C: CMP   #$91      ;If the command sequence
610E: BEQ   $611E      ;'KEY ON' was used, then generate
                        ;An 'UNIMPLEMENTED COMMAND' error
6110: CMP   #$FE      ;If the command sequence was not used, then
6112: BNE   $60E1      ;Check to see if the 'KEY OFF'
6114: JSR   $0380      ;Command sequence was used
6117: CMP   #$24      ;And if it was,
6119: BEQ   $611E      ;Then generate an 'UNIMPLEMENTED COMMAND' error
611B: JMP   $796C      ;Generate a 'SYNTAX' error
611E: JMP   $4846      ;Generate an 'UNIMPLEMENTED COMMAND' error
6121: LDX   #0         ;Clear the index register to the text to be
                        ;Printed and to the key definition table
6123: LDY   #0         ;Index pointers
6125: INX                   ;Subtract one from the index register
6126: LDA   $0FFF,X     ;Get the length of the KEY definition
6129: BEQ   $617E      ;If the length is zero, then branch to move to
                        ;The next definition
612B: STA   $78         ;Save the length of the KEY definition
612D: STX   $77         ;Save the KEY X number
612F: LDX   #5         ;Index of the number of characters to print
6131: LDA   $A82A,X    ;Get a character of 'KEY 0'
6134: DEX                   ;Move the index to the next character
6135: BNE   $6139      ;If we are still doing the letters, then branch
6137: ORA   $77         ;If we are not, then create the KEY number by
                        ;Combining the index with the zero in KEY 0
6139: JSR   $9269      ;Output the character
613C: TXA                   ;Move the index into the accumulator
613D: BPL   $6131      ;Continue output until five characters are done
613F: LDX   #$07       ;Set up the output flag
6141: LDA   $100A,Y    ;Get a character of the KEY definition
6144: INY                   ;Move the index to the next character
6145: PHA                   ;Save the character onto the stack
6146: STX   $79         ;Save the output flag
6148: LDX   #$04       ;Set up the index to the control character table
614A: CMP   $61A3,X    ;Is the character in the accumulator one of the
                        ;Control characters?
614D: BEQ   $6183      ;If it is, then branch
614F: DEX                   ;If not, then move to the next control character
6150: BNE   $614A      ;Continue until the whole table has been checked
6152: LDX   $79         ;Get the output flag
6154: CPX   #$08       ;Check to see if the text 'CHR$ (' has just been
                        ;Outputted
6156: BCC   $615F      ;If we are at first character of the definition,
                        ;Then branch
6158: BNE   $6164      ;If we are outputting normal text, then branch

```

```

615A: LDA  #'+'      ;Get the value for a plus sign
615C: JSR  $9269     ;And output it
615F: LDA  #'"'      ;Get the value for a quote
6161: JSR  $9269     ;And output it
6164: PLA          ;Get the character of the KEY definition
6165: JSR  $9269     ;And output it
6168: LDX  #$09      ;Indicate that we are outputting normal text
616A: DEC  $78       ;Decrement the length of the definition by one
616C: BNE  $6141     ;If not done, then branch to continue output
616E: CPX  #$09      ;Check to see if we have been outputting text
                        ;Within quotes or 'CHR$'
6170: BCC  $6177     ;If we have outputted '+CHR$ (' , then do not
                        ;Output a quote
6172: LDA  #'"'      ;Print a quote mark to the
6174: JSR  $9269     ;Current output device (screen)
6177: LDA  #141      ;Get the value for a shifted return
6179: JSR  $9269     ;And output it
617C: LDX  $77       ;Get the KEY number we are working on
617E: CPX  #$08      ;Have we done all eight keys
6180: BNE  $6125     ;If not, then branch to continue with next KEY
6182: RTS          ;Exit this routine

*****
*
* This routine is used by the KEY routine to output
* the text '+CHR$(' or 'CHR$(' . If the X register
* contains a seven, the text '+CHR$(' will be
* outputted. However, if the X register contains an
* eight, the text 'CHR$(' will be outputted. On
* exiting this routine, the X register will contain an
* eight to inform the KEY routine that the text
* 'CHR$(' has just been outputted.
*
*****

6183: LDX  $79       ;Get the output flag
6185: LDA  $619A,X   ;Get the character of the '+ CHR$ ('
6188: JSR  $9269     ;And output it
618B: DEX          ;Move to the next character
618C: CPX  #$03      ;Continue until
618E: BCS  $6185     ;The proper amount of characters are outputted
6190: PLA          ;Get the character string value off of the stack
6191: JSR  $A830     ;Output its ASCII equivalent
6194: LDA  #' ) '    ;Value for a closing parenthesis
6196: JSR  $9269     ;Output it
6199: LDX  #$08      ;
619B: BNE  $616A     ;

```

```

619D: TXT  '($RHC+' ;Text for '+CHR$('
61A3: TXT  '''      ;Quote mark
61A4: .BYTE $0D      ;Carriage return
61A5: .BYTE $8D      ;Shifted return
61A6: TXT  '''      ;Quote mark
61A7: .BYTE $1B      ;Escape

```

```

*****
*
*
*          BASIC PAINT COMMAND
*
* Command syntax:  PAINT [color source],x,y[,mode]
*
* NOTE:  color source =  0 (bit map foreground)
*                   1 (bit map background)
*                   2 (multicolor 1)
*                   3 (multicolor 2)
*
*           x,y = the coordinate to start
*                PAINTing at
*
*           mode =  0 (this mode will paint an
*                   area that is defined by
*                   the color source specified)
*                   1 (this mode will paint an
*                   area that is defined by any
*                   non-background source)
*
* This command allows you to fill an area with color
* around the specified starting coordinate.
*
*****

```

```

61A8: JSR  $9E2F      ;Get color source selected and save at $83
61AB: LDX  #$04      ;Set the index to the starting coordinates
61AD: JSR  $9E52      ;Get the X and Y coordinates and save them
61B0: JSR  $9DF2      ;Copy coordinates from locations $1135 - $1138
                    ;To $1131 - $1134
61B3: JSR  $9E1C      ;Check for mode flag and put into the X register
61B6: CPX  #$02      ;If the mode was less than 2,
61B8: BCC  $61BD      ;Then branch
61BA: JMP  $7D28      ;Else, generate an 'ILLEGAL QUANTITY' error
61BD: TXA                    ;Move the flag into the Accumulator
61BE: LSR                    ;Divide it by 2
61BF: ROR                    ;And shift the flag into bit 7
61C0: STA  $8B      ;Save the mode flag
61C2: BPL  $61C8      ;If the mode was zero, then branch

```

```

61C4: LDA    $83           ;Get the color source that was selected
61C6: BEQ    $61CF        ;If the color source was zero, then exit
61C8: JSR    $9C49        ;Calculate the bit map address
61CB: BCS    $61CF        ;If the coordinates are out-of-range, then exit
61CD: BNE    $61D0        ;If the color source at the location is not the
                        ;One selected, then branch
61CF: RTS                    ;Else exit this routine
61D0: JSR    $92EA        ;Perform a Garbage collection
61D3: STA    $FF03        ;Enable BASIC BANK 14
61D6: LDA    $33           ;Save the LSB
61D8: STA    $24           ;Of STREND (STRing END address)
61DA: LDA    $34           ;Save the MSB
61DC: STA    $25           ;Of STREND
61DE: SEC                    ;Set the carry for subtraction
61DF: LDA    $35           ;Get the LSB of FRETOP, start address of strings
61E1: SBC    #$03         ;Subtract 3 to point to the first free byte
                        ;In string storage
61E3: STA    $1B           ;Save the result
61E5: LDA    $36           ;Get the MSB of FRETOP
61E7: SBC    #0            ;Subtract the carry from it
61E9: STA    $1C           ;And save it
61EB: LDX    #0            ;Clear out an index
61ED: STX    $63           ;And
61EF: STX    $64           ;Two flags
61F1: LDX    $1133        ;Get the LSB of the Y coordinate
61F4: BNE    $61F9        ;If it is not equal to zero, then branch
61F6: DEC    $1134        ;Decrement the MSB of the Y coordinate by one
61F9: DEC    $1133        ;Decrement the LSB of the Y coordinate by one
61FC: JSR    $9C49        ;Calculate the new screen address
61FF: BCS    $6203        ;If Y coordinate is out-of-range, then branch
6201: BNE    $61F1        ;If the color at this point is not the same as
                        ;The one that was selected, then branch
6203: INC    $1133        ;Increment the LSB of the Y coordiante by one
6206: BNE    $620B        ;If it is not equal to zero, then branch
6208: INC    $1134        ;If it is equal to zero, then increment the MSB
                        ;Of the Y coordinate by one
620B: JSR    $9C19        ;Set the color and pixel at these coordiantes
620E: LDX    $1131        ;Get the LSB of the X coordinate
6211: BNE    $6216        ;If it is not equal to zero, then branch
6213: DEC    $1132        ;Decrement the MSB of the X coordinate by one
6216: DEC    $1131        ;Decrement the LSB of the X coordinate by one
6219: LDA    $63           ;Get the first coordinate flag
621B: JSR    $627C        ;Calculate the new screen address and save the
                        ;The coordinates if needed
621E: STA    $63           ;Save the coordinate flag back
                        ;($80 = saved, $00 = not saved)
6220: CLC                    ;Clear the carry for addition

```

---

```

6221: LDA  $1131      ;Get the LSB of the X coordinate
6224: ADC  #$02       ;And increase it by 2
6226: STA  $1131      ;Save it back
6229: BCC  $622E      ;If there was no overflow, then branch
622B: INC  $1132      ;If there was an overflow, then increment the
                        ;MSB by 1
622E: LDA  $64        ;Get the second coordinate flag
6230: JSR  $627C      ;Calculate the new screen address and save
                        ;The coordinates if needed
6233: STA  $64        ;Save the coordinate flag back
6235: LDX  $1131      ;Get the LSB of the X coordinate
6238: BNE  $623D      ;If it is not equal to zero, then branch
623A: DEC  $1132      ;Decrement the MSB of the X coordinate by one
623D: DEC  $1131      ;Decrement the LSB of the X coordinate by one
6240: INC  $1133      ;Increment the LSB of the Y coordinate by one
6243: BNE  $6248      ;If it is not equal to zero, then branch
6245: INC  $1134      ;Increment the MSB of the Y coordinate by one
6248: JSR  $9C49      ;Calculate the new screen address
624B: BCS  $624F      ;If Y coordinate is out-of-range, then branch
624D: BNE  $620B      ;If the color at this point is not the same as
                        ;The one that was selected, then branch
624F: LDX  #$03       ;Set up an index for 4 bytes - 1
6251: LDY  #0         ;Clear the Y index
6253: LDA  $25        ;Get the MSB of the current storage address
6255: CMP  $34        ;Compare it with the MSB of the start
                        ;Of the storage area
6257: BNE  $625F      ;If it is not equal to it, then branch
6259: LDA  $24        ;Get the LSB of the current storage address
625B: CMP  $33        ;Compare it with the LSB of the start
                        ;Of the storage area
625D: BEQ  $6279      ;If they are equal, then there are no
                        ;Coordinates to read, so branch
625F: LDA  $24        ;Get the LSB of the current storage address
6261: BNE  $6265      ;If it is not equal to zero, then branch
6263: DEC  $25        ;Decrement the MSB by one
6265: DEC  $24        ;Decrement the LSB by one
6267: JSR  $03B7      ;Get a byte of the coordinates
626A: STA  $FF03      ;Enable BASIC BANK 14
626D: STA  $1131,X    ;Save them as the current coordinates
6270: DEX             ;Move to the next byte of the coordinates
6271: BPL  $625F      ;Continue until four bytes have been transferred
6273: JSR  $4BB5      ;Check the STOP key and abort if it is depressed
6276: JMP  $61EB      ;Continue filling
6279: JMP  $9DF2      ;Copy coordinates from locations $1135 - $1138
                        ;To $1131 - $1134
627C: PHA             ;Save the coordinate saved flag
627D: JSR  $9C49      ;Calculate the new screen address

```

```

6280: BCS $629A ;If coordinates are out-of-range, then branch
6282: BEQ $629A ;If the color at this point is the same as the
;Selected color source, then branch
6284: PLA ;Get the flag back
6285: BNE $629D ;If the coordinates were saved, then exit
6287: TAX ;Clear out
6288: TAY ;The X and Y registers
6289: LDA $25 ;Get the MSB of the present storage address
628B: CMP $1C ;Is it the same as the Maximum size MSB?
628D: BCC $629E ;If it is less than, then branch
628F: BNE $6297 ;If it is greater than, then generate an
;'OUT OF MEMORY' error
6291: LDA $24 ;Get the LSB of the present storage address
6293: CMP $1B ;Is it the same as the Maximum size LSB?
6295: BCC $629E ;If it is less than, then branch
6297: JMP $4D3A ;Generate an 'OUT OF MEMORY' error
629A: PLA ;Get the flag back
629B: LDA #$00 ;Flag for coordinates not saved
629D: RTS ;Exit this routine
629E: LDA $1131,X ;Get the current coordinate
62A1: STA $FF04 ;Enable BASIC BANK 14 with RAM BANK 1 enabled
62A4: STA ($24),Y ;Save the coordinate temporarily
62A6: STA $FF03 ;Enable BASIC BANK 14
62A9: INC $24 ;Increment the LSB of the storage address
62AB: BNE $62AF ;If there was no overflow, then branch
62AD: INC $25 ;If there was overflow, then increment MSB by 1
62AF: INX ;Increment to the next byte of the coordinates
62B0: CPX #$04 ;Continue until
62B2: BNE $629E ;4 bytes have been saved
62B4: LDA #$80 ;Flag to indicate that the coordinates are saved
62B6: RTS ;Exit this routine

```

```

*****
*
*
*          BASIC BOX COMMAND
*
* Command syntax:  BOX [color source],X1,Y1 coords
*                  [,X2,Y2 coords][angle][paint]
*
* NOTE: color source = 0 (background color)
*                  1 (foreground color)
*                  2 (multicolor 1)
*                  3 (multicolor 2)
*
*          X1,Y1 coords = the top left corner coordinates
*
*

```

```

*           X2,Y2 coords = the bottom right corner that is *
*                   opposite the X1,Y1 coordinates *
*
*           angle = the angle in degrees *
*
*           paint = 0 (do not paint the shape) *
*                   1 (paint the shape) *
*
* This command allows you to draw a BOX on the screen *
* as specified by the parameters that follow the BOX *
* command. *
*
*****

```

```

62B7: JSR   $9E2F       ;Get color source selected and store at $83
62BA: LDX   #$1F       ;Get the top right X, Y coordinates
62BC: JSR   $9E6D       ;And store them at locations $1150 - $1153
62BF: LDX   #$2B       ;Get the bottom X, Y coordinates
62C1: JSR   $9E52       ;And store them at locations $115C - $115F
62C4: JSR   $9E06       ;Get the rotation angle into the Y register and
                        ;The accumulator (LSB, MSB)
62C7: STY   $1154       ;Save the LSB
62CA: STA   $1155       ;And the MSB of the rotation angle
62CD: JSR   $9E1C       ;Get the fill flag, if any
62D0: CPX   #$02       ;If the value is
62D2: BCC   $62D7       ;Greater than 1,
62D4: JMP   $7D28       ;Then generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
* This is the entry point for machine language program-
* mers who wish to use the BOX drawing routines. You
* must enter this routine with the top right X and Y
* coordinates in locations $1150 - $1153 and the bottom
* X and Y coordinates in locations $115C - $115F. Also,
* the rotation angle must be in locations $1154 - $1155
* and either a non zero value must be in the X register
* to select fill or a zero for no fill.
*
*****

```

```

62D7: STX   $116C       ;Save the fill flag
62DA: TXA                   ;Move the flag to the accumulator
62DB: PHA                   ;And save it onto the stack
62DC: JSR   $6389
62DF: PLA                   ;Get the Fill Flag back
62E0: BNE   $62FE       ;If the Fill was selected, then branch

```

---

```
62E2: BEQ    $62E7    ;If the Fill was not selected, then branch
62E4: JSR    $640B
62E7: JSR    $9B30    ;Draw a side of the box
62EA: LDA    $114E    ;Check if we are done
62ED: BNE    $62E4    ;Continue until 4 sides are drawn
62EF: LDX    #$04     ;Index to number of bytes to transfer
62F1: LDA    $115B,X  ;Move the bottom right X and Y
62F4: STA    $1130,X  ;Coordinates from locations $115C - $115F
62F7: DEX
62F8: BNE    $62F1    ;Continue until 4 bytes have been transfered
62FA: STX    $116C    ;Clear the fill flag
62FD: RTS
62FE: LDX    #0
6300: LDA    $1149
6303: LSR
6304: BCC    $6308
6306: LDX    #$02
6308: LDA    $1160,X
630B: STA    $115A
630E: LDA    $1161,X
6311: STA    $115B
6314: LDA    #0
6316: LDX    #$03
6318: STA    $1156,X
631B: DEX
631C: BPL    $6318
631E: LDX    #$07
6320: LDA    $1131,X
6323: PHA
6324: DEX
6325: BPL    $6320
6327: JSR    $9B30
632A: LDX    #0
632C: PLA
632D: STA    $1131,X
6330: INX
6331: CPX    #$08
6333: BNE    $632C
6335: LDA    $115A
6338: BNE    $633F
633A: DEC    $115B
633D: BMI    $62EF
633F: DEC    $115A
6342: LDX    #$25
6344: LDY    #$1B
6346: LDA    $1149
6349: LSR
```



---

```

634A: BCC    $634E
634C: LDY    #$19
634E: LDA    #0
6350: LSR
6351: PHA
6352: JSR    $9D6D
6355: STA    $1131,X
6358: TYA
6359: STA    $1132,X
635C: PLA
635D: BCC    $6361
635F: ORA    #A0
6361: INX
6362: INX
6363: LDY    #$19
6365: LSR    $1149
6368: BCC    $636C
636A: LDY    #$1B
636C: ROL    $1149
636F: CPX    #$27
6371: BEQ    $6350
6373: LDX    #$06
6375: ASL
6376: BEQ    $6335
6378: BCC    $6382
637A: INC    $1131,X
637D: BNE    $6382
637F: INC    $1132,X
6382: ASL
6383: DEX
6384: DEX
6385: BPL    $6378
6387: BMI    $631E
6389: LDY    #$23      ;Index to the rotation angle
638B: JSR    $9A74      ;Calculate the proper rotation angle index
638E: LDX    #$1F      ;Index to the starting coordinates
                          ;At locations $1150 - $1153
6390: LDY    #$2B      ;Index to the ending X coordinates
                          ;At locations $115C - $115F
6392: TYA
                          ;Save the ending
6393: PHA
                          ;Coordinate index
6394: JSR    $9D99      ;Calculate difference between starting and
                          ;Ending X coordinate
6397: STA    $1135,X      ;Save the result
639A: STA    $1139,X      ;In locations $1154 - $1157
639D: STA    $1141,X      ;And $1158 - $115B
63A0: TYA

```

---

```
63A1: STA $1136,X ;And also in locations $1160 - $1163
63A4: STA $113A,X
63A7: STA $1142,X
63AA: PLA ;Get the index to the ending X coordinate
63AB: TAY ;Move it to the Y register
63AC: JSR $9D6D ;Add starting coordinate to ending coordinate
63AF: STA $1131,X ;Save the result
63B2: TYA ;In locations $1150, $1151
63B3: STA $1132,X
63B6: LDY #$2D ;Index to the starting Y coordinate
;At locations $115E, $115F
63B8: INX ;Increment the index
63B9: INX ;To the Y coordinate
63BA: CPX #$21 ;Branch to do the Y
63BC: BEQ $6392 ;Coordinate
63BE: LDA #$90
63C0: JSR $9AF3
63C3: LDA $1149 ;Get the value - (Rotation angle/90)
63C6: AND #$03 ;Drop the unwanted bits
63C8: STA $1149 ;And save the result
63CB: TAX ;Use the value as an index
63CC: LDA $63ED,X
63CF: JSR $640B
63D2: JSR $9DF2
63D5: LDA $114E
63D8: JSR $640B
63DB: LDX $1149
63DE: LDA $63ED,X
63E1: AND #$F0
63E3: STA $114F
63E6: LDA $63F1,X
63E9: STA $114E
63EC: RTS
63ED: .BYTE $BE,$E4,$41
63F0: .BYTE $1B
63F1: .BYTE $41,$1B,$BE
63F4: .BYTE $E4
```

```
*****
*
* Below are the software programmers' names.
*
*****
```

```
63F5: TXT 'fred b'
63FB: .BYTE $0D ;<cr>
63FC: TXT 'terry r'
6403: .BYTE $0D ;<cr>
6404: TXT 'mike i'
640A: .BYTE $0D ;<cr>
640B: JSR $6767
640E: LDX #$04
6410: LDA $1132,X
6413: ASL
6414: ROR $1132,X
6417: ROR $1131,X
641A: BCC $6424
641C: INC $1131,X
641F: BNE $6424
6421: INC $1132,X
6424: INX
6425: INX
6426: CPX #$06
6428: BEQ $6410
642A: RTS
```

```
*****
*
* BASIC SSHAPE COMMAND
*
* Command syntax: SSHAPE string, X1,Y1 coords
*                  [,X2,Y2 coords]
*
* NOTE:          string = the string variable to which you*
*                  wish to save the area of screen *
*
*
* X1,Y1 coords = this is the starting corner
*                  coordinates for the save
*
* X2,Y2 coords = this is the coordinates of the
*                  corner that is opposite the
*                  the starting corner
*                  (X1,Y1 coords)
*
*****
```

```

* This command allows you to save an area of the      *
* the screen to a BASIC string variable.             *
*                                                     *
*****

```

```

642B: JSR    $A074      ;Check if the Graphics screen is allocated.
                        ;If the Graphics screen is not allocated,
                        ;Then generate an error
642E: JSR    $7AAF      ;Check to see if the string exists; if not,
                        ;Create it and return its address in the
                        ;Accumulator and Y register
6431: STA    $FF03      ;Enable BASIC BANK 14
6434: STA    $115F      ;Save the address of
6437: STY    $1160      ;The string's descriptor
643A: BIT    $0F        ;Is the variable a string?
643C: BMI    $6441      ;If it is a string, then branch
643E: JMP    $77E7      ;If not, then generate a 'TYPE MISMATCH' error
6441: LDX    #$28        ;Get the starting X and Y coordinates
6443: JSR    $9E6D      ;And store them at locations $1159 - $115C
6446: LDX    #$04        ;Get the opposite X and Y coordinates
6448: JSR    $9E52      ;And store them at locations $1135 - $1138
644B: LDX    #$2A        ;Index to Y1
644D: LDY    #$06        ;Index to Y2
644F: LDA    #$02        ;Index to the storage area
6451: STA    $8E         ;For Y2 at locations $1133 - $1134
6453: JSR    $9D99      ;Calculate the difference between Y1 and Y2
6456: TAX                    ;Save the LSB of the result in the X register
6457: TYA                    ;Save the MSB of the result
6458: PHA                    ;Onto the stack
6459: LDY    $8E         ;Get the index to Y2
645B: JSR    $9DF9      ;Move Y2 from $1137-$1138 to $1133-$1134
645E: BCC    $646C      ;If the starting coordinate was less than the
                        ;Ending coordinate, then branch
6460: LDA    $1159,Y     ;Else transfer the starting
6463: STA    $1131,Y     ;Coordinate (X1 or Y1) from
6466: LDA    $115A,Y     ;Locations $1159 - $115A to $1131 - $1134
6469: STA    $1132,Y
646C: TXA                    ;Move LSB of the difference between coordinates
646D: STA    $1159,Y     ;Into the accumulator and save it
6470: STA    $03DB,Y
6473: PLA                    ;Save the MSB of the difference
6474: STA    $115A,Y
6477: STA    $03DC,Y
647A: LDX    #$28        ;Index to X1
647C: LDY    #$04        ;Index to X2
647E: DEC    $8E         ;Decrement the index by 2
6480: DEC    $8E         ;To point to X2

```

---

```

6482: BEQ    $6453    ;Branch to do X1 and X2
6484: LDY    #$FF      ;Set the string length to
6486: STY    $1155     ;The maximum length of 255
6489: LDA    $1131     ;Get the X1 value
648C: STA    $115D     ;And transfer it
648F: LDA    $1132     ;From locations $1131 - $1132
6492: STA    $115E     ;To $115D - $115F
6495: TYA                ;Move the length of 255 to the accumulator
6496: JSR    $8690     ;Allocate 255 bytes in string memory for data
6499: STA    $FF03     ;Enable BASIC BANK 14
649C: JSR    $9CE3     ;Calculate address of the point on HIRES-screen
649F: LDA    ($8C),Y   ;If the coordinates are
64A1: BCC    $64B1     ;Within range, then branch
64A3: LDA    $1131     ;Get the LSB of X1
64A6: BIT    $D8      ;Check to see if we are in the multicolor mode
64A8: BPL    $64AC     ;If not, then branch
64AA: SEC                ;Set the carry flag
64AB: ROL                ;And shift it into bit 0
64AC: AND    #$07     ;Calculate the bit position
64AE: TAX                ;And move the result to the X register
64AF: LDA    #0
64B1: BIT    $D8      ;Check to see if we are in the multicolor mode
64B3: BPL    $64B6     ;If we are not, then branch
64B5: DEX                ;Subtract one from the bit position
64B6: STX    $1161     ;Save the bit position
64B9: ASL                ;Shift the bit
64BA: DEX                ;To the proper
64BB: BPL    $64B9     ;Position in the byte
64BD: ROR                ;
64BE: STA    $8E      ;Save the result
64C0: LDA    #$08     ;Value to move to next byte
64C2: BIT    $D8      ;Check to see if we are in the multicolor mode
64C4: BPL    $64C7     ;If we are not, then branch
64C6: LSR                ;Divide by 2
64C7: CLC                ;Clear the carry for addition
64C8: ADC    $1131     ;Add value to X1's LSB to move to the next byte
                        ;Across
64CB: STA    $1131     ;Across and save it
64CE: BCC    $64D3     ;If there was no overflow, then branch
64D0: INC    $1132     ;Increment the MSB by 1
64D3: JSR    $9CE3     ;Calculate the HIRES screen address
64D6: LDA    #0        ;Default value
64D8: BCS    $64DC     ;If coordinates are out of range, then branch
64DA: LDA    ($8C),Y   ;Get the current byte value there
64DC: STA    $8F      ;And save it
64DE: LDX    $1161     ;Get the bit position
64E1: LSR                ;Shift the bit

```

```

64E2: INX                ;To its proper location in the byte
64E3: CPX    #$08
64E5: BNE    $64E1
64E7: ORA    $8E        ;Add the bit to be set
64E9: INC    $1155      ;Increment the string length
64EC: LDY    $1155      ;And use it as an index
64EF: CPY    #$FC      ;If the length is less than 255,
64F1: BCC    $64F6      ;Then ok
64F3: JMP    $A5ED      ;Else generate a 'STRING TOO LONG' error
64F6: STA    $FF04      ;Enable BANK 14 with RAM BANK 1
64F9: STA    ($64),Y    ;Save into the string
64FB: STA    $FF03      ;Enable BASIC BANK 14
64FE: LDX    $1161      ;Get the bit position
6501: LDA    $1159      ;Get the LSB of the X coordinate
6504: SEC
6505: BIT    $D8        ;Check to see if we are in the multicolor mode
6507: BPL    $650C      ;If we are in standard bit map mode, then branch
6509: SBC    #$04      ;If we are in multicolor mode, the subtract 4
650B: .BYTE  $2C        ;Mask
650C: SBC    #$08      ;If we are in standard bit map, then subtract 8
650E: STA    $1159      ;Save the result
6511: LDA    $8F        ;Get the current byte
6513: BCS    $64B9      ;If there was no underflow, then branch
6515: DEC    $115A      ;Else decrement MSB by 1 to move to next byte
6518: BPL    $64B9      ;Continue until the row is done
651A: LDX    $115B      ;Decrement the Y size
651D: BNE    $6567      ;By one
651F: DEC    $115C
6522: BPL    $6567      ;Continue until the columns are completed
6524: BIT    $D8        ;Check to see if we are in the multicolor mode
6526: BPL    $652E      ;If we are in standard bit map mode, then branch
6528: ASL    $03DB      ;If we are in multicolor mode, then divide the
652B: ROL    $03DC      ;X size in half
652E: LDX    #0         ;Starting index of zero
6530: LDA    $03DB,X    ;Get a byte of the
6533: INY
6534: STA    $FF04      ;Enable BANK 14 with RAM BANK 1
6537: STA    ($64),Y    ;Save the X and Y size after the shape
6539: STA    $FF03      ;Enable BASIC BANK 14
653C: INX
653D: CPX    #$04      ;Continue until 4 bytes
653F: BNE    $6530      ;Are stored at the end of the string
6541: INY
6542: STY    $03DB      ;And save it
6545: LDA    $64        ;Get the LSB of the string address
6547: STA    $03DC      ;And save it
654A: LDA    $65        ;Get the MSB of the string address

```

```

654C: STA  $03DD      ;And save it
654F: LDA  #$DB      ;Value for LSB of address of descriptor ($03DB)
6551: STA  $66        ;Save it
6553: LDA  #$03      ;MSB of descriptor address ($03DB)
6555: STA  $67        ;Save it
6557: LDA  $115F     ;Move the string's descriptor
655A: STA  $4B        ;Address from
655C: LDA  $1160     ;STRADR
655F: STA  $4C        ;Into LSTPNT
6561: JSR  $5494     ;Create the string
6564: JMP  $9DF2     ;Make X and Y destination into X and Y position
6567: DEC  $115B     ;Decrement the length by 1
656A: INC  $1133     ;Increment the
656D: BNE  $6572     ;Current Y coordinate
656F: INC  $1134     ;By 1 to the next row
6572: LDA  $115D     ;Return the starting
6575: STA  $1131     ;X1 coordinate
6578: LDA  $115E     ;To locations $1131 - $1132
657B: STA  $1132
657E: LDA  $03DB     ;Get the
6581: STA  $1159     ;X size
6584: LDA  $03DC     ;And
6587: STA  $115A     ;Save it
658A: JMP  $649C     ;Continue with the new row

```

```

*****
*
*          BASIC GSHAPE COMMAND
*
* Command syntax:  GSHAPE string, [,X,Y coords][mode]
*
* NOTE:          string = the string variable where the
*                  shape to be drawn is stored
*
*                X,Y coords = the top left corner coordinates
*                  where the shape is to be drawn
*
*                mode = 0 (place shape as is)
*                      1 (invert the shape)
*                      2 (OR the shape with the area)
*                      3 (AND the shape with the area)
*                      4 (XOR the shape with the area)
*
* This command allows you to take a shape stored
* in a BASIC string variable and draw it on the screen.*
*
*****

```

---

```

658D: JSR   $A074      ;Check if the Graphics screen is allocated.
                        ;If the Graphics screen is not allocated, then
                        ;Generate an error
6590: JSR   $877B      ;Place the string address in locations $24, $25
                        ;And the length in the accumulator
6593: STA   $FF03      ;Enable BANK 14
6596: STA   $1153      ;Place the length of the string in STRSZ
6599: STX   $26        ;Save the address of the string
659B: STY   $27        ;In INDEX2
659D: LDX   #$04       ;Get the top left X and Y coordinates
659F: JSR   $9E52      ;And store them at locations $1135 - $1138
65A2: JSR   $9E1C      ;Get the optional mode flag in the X register
65A5: CPX   #$05       ;Check to ensure the mode value
65A7: BCC   $65AC      ;Is less than 5
65A9: JMP   $7D28      ;If it is not, then generate an
                        ;'ILLEGAL QUANTITY' Error
65AC: STX   $1154      ;Save the mode value in GETTYP
65AF: LDX   #$03       ;Index to number of bytes in string's descriptor
65B1: LDY   $1153      ;Get the length of the string
65B4: CPY   #$05       ;If the length of the string
65B6: BCS   $65B9      ;Is less than 5,
65B8: RTS                ;Then exit
65B9: DEY                ;Decrement the length by one
65BA: LDA   #$26       ;Pointer to the string's descriptor
65BC: JSR   $03AB      ;Get a byte of the string's descriptor
65BF: STA   $FF03      ;Enable BASIC BANK 14
65C2: STA   $1159,X    ;Save the descriptor in locations $1159 - $115B
65C5: DEX                ;Continue until 3 bytes
65C6: BPL   $65B9      ;Have been transferred
65C8: STX   $1155      ;Initialize a counter
65CB: JSR   $9DF2      ;Copy the coordinates from locations
                        ;$1135 - $1138 to $1131 - $1134
65CE: LDA   $1159      ;Save the
65D1: STA   $115D      ;Address of
65D4: LDA   $115A      ;The string in RAM BANK 1
65D7: STA   $115E
65DA: LDA   #$08       ;Initialize a bit counter
65DC: STA   $1169      ;To 8 bits in BITCNT
65DF: INC   $1155      ;Increment the counter by 1
65E2: LDY   $1155      ;Get the value as an index
65E5: LDA   #$26       ;Pointer to the string
65E7: JSR   $03AB      ;Get a byte of the string
65EA: STA   $FF03      ;Enable BASIC BANK 14
65ED: STA   $1157      ;Save the byte of the string
65F0: JSR   $9C49      ;Get the corresponding byte from the bit map
65F3: STA   $1156      ;And save it
65F6: ASL   $1157      ;Shift one bit right

```



---

```

65F9: ROL                ;The string value
65FA: DEC    $1169      ;Decrement the bit counter
65FD: BIT    $D8        ;Check to see if we are in multicolor mode
65FF: BPL    $6608      ;If we are in standard bit map mode, the branch
6601: ASL    $1157      ;Else shift the byte
6604: ROL                ;One more bit
6605: DEC    $1169      ;Decrement the bit counter by 1
6608: LDX    $1154      ;Get the mode type
660B: CPX    #$03        ;Is it the mode for 'AND'?
660D: BCC    $661B      ;If it is less than, then branch
660F: BEQ    $6616      ;Yes, branch
6611: EOR    $1156      ;'EOR' the value with memory
6614: BCS    $6627      ;And branch
6616: AND    $1156      ;'AND' the value with memory
6619: BCS    $6627      ;And branch
661B: CPX    #1         ;Should the shape be inverted?
661D: BCC    $6627      ;If not, then branch
661F: BEQ    $6625      ;Yes, invert the byte
6621: ORA    $1156      ;Combine the two values
6624: .BYTE  $2C        ;Mask
6625: EOR    #$FF        ;Invert the value
6627: AND    #$03        ;Drop the unwanted bits
6629: BIT    $D8        ;Check to see if we are in the multicolor mode
662B: BMI    $662F      ;If we are in multicolor mode, then branch
662D: AND    #1         ;If we are in standard bit map, then mask bit 0
662F: STA    $83        ;And use it as the color source
6631: JSR    $9C19      ;Plot the point
6634: INC    $1131      ;Increment to
6637: BNE    $663C      ;The next
6639: INC    $1132      ;X coordinate
663C: SEC                ;Set the carry for subtraction
663D: LDA    $115D      ;Get the LSB of the string address
6640: BIT    $D8        ;Check to see if we are in the multicolor mode
6642: BPL    $6647      ;If we are in standard bit map mode, then branch
6644: SBC    #$02        ;If we are in multicolor mode, then subtract 2
6646: .BYTE  $2C        ;Mask
6647: SBC    #$01        ;If we are in standard bit map, then subtract 1
6649: STA    $115D      ;Save the result
664C: LDA    $115E      ;Subtract the carry, if any
664F: SBC    #0         ;From the MSB of the string address
6651: STA    $115E      ;And save the result
6654: BCS    $6683
6656: LDX    #1
6658: LDA    $1159,X
665B: STA    $115D,X
665E: LDA    $1135,X
6661: STA    $1131,X

```

```

6664: DEX
6665: BPL $6658
6667: INC $1133 ;Increment the Y coordinate by 1
666A: BNE $666F
666C: INC $1134
666F: SEC ;Set the carry for subtraction
6670: LDA $115B
6673: SBC #1
6675: STA $115B
6678: LDA $115C
667B: SBC #0
667D: STA $115C
6680: BCS $668B
6682: RTS
6683: LDA $1169 ;Check if 8 bits have been done
6686: BEQ $668B ;If so, then continue with next byte
6688: JMP $65F0 ;Else continue
668B: JMP $65DA ;Transfer the next byte of the string

```

```

*****
*
*          BASIC CIRCLE COMMAND
*
* Command syntax: CIRCLE [color source],X,Y coords
*                  [,X,Y radii][,sarc][,earc][,angle]
*                  [incr]
*
* NOTE: color source = 0 (background color)
*                  1 (foreground color)
*                  2 (multicolor 1)
*                  3 (multicolor 2)
*
*          X,Y coords = the coordinates of the center
*                  of the circle
*
*          X,Y radii = the X and Y radii
*
*          sarc = the starting arc angle
*
*          earc = the ending arc angle
*
*          angle = the rotation in degrees
*
*          incr = the number of degrees between
*                  each segment
*
*

```

```

* This command allows you to draw circles, ellipses, *
* or arcs depending on the parameter values that were *
* specified after the CIRCLE command. *
* *
*****

```

```

668E: JSR $9E2F ;Get color source and store it at location $83
6691: LDX #$1F ;Get the center X and Y coordinates
6693: JSR $9E52 ;And store them at locations $1150 - $1153
6696: JSR $9E06 ;Get the X radius
6699: STY $1154 ;Save the LSB
669C: STA $1155 ;And the MSB of the X radius
669F: JSR $9E06 ;Get the Y radius
66A2: STY $1156 ;Save the LSB
66A5: STA $1157 ;And the MSB of the Y radius
66A8: PHP ;Save the status register
66A9: LDX #$23 ;Pointer to the X radius
66AB: JSR $9D4A ;Check scaling; adjust coordinates if selected
66AE: PLP ;Restore the status flags
66AF: BCS $66C2 ;If the Y radius was specified, then branch
66B1: LDA $1154 ;If the Y radius was not specified, then
;Make the Y radius
66B4: STA $1156 ;Equal the X radius
66B7: LDA $1155 ;Get the MSB of the X radius
66BA: BIT $D8 ;Check for multicolor mode
66BC: BPL $66C2 ;If we are not in multicolor mode, then branch
66BE: ASL $1156 ;Divide the X radius by 2
66C1: ROL ;To arrive at the correct
66C2: STA $1157 ;Y radius
66C5: JSR $9E06 ;Get the starting arc angle
66C8: STY $115C ;And save
66CB: STA $115D ;It
66CE: JSR $9E06 ;Get the ending arc angle
66D1: STY $115E ;And save
66D4: STA $115F ;It
66D7: JSR $9E06 ;Get the rotation angle
66DA: STA $77 ;Save the MSB
66DC: TYA ;Move the LSB to the accumulator
66DD: LDY $77 ;And the MSB to the Y register
66DF: JSR $9A77
66E2: LDX #$2D ;Index to ending arc angle
66E4: LDY #$2B ;Index to starting arc angle
66E6: JSR $9D7C ;Compute the difference between the starting and
;Ending arc angles
66E9: BCC $66F9 ;If Ending arc angle was greater, then branch
66EB: LDA #$68 ;Else add 360
66ED: LDY #1 ;To the ending

```

```

66EF: JSR   $9D70      ;Arc angle
66F2: STA   $1131,X    ;And save
66F5: TYA                      ;The result
66F6: STA   $1132,X
66F9: LDX   #$03      ;Move the X and Y
66FB: LDA   $1154,X    ;Radius from locations $1154 - $1157
66FE: STA   $1158,X    ;To $1158 - $115B
6701: DEX
6702: BPL   $66FB
6704: LDA   #$90
6706: JSR   $9AF3
6709: LDX   #$07      ;Index to the number of bytes to transfer
670B: LDA   $1154,X    ;Move all the coordinates
670E: STA   $1160,X    ;From locations $1154 - $115B
6711: DEX              ;To $1160 - $1167
6712: BPL   $670B
6714: JSR   $6750
6717: JSR   $9DF2
671A: LDX   #$02      ;Set default number of degrees per segment to 2
671C: JSR   $9E1E      ;Get the number of degrees per segment
671F: TXA                      ;Move the value to the accumulator
6720: BNE   $6725      ;If the value is equal to zero,
6722: JMP   $7D28      ;Then generate an 'ILLEGAL QUANTITY' error
6725: STX   $1220      ;Save the number of degrees per segment
6728: CLC                      ;Clear the carry for addition
6729: LDA   $1220      ;Add the number of
672C: ADC   $115C      ;Degrees per segment to
672F: STA   $115C      ;The starting arc angle
6732: BCC   $6737      ;If there was no overflow, then branch
6734: INC   $115D      ;If there was overflow, increment MSB by 1
6737: LDX   #$2D      ;Index to ending arc angle
6739: LDY   #$2B      ;Index to starting arc angle plus
                          ;Number of degrees per segment
673B: JSR   $9D7C      ;Compute difference between starting and
                          ;Ending arc angles
673E: BCS   $6748      ;If starting arc angle was greater, then branch
6740: JSR   $6750
6743: JSR   $9B30
6746: BCC   $6729
6748: LDY   #$2D
674A: JSR   $6752
674D: JMP   $9B30
6750: LDY   #$2B
6752: JSR   $9A74
6755: LDX   #$07
6757: LDA   $1160,X
675A: STA   $1154,X

```

```

675D: DEX
675E: BPL $6757
6760: LDA #$50
6762: JSR $9AF3
6765: LDA #$10
6767: STA $114E
676A: LDY #$1F
676C: LDX #$23
676E: ASL $114F
6771: ROL $114E
6774: JSR $9D6B
6777: INX
6778: INX
6779: ASL $114F
677C: ROL $114E
677F: JSR $9D67
6782: PHA
6783: TYA
6784: PHA
6785: LDY #$21
6787: INX
6788: INX
6789: CPX #$27
678B: BEQ $676E
678D: LDX #$03
678F: PLA
6790: STA $1135,X
6793: DEX
6794: BPL $678F
6796: RTS
    
```

```

*****
*
*          BASIC DRAW COMMAND
*
* Command syntax: DRAW [color source],[X1,Y1 coords]
*                  [TO X2,Y2 coords]...
*
* NOTE: color source = 0 (bit map background)
*                  1 (bit map foreground)
*                  2 (multicolor 1)
*                  3 (multicolor 2)
*
*          X1,Y1 coords = the starting coordinates
*
*          X2,Y2 coords = the ending coordinates
*
*
    
```

```

* This command allows you to DRAW dots, lines, and *
* shapes at the positions specified by the coordinates.*
* *
*****

```

```

6797: JSR  $A074      ;Check if Graphics screen is allocated. If the
                    ;Graphics screen was not allocated, then
                    ;Generate an error
679A: LDX  #1         ;Set the default color source
679C: STX  $83        ;To 1
679E: JSR  $0386     ;Get the character after the DRAW command
67A1: CMP  #$A4       ;Is it the token for 'TO'?
67A3: BEQ  $67B0     ;If it is, then branch
67A5: JSR  $9E32     ;Get color source and store it at location $83
67A8: JSR  $0386     ;Get the next character
67AB: BNE  $67B0     ;If there are more parameters, then branch
67AD: JMP  $9BFB     ;Else plot the pixel and exit
67B0: JSR  $0386     ;Check if the next character
67B3: CMP  #$2C      ;Is a comma
67B5: BEQ  $67BC     ;If the character is a comma, then branch
67B7: CMP  #$A4       ;Is it the token for 'TO'?
67B9: BEQ  $67BC     ;If it is, then branch
67BB: RTS             ;If it is not, then exit
67BC: PHA             ;Save the character temporarily onto the stack
67BD: JSR  $0380     ;Get the next character
67C0: LDX  #$04       ;Obtain the ending coordinates
67C2: JSR  $9E70     ;And store them at locations $1135 - $1138
67C5: PLA             ;Get the character back
67C6: BPL  $67CE     ;If it is not the token for 'TO', then branch
67C8: JSR  $9B30     ;Draw the Line
67CB: JMP  $67B0     ;Continue, and check next character
67CE: JSR  $9DF2     ;Move the coordinates from locations
                    ;$1135 - $1138 to $1131 - $1134
67D1: JSR  $9BFB     ;Plot the point
67D4: JMP  $67B0     ;And jump to process the next character

```

```

*****
* *
*           BASIC CHAR COMMAND           *
* *
* Command syntax: CHAR [color source],C,R [,string] *
*                [,rev. flag]           *
* *
* NOTE: color source = 0 (background color) *
*                1 (foreground color)    *
* *
*                C = column to start printing at *

```

```

*           R = row to start printing at           *
*
*           string = string of characters to be     *
*                   printed                         *
*           rev. flag = reverse field flag         *
*
*           where: 0 = off                          *
*                   1 = on                         *
*
* This command allows you to display characters to the *
* screen at the point specified by the 'C' and 'R'   *
* parameters. This command also has the option to   *
* allow you to print the characters in the reverse  *
* field. If the 'rev. flag' is a one, then the     *
* reverse field is enabled.                         *
*
*****

```

```

67D7: JSR   $9E32      ;Get the color source, if any, and save it
                        ;At location $83
67DA: LDX   #$29      ;Load the X register with the maximum number
                        ;Of columns plus one
67DC: LDY   #$1A      ;Load the Y register with the maximum number
                        ;Of rows plus one
67DE: LDA   $D8       ;Check if we are in the text or graphics mode
67E0: BNE   $67E7     ;If we are in the graphics mode, then branch
67E2: JSR   $FFED     ;Get the size of the current window
67E5: INX                   ;Add one to the column value
67E6: INY                   ;Add one to the row value
67E7: STX   $115E     ;Save the maximum column value plus one
67EA: STY   $115F     ;Save the maximum row value plus one
67ED: JSR   $8809     ;Get the column into the X register
67F0: CPX   $115E     ;If it is greater than
67F3: BCS   $6800     ;The maximum number of columns, then generate
                        ;An 'ILLEGAL QUANTITY' error
67F5: STX   $115E     ;Save the column to start outputting at
67F8: JSR   $8809     ;Get the row value into the X register
67FB: CPX   $115F     ;If the row value specified
67FE: BCC   $6803     ;Is within the proper range, then branch
6800: JMP   $7D28     ;Generate an 'ILLEGAL QUANTITY' error
6803: STX   $115F     ;Save the row to start outputting at
6806: JSR   $0386     ;See if there is anything after the
6809: BNE   $680F     ;Row coordinate, and if so, then branch
680B: LDA   #0        ;If not, then set the length
680D: BEQ   $6815     ;Of the string to zero, and branch
680F: JSR   $795C     ;Check for a comma, and generate a 'SYNTAX'
                        ;error if not found

```

---

```

6812: JSR   $877B      ;Evaluate the string of characters to output
6815: STA   $FF03      ;Enable BANK 14
6818: STA   $116E      ;Save the length of the string
681B: TYA                ;Save the MSB of the
681C: PHA                ;Address of the string onto the stack
681D: TXA                ;Save the LSB of the
681E: PHA                ;Address of the string onto the stack
681F: JSR   $9E1C      ;Get REVERSE flag, if any, into the X register
6822: TXA                ;Move the flag into the accumulator
6823: ROR                ;If the flag = 1, then carry flag = 1
6824: ROR   $113D      ;Move the carry into bit 7
6827: PLA                ;Get the LSB of the string's address
6828: STA   $24         ;And save it
682A: PLA                ;Get the MSB of the string's address
682B: STA   $25         ;And save it
682D: LDA   $D8         ;Check to see if we are in the graphics mode
682F: BNE   $6863      ;If we are in the graphics mode, then branch
6831: LDX   $115F      ;Get the column
6834: LDY   $115E      ;And the row to start outputting at
6837: CLC                ;Clear the carry for addition
6838: JSR   $928D      ;PLOT - position the cursor
683B: LDY   #0          ;Value to check string length
683D: BIT   $113D      ;Check the reverse flag
6840: BPL   $6847      ;If it is to be non-reversed text, then branch
6842: LDA   #$12        ;Value for reverse mode
6844: JSR   $C00C      ;Turn on the reverse mode
6847: CPY   $116E      ;Have we outputted the whole string yet?
684A: BEQ   $6858      ;If so, then exit
684C: JSR   $03B7      ;Get a character of the string
684F: JSR   $A845      ;Enable BANK 15
6852: JSR   $C00C      ;Output the character to the screen
6855: INY                ;Move to the next character
6856: BNE   $6847      ;Branch to continue
6858: BIT   $113D      ;Check the reverse flag
685B: BPL   $6862      ;If the reverse mode is not on, then exit
685D: LDA   #$92        ;Value to turn off reverse mode
685F: JSR   $C00C      ;Output it
6862: RTS                ;Exit
6863: JSR   $A074      ;Ensure the graphics screen has been allocated
6866: LDA   $11EC      ;Get page number for the Uppercase characters
6869: STA   $1168      ;And save it
686C: LDA   $86         ;Get the current foreground color
686E: TAX                ;And save it
686F: PHA                ;Onto the stack
6870: LDA   $83         ;Get the current color source
6872: PHA                ;And save it onto the stack
6873: BIT   $D8         ;Check which graphics mode we are in

```



---

```

6875: BPL   $6885   ;If it is not multicolor mode, then branch
6877: PLA                               ;Get the color source back
6878: BEQ   $6890   ;If color source was background, then branch
687A: LSR                               ;Shift bit 0 of the color source into the carry
687B: BEQ   $6890   ;If the color source was 1, then branch
687D: LDX   $84     ;Get the multicolor 1 value
687F: BCC   $6890   ;If the color source was 2, then branch
6881: LDX   $85     ;Get the multicolor 2 value
6883: BCS   $6890   ;If the color source was 3, then branch
6885: LDX   $86     ;Get the current foreground color
6887: PLA                               ;Get the color source back
6888: BNE   $6890   ;If it was not zero, then branch
688A: JSR   $A845   ;Enable BANK 15
688D: LDX   $D021   ;Get the current background color
6890: STX   $86     ;Save it as the current foreground color
6892: LDX   $115F   ;Get the selected row
6895: LDY   #0      ;Clear the character
6897: STY   $1160   ;Counter
689A: LDY   $1160   ;Get the counter as an index
689D: INC   $1160   ;Increment the counter by one
68A0: JSR   $03B7   ;Get a character of the string
68A3: STA   $FF03   ;Enable BANK 14
68A6: DEC   $116E   ;Subtract one from the string length
68A9: BMI   $68D7   ;If we are done, then exit
68AB: CMP   #$0E    ;Is the character CHR$(14) for lowercase?
68AD: BNE   $68B4   ;If not, then branch
68AF: LDA   $11EB   ;Get the page number of the Lowercase
68B2: BNE   $68BB   ;Character set and branch
68B4: CMP   #$8E    ;Is it the value for the Uppercase characters?
68B6: BNE   $68C0   ;If not, then branch
68B8: LDA   $11EC   ;Get page number of Uppercase character set
68BB: STA   $1168   ;And save it
68BE: BNE   $68C9   ;Branch to output
68C0: LDY   $115E   ;Get the current column for output
68C3: JSR   $68DB   ;Output the character to the HIRES screen
68C6: INC   $115E   ;Move to the next column
68C9: CPY   #$27    ;Was the last column 39?
68CB: BCC   $689A   ;If not, then continue
68CD: LDY   #0      ;If it was column 39,
68CF: STY   $115E   ;Set the column to zero
68D2: INX                               ;And increment the row by 1
68D3: CPX   #$18    ;Have we passed the maximum number of rows?
68D5: BCC   $689A   ;If not, then continue
68D7: PLA                               ;Get the foreground color back
68D8: STA   $86     ;And save it
68DA: RTS                               ;Exit the routine

```

```
*****
*
* This routine is used to output a character to the
* HIRES screen. On entry to this routine, the
* accumulator must hold the character to be outputted.
* The Y and X registers must hold the row and column
* respectively, of where you wish the character to be
* stored. Location $113D must have bit 7 set if a
* reversed character is to be outputted and location
* $1168 must hold the page number (MSB) of the address
* of the character set you wish to use. On exit from
* this routine, the A, X, and Y registers are
* preserved and the system is configured to BANK 14.
*
*****
```

```
68DB: PHA                ;Save the character to output
68DC: JSR    $9C70        ;Fill specified color cell with selected color
68DF: TYA                ;Move the column value to the accumulator
68E0: CLC                ;Clear the carry for addition
68E1: ADC    $C033,X      ;Add column to LSB of appropriate screen address
68E4: STA    $8C          ;Save the result
68E6: LDA    $C04C,X      ;Get the MSB of the screen address
68E9: ADC    #0           ;Add the carry to it
68EB: ASL    $8C          ;Multiply the 16 bit
68ED: ROL                ;Value by 8
68EE: ASL    $8C          ;To calculate the appropriate
68F0: ROL                ;HIRES screen address
68F1: ASL    $8C
68F3: ROL
68F4: STA    $8D          ;Save the MSB of the address
68F6: STA    $FF03        ;Enable BANK14
68F9: LDA    #0           ;Clear a temporary
68FB: STA    $77          ;Storage area
68FD: PLA                ;Get the character to output
68FE: PHA                ;Save it back onto the stack
68FF: ASL                ;Multiply the character value
6900: ROL    $77          ;By 8
6902: ASL                ;Saving any overflow
6903: ASL                ;As the MSB
6904: ROL    $77          ;In location $77
6906: STA    $26          ;Save the result as the LSB of the character set
                        ;Address
6908: LDA    $77          ;Get the MSB
690A: ADC    $1168        ;Add to page of character set to get the MSB
690D: STA    $27          ;Of the proper address in the character ROM of
                        ;that character and save it
```

---

```
690F: TYA                ;Move the column to the accumulator
6910: PHA                ;And save it onto the stack
6911: LDY    #$07        ;Index - 1 to the number of bytes in the
                        ;character definition
6913: LDA    $113D       ;Get the reverse flag
6916: ASL                ;And move reverse bit, if any, into the carry
6917: LDA    ($26),Y     ;Get a byte of the character definition
6919: BCC    $691D       ;If not reverse, then branch
691B: EOR    #$FF        ;Invert the character definition
691D: BIT    $D8         ;Check which type of graphics mode we are in
691F: BPL    $694C       ;If it is not the multicolor mode, then branch
6921: AND    #$AA        ;Mask the bits for multicolor 2
6923: STA    $77         ;And save the result
6925: LDA    $83         ;Get the current color source
6927: BNE    $6938       ;If it is not background (0), then branch
6929: LDA    $77         ;Get the modified bit pattern
692B: BCS    $6934       ;If the reverse flag is set, then branch
692D: LSR                ;Shift the bit pattern right one bit
692E: EOR    $77        ;Invert the bits
6930: EOR    #$AA        ;To generate the reverse value
6932: BNE    $694C       ;Branch to store it in the bit map
6934: ORA    #$55        ;Set the even bits for multicolor 1
6936: BNE    $694C       ;And branch to store it in the bit map
6938: CMP    #$02        ;Is the color source 2?
693A: BNE    $6940       ;If not, then branch
693C: LDA    $77         ;Get the modified bit pattern
693E: BCS    $694C       ;If the reverse is specified, then branch
6940: BCC    $6949       ;If the color source is less than 2, then branch
6942: LDA    $77         ;Get the modified bit pattern
6944: LSR                ;Move it one bit to the right
6945: EOR    $77        ;Invert the odd bits for multicolor 2
6947: BCC    $694C       ;Branch to output it to the bit map
6949: LDA    $77         ;Get the modified bit pattern
694B: LSR                ;Move it one bit to the right
694C: STA    ($8C),Y     ;Store character byte to the HIRES screen
694E: DEY                ;Move to next byte of character definition
694F: BPL    $6913       ;Continue until eight bytes have been done
6951: PLA                ;Get the column value back
6952: TAY                ;And save it in the Y register
6953: PLA                ;Get the character that was outputted
6954: RTS                ;And exit
```

```

*****
*
*          BASIC LOCATE COMMAND
*
* Command syntax:  LOCATE X,Y coordinates
*
* NOTE:  X,Y coordinates = the position to move the
*          bit map pixel cursor to
*
* This command allows you to place the bit map pixel
* cursor to a specified position on the screen.  The
* values of the X, Y coordinates may range from 0, 0
* to 320, 200.
*
*****

```

```

6955: JSR   $A074      ;Ensure GRAPHICS area has been allocated, if not
                        ;then generate a 'NO GRAPHICS AREA' error
6958: LDX   #$04      ;Obtain the coordinates
695A: JSR   $9E70      ;And store them in locations $1135 - $1138
695D: JMP   $9DF2      ;Move X and Y destination coordinates
                        ;Into locations $1131 - $1134

```

```

*****
*
*          BASIC SCALE COMMAND
*
* Command syntax:  SCALE n [,X max,Y max]
*
* NOTE:          n = 0 (turn the scaling off)
*                1 (turn the scaling on)
*
*          X max = the X coordinate may be scaled
*                from 0 - 32767
*
*          Y max = the Y coordinate may be scaled
*                from 0 - 32767
*
* This command allows you to scale the X and Y
* coordinates in the graphics mode.
*
*****

```

```

6960: JSR   $87F4      ;Convert first parameter into Hex in X register
6963: CPX   #2         ;Is the value less than two?
6965: BCC   $696A      ;Yes, then branch
6967: JMP   $7D28      ;If it is greater than two, then generate an

```

```

; 'ILLEGAL QUANTITY' error
696A: STX   $116A   ; Save the flag to indicate if scaling is to be
; used in SCALEM
696D: JSR   $0386   ; Get the second parameter (X scaling factor)
6970: BNE   $6986   ; If the second parameter is specified, then
; branch
6972: LDX   #0      ; If the first parameter is zero or the second
; parameter is not specified,
6974: LDA   #$50    ; Then set the X and Y scaling
6976: LDY   #$32    ; Factor to
6978: BIT   $D8     ; To 1023 x 1023 if the multicolor
697A: BPL   $697D   ; Mode is not enabled
697C: LSR   ; If the multicolor mode is enabled,
697D: STX   $87     ; Then set the
697F: STA   $88     ; X and Y scaling factors to
6981: STX   $89     ; The default value of
6983: STY   $8A     ; 2047 x 1023
6985: RTS                ; Exit this routine

```

```

*****
*
* This routine is used to obtain the X and Y scaling
* factors the programmer has requested by using the
* following formulas;
*
*
*          SCALE - X = 20971200/X MAX
*
*          and
*
*          SCALE - Y = 13107000/Y MAX
*
* Upon entry to this routine, TXTPTR is pointing to
* the comma before the X-MAX parameter in the BASIC
* SCALE command and the result of the aforementioned
* formulas are stored as follows:
*
*          SCALE - X = $87, $88
*
*          and
*
*          SCALE - Y = $89, $8A
*
*****

```

```

6986: JSR   $69C4   ; Get the value of the X - scaling factor and
; Place it into FAC1

```

```

6989: LDA  #D8      ;Move the value
698B: LDY  #69      ;20971200 into
698D: JSR  $8A89    ;FAC2
6990: JSR  $8B4C    ;FAC1 = 20971200/ the X - scaling factor
6993: JSR  $8815    ;Convert the result into a two byte integer in
                    ;The Y register and the accumulator (LSB, MSB)
6996: CMP  #0       ;Check to see if the value
6998: BNE  $699E    ;Is equal to zero
699A: CPY  #0       ;And if it is, then
699C: BEQ  $69D5    ;Branch to generate an 'ILLEGAL QUANTITY' error
699E: PHA                ;Save the MSB of the value onto the stack
699F: TYA                ;Move the LSB into the accumulator
69A0: PHA                ;Save the LSB onto the stack
69A1: JSR  $69C4    ;Get the value of the Y - scaling factor and
                    ;place it into FAC1
69A4: LDA  #DD      ;Move the value
69A6: LDY  #69      ;13107000 into
69A8: JSR  $8A89    ;FAC2
69AB: JSR  $8B4C    ;FAC1 = 13107000/ the Y - scaling factor
69AE: JSR  $8815    ;Convert the result into a two byte integer in
                    ;The Y register and the accumulator (LSB, MSB)
69B1: CMP  #0       ;Check to see if the value
69B3: BNE  $69B9    ;Is equal to zero
69B5: CPY  #0       ;And if it is, then branch to
69B7: BEQ  $69D5    ;generate an 'ILLEGAL QUANTITY' error
69B9: STY  $89      ;Save the Y - scaling factor
69BB: STA  $8A      ;Into SCALE - Y
69BD: PLA                ;Get the X - scaling
69BE: STA  $87      ;Factor off of the stack
69C0: PLA                ;And save it
69C1: STA  $88      ;Into SCALE - X
69C3: RTS                ;Exit the routine

```

```

*****
*
* This routine first checks to ensure that the
* character that TXTPTR is pointing to is a comma.
* If the character is a comma, then the routine skips
* it and converts the ASCII value of the character
* that follows the comma into its equivalent Floating
* Point Number in FAC1. The routine then ensures that
* the number is positive and less than 32767.
*
*****

```

```

69C4: JSR  $795C    ;Check to ensure that there is a comma
                    ;Between the parameters

```

```

69C7: JSR  $77D7      ;Evaluate the expression
69CA: LDA  $68        ;If the value is not positive, then
69CC: BMI  $69D5      ;branch to generate an 'ILLEGAL QUANTITY' error
69CE: LDA  $63        ;Get the exponent of the Floating Point Number
                        ;(First parameter)
69D0: CMP  #$90       ;And ensure that the value is less than +32767
69D2: BCS  $69D5      ;If not, generate an 'ILLEGAL QUANTITY' error
69D4: RTS                    ;Exit the routine
69D5: JMP  $7D28      ;Generate an 'ILLEGAL QUANTITY' error
    
```

```

*****
*
*                               X - SCALE
*
* The following value is used to obtain the scaling
* factor for the X coordinate. This value is stored
* in Floating Point format.
*
*****
    
```

```

EX  M1  M2  M3  M4
--  --  --  --  --
    
```

```

69D8: .BYTE $99,$1F,$FF,$60,$00 ; 20971200
    
```

```

*****
*
*                               Y - SCALE
*
* The following value is used to obtain the scaling
* factor for the Y coordinate. This value is stored
* in Floating Point format.
*
*****
    
```

```

EX  M1  M2  M3  M4
--  --  --  --  --
    
```

```

69DD: .BYTE $98,$47,$FF,$38,$00 ; 13107000
    
```

```

*****
*
*                               BASIC COLOR COMMAND
*
* Command syntax: COLOR source, color
*
* NOTE: source = 0 (40 column background color)
*          1 (40 column foreground color)
*          2 (multicolor 1)
*
*****
    
```

```

*           3 (multicolor 2)           *
*           4 (40 column border color) *
*           5 (character color - 40 or 80 column) *
*           6 (80 column background color) *
*
*           color = the value of the color to be used *
*           (1 = white, 2 = black, etc...) *
*
* This command lets you set colors for the *
* various screen modes. *
*
*****

```

```

69E2: JSR  $87F4      ;Convert ASCII to numeric
69E5: CPX  #7         ;Is the source number greater than 6?
69E7: BCS  $6A49      ;If so, generate an 'ILLEGAL QUANTITY' error
69E9: STX  $77        ;Save the source number
69EB: JSR  $8809      ;Get the color number
69EE: DEX           ;Subtract one
69EF: CPX  #16        ;Is the color number greater than 16?
69F1: BCS  $6A49      ;If so, generate an 'ILLEGAL QUANTITY' error
69F3: JSR  $A845      ;BANK 15 COMMAND
69F6: TXA           ;Transfer the color number into the accumulator
69F7: LDX  $77        ;Get the source number
69F9: CPX  #1         ;Is it for the 40 column foreground?
69FB: BEQ  $6A04      ;If it is, then branch
69FD: BCS  $6A08      ;If it is greater than one, then branch
69FF: STA  $D021      ;Set the background color
6A02: BNE  $6A43      ;If color number is not for black, then branch
6A04: STA  $86        ;Save the color number
6A06: BEQ  $6A43      ;If it is for black (0), then branch
6A08: CPX  #3         ;Source color = multicolor 2?
6A0A: BEQ  $6A12      ;If so, then branch
6A0C: BCS  $6A16      ;Greater than three, then branch
6A0E: STA  $84        ;Save multicolor 1 color number
6A10: BNE  $6A43      ;If color number is not for black, then branch
6A12: STA  $85        ;Save the color number
6A14: BEQ  $6A43      ;If color number is for black, then branch
6A16: CPX  #5         ;Is it the character color?
6A18: BEQ  $6A21      ;If it is, then branch
6A1A: BCS  $6A32      ;It is the 80 column background color
6A1C: STA  $D020      ;Set the border color
6A1F: BNE  $6A43      ;If color number is not for black, then branch
6A21: BIT  $D7        ;Check to see if we are in the 40 column mode
6A23: BPL  $6A2D      ;If we are, then branch
6A25: TAX           ;Transfer the color code into the accumulator
6A26: LDA  $F1        ;Get the current character color code

```



```

6A28: AND   #$F0      ;Drop bits 4 - 7
6A2A: ORA   $6A4C,X   ;Add the new character color from the table
6A2D: STA   $F1       ;Save the new character color code
6A2F: JMP   $6A43

```

```

*-----*
*
*   This section of code handles the 80 column screen.
*
*-----*

```

```

6A32: TAX
6A33: LDA   #26       ;Address VDC register 26 to set
6A35: STA   $D600     ;The new Foreground/Background color
6A38: LDA   $D601     ;Get the register's current contents
6A3B: AND   #$F0      ;Drop bits 4 - 7 (foreground)
6A3D: ORA   $6A4C,X   ;Add the new character color from the table
6A40: STA   $D601     ;Save the new character color code
6A43: JSR   $6A5C
6A46: JMP   $9E1E
6A49: JMP   $7D28     ;Generate an 'ILLEGAL QUANTITY' error

```

```

*-----*
*
*           80 COLUMN COLOR CODES FOR THE RGB MONITOR
*
*   NOTE:  R = Red, B = Blue, G = Green, I = Intensity
*
*-----*

```

```

6A4C: .BYTE  #%00000000 ; 01 BLACK
6A4D: .BYTE  #%00001111 ; 02 WHITE
6A4E: .BYTE  #%00001000 ; 03 Dark Red
6A4F: .BYTE  #%00000111 ; 04 Light Cyan
6A50: .BYTE  #%00001011 ; 05 Light Purple
6A51: .BYTE  #%00000100 ; 06 Dark Green
6A52: .BYTE  #%00000010 ; 07 Dark Blue
6A53: .BYTE  #%00001101 ; 08 Light Yellow
6A54: .BYTE  #%00001010 ; 09 Dark Purple
6A55: .BYTE  #%00001100 ; 10 Brown
6A56: .BYTE  #%00001001 ; 11 Light Red
6A57: .BYTE  #%00000110 ; 12 Dark Cyan
6A58: .BYTE  #%00000001 ; 13 Medium Gray
6A59: .BYTE  #%00000101 ; 14 Light Green
6A5A: .BYTE  #%00000011 ; 15 Light Blue
6A5B: .BYTE  #%00001110 ; 16 Light Gray

```

```

6A5C: LDA   $86           ;Get the current foreground color
6A5E: ASL                   ;Shift the lower nybble into the upper nybble
6A5F: ASL                   ;As the foreground is at $D021
6A60: ASL                   ;In the LSB format
6A61: ASL
6A62: STA   $77
6A64: JSR   $A845         ;Enable BANK 15
6A67: LDA   $D021        ;Get the current foreground color
6A6A: AND   #$0F         ;Drop off bits 0 - 3 (background)
6A6C: ORA   $77         ;Add the new foreground color
6A6E: STA   $03E2        ;Save as packed foreground/background colors
6A71: LDA   $84         ;Get Multicolor 1's color
6A73: ORA   $77         ;Add the new foreground color
6A75: STA   $03E3        ;Save as packed foreground/multicolor 1's nybble
6A78: RTS                   ;Exit routine

```

```

*****
*
*           BASIC SCNCLR COMMAND           *
*
* Command syntax:  SCNCLR screen mode    *
*
* NOTE:  screen mode = 0 (40 column text) *
*                1 (bit map)             *
*                2 (split screen bit map) *
*                3 (multicolor bit map)   *
*                4 (split screen multicolor *
*                   bit map)             *
*                5 (80 column text)      *
*
* This command lets you clear the screen *
* specified in the 'screen mode' parameter. *
* If no screen mode number is specified, *
* the GRAPHICS screen is cleared.        *
* However, if there is not a GRAPHICS *
* screen present, then the current text *
* screen is cleared.                    *
*
*****

```

```

6A79: BNE   $6A8F         ;If there is a mode number, then branch
6A7B: JSR   $818C         ;Find out which mode we are in
6A7E: CMP   #5           ;Mode 5 (80 COLUMN TEXT) ?
6A80: BCC   $6A8B         ;If it is less than 5, then branch
6A82: SBC   #5           ;Subtract 5 from the mode number
6A84: BEQ   $6ADE         ;If the mode number was 5, then branch
6A86: PHA                   ;Save the mode number onto the stack
6A87: JSR   $6ADE         ;Clear the 80 column text screen
6A8A: PLA                   ;Get the mode value back off of the stack

```

---

```

6A8B: TAX                ;Move the value to the X register
6A8C: JMP $6A9B          ;Clear the proper screen
6A8F: JSR $87F4          ;Get the mode number into the X register
6A92: CPX #5             ;Is it the 80 column mode?
6A94: BEQ $6ADE          ;If it is, then branch to clear the 80
                        ;Column text screen
6A96: BCC $6A9B          ;If it is less than 5, then branch
6A98: JMP $7D28          ;Generate an 'ILLEGAL QUANTITY' error
6A9B: TXA                ;Move the mode value to the accumulator
6A9C: BEQ $6AF2          ;If the mode is zero, then branch to
                        ;Clear the 40 column text screen
6A9E: JSR $A074          ;Ensure that GRAPHICS screen has been allocated
6AA1: TXA                ;Save the mode value
6AA2: PHA                ;Onto the stack
6AA3: AND #1             ;Mask bit 0 to check for mode 1 and mode 3
6AA5: BNE $6AC8          ;If it is mode 1 or 3, then branch
6AA7: JSR $6AF2          ;Clear the 40 column text screen
6AAA: LDA $D7            ;Get screen mode (40/80 column)
6AAC: PHA                ;Save it onto the stack
6AAD: BPL $6AB2          ;If we are in the 40 column mode, then branch
6AAF: JSR $FF5F          ;Switch to 40 column mode
6AB2: LDA $0A34          ;Get the split screen raster line value
6AB5: SEC                ;Set the carry for subtraction
6AB6: SBC #$30           ;Subtract 48 from it
6AB8: LSR                ;And divide the
6AB9: LSR                ;Result
6ABA: LSR                ;By 8 to acquire the new value
6ABB: TAX                ;Move that value to the X register
6ABC: LDY #0             ;Set cursor column to 0
6ABE: CLC                ;Clear the carry to set the cursor position
6ABF: JSR $928D          ;Set the cursor position
6AC2: PLA                ;Get the screen mode back
6AC3: BPL $6AC8          ;If we are in the 40 column mode, then branch
6AC5: JSR $FF5F          ;Switch to 40 column mode
6AC8: PLA                ;Get the mode value off of the stack
6AC9: AND #$02           ;Mask bit 1 to check for mode 3
6ACB: BEQ $6AD0          ;If it is not mode 3, then branch
6ACD: JSR $6B17          ;Fill color memory at $D800 with the
                        ;Color value from location $85
6AD0: JSR $6B30          ;Clear the bit map screen
6AD3: LDA #0             ;
6AD5: LDX #$03           ;Load the X register with the number of bytes
                        ;Minus one to clear
6AD7: STA $1131,X        ;Clear the current
6ADA: DEX                ;X and Y
6ADB: BPL $6AD7          ;Coordinates
6ADD: RTS                ;And exit

```

```
*****
*
* This routine is used to clear the 80 column text
* screen. It first checks if the current screen mode
* is 80 column and if it is not, the routine SWAPPER
* at $FF5F is called to switch into the 80 column
* screen. Then a CHR$(147) is printed to clear the
* screen. On exiting, the routine calls SWAPPER to
* return the screen to the 40 column mode screen
* if it was enabled on entry to this routine.
*
*****
```

```
6ADE: LDA    $D7          ;Get the current screen mode (40/80 column)
6AE0: PHA          ;Save it onto the stack
6AE1: BMI    $6AE6       ;If we are in the 80 column mode, then branch
6AE3: JSR    $FF5F       ;Switch to the 80 column mode
6AE6: LDA    #147        ;CHR$ value for Clear Screen
6AE8: JSR    $9269       ;Clear the text screen
6AEB: PLA          ;Get the previous screen mode off the stack
6AEC: BMI    $6AF1       ;If it is 80 column mode, then branch to exit
6AEE: JSR    $FF5F       ;Switch to the 40 column mode
6AF1: RTS          ;Exit the routine
```

```
*****
*
* This routine operates the same as the routine at
* $6ADE only this routine clears the 40 column screen.
*
*****
```

```
6AF2: LDA    $D7          ;Get the current screen mode (40/80 column)
6AF4: PHA          ;Save it onto the stack
6AF5: BPL    $6AFA       ;If we are in the 40 column mode, then branch
6AF7: JSR    $FF5F       ;Switch to the 40 column mode
6AFA: LDA    #147        ;Character for clearing the screen
6AFC: JSR    $9269       ;Print it to the screen
6AFF: PLA          ;Get the old screen mode back
6B00: BPL    $6B05       ;If it is the 40 mode, then branch to exit
6B02: JSR    $FF5F       ;Switch to the 80 column mode
6B05: RTS          ;Exit routine
```

```
*****
*
*                               FILL
*
*                               Fill from Y*256 to (Y+X*256)-1
*
```

```

* On entry into this routine, the accumulator holds *
* the fill value, the Y register holds the MSB of the *
* starting address to be filled, and the X register *
* holds the number of pages to be filled. *
* *
* EXAMPLE: A = $00, Y = $20, X = $20 *
* *
* This combination of the registers would fill from *
* $2000 to $3FFF with $00. *
* *
*****

```

```

6B06: STY $8D ;Save the MSB of the starting address
6B08: LDY #0 ;Index
6B0A: STY $8C ;Zero the LSB of the starting address
6B0C: STA ($8C),Y ;Fill from $XX00 to $XXFF
6B0E: DEY ;Decrement the index pointer
6B0F: BNE $6B0C ;If 256 bytes have not been filled, then branch
6B11: INC $8D ;Add one to the MSB of the starting address
6B13: DEX ;Decrement the number of pages to be filled
6B14: BNE $6B0C ;Continue until X = 0
6B16: RTS ;Exit

```

```

*****
* *
* FILCLR *
* *
* FILl CoLoR memory *
* *
* This routine is used to fill color memoy from $D800 *
* to $DBFF with the color value specified in Location *
* $85. Note: This routine clears location $01 to *
* ensure that COLOR RAM 0 is always used. If you *
* want to fill COLOR RAM 1, you must alter the DDR *
* register at Location $00. *
* *
*****

```

```

6B17: JSR $A845 ;Enable BANK 15
6B1A: SEI ;Stop the background jobs
6B1B: LDA $01 ;Get the data register value
6B1D: PHA ;Save it onto the stack
6B1E: AND #$FE ;Drop bit 0 to ensure that COLOR RAM 0 is used
6B20: STA $01 ;Save the value
6B22: LDA $85 ;Get the color value
6B24: LDY #$D8 ;Page number to start at
6B26: LDX #4 ;Number of pages to fill

```

```

6B28: JSR   $6B06      ;Fill from $D800 to $DBFF with color
6B2B: PLA                      ;Restore the data register
6B2C: STA   $01        ;To its original value
6B2E: CLI                      ;Restore the background jobs
6B2F: RTS                      ;And exit the routine

```

```

*****
*
*      CLEAR THE BIT MAP SCREEN FROM $2000 - $3FFF      *
*
*  This routine is used to clear the bit map screen from*
*  $2000 to $3FFF and to initialize the color for the  *
*  bit map by storing the value that is in $03E2 for a  *
*  normal bit map or from $03E3 for the multicolor bit  *
*  map into $1C00 to $1FFF.  On exit, the routine re-  *
*  initializes the sprite ID values at $1FF8 to $1FFF.  *
*
*****

```

```

6B30: LDA   #0          ;Fill character
6B32: LDY   #$20        ;Fill from $2000 to $4000
6B34: LDX   #$20        ;With zeros (clear bit map screen)
6B36: JSR   $6B06      ;NOTE: See comments at $6B06
6B39: LDA   $03E2      ;Get the Foreground/Background color from FGGB
6B3C: BIT   $D8        ;Check for the screen mode
6B3E: BPL   $6B43      ;If multicolor mode is not being used, then
                          ;branch
6B40: LDA   $03E3      ;Get the Foreground/Background fill
                          ;Character from FGMC1
6B43: JSR   $A845      ;Enable BANK 15
6B46: LDY   #$1C        ;Fill from $1C00 to $2000
6B48: LDX   #4         ;With color
6B4A: JSR   $6B06      ;Fill the specified area
6B4D: LDX   #$3F
6B4F: LDY   #$07      ;Reintialize
6B51: TXA                      ;The
6B52: STA   $1FF8,Y     ;Sprite
6B55: DEX                      ;ID
6B56: DEY                      ;Table
6B57: BPL   $6B51
6B59: RTS                      ;And exit the routine

```

```

*****
*
*          BASIC GRAPHIC COMMAND          *
*
* Command syntax: GRAPHIC graphic mode [,clr][,sl] *
*          GRAPHIC CLR                    *
*
* NOTE:  graphic mode = 0 (40 column text) *
*          1 (bit map)                    *
*          2 (split screen bit map)       *
*          3 (multicolor bit map)        *
*          4 (split screen multicolor    *
*          bit map)                       *
*          5 (80 column text)            *
*
*          clr = 0 (do not clear the bit map *
*          screen)                        *
*          1 (clear the bit map screen)   *
*
*          sl = the starting line number of *
*          the split screen               *
*
* This command allows you to enter one of the six *
* GRAPHICS modes that are in the 128.  When the *
* GRAPHIC CLR format is used, the bit map screen *
* is cleared and then deallocated.        *
*
*****

```

```

6B5A:  CMP   #$9C      ;Token for CLR?
6B5C:  BNE   $6B69     ;If not, then branch

```

```

-----*
*
*          GRAPHIC CLR COMMAND ENTRY POINT          *
*
* The routine is entered here if the GRAPHIC CLR *
* command was issued.  The routine first moves BASIC *
* START back to $1C00 and resets all of the various *
* BASIC pointers that were altered.  The routine exits *
* by returning the system to the text mode.        *
*
*-----*

```

```

6B5E:  JSR   $A022     ;Move BASIC and update the pointers if needed
6B61:  JSR   $0380     ;Get the next character after the CLR token

```

```

6B64: LDA  #0          ;Return the system
6B66: STA  $D8        ;To the text mode
6B68: RTS             ;Exit the routine

*****
*
*                SELECT GRAPHICS MODE
*
* This routine is entered here by the GRAPHIC command. *
*
*****

6B69: JSR  $87F4      ;Get the GRAPHICS mode into the X register
6B6C: TXA             ;Move the value to the accumulator
6B6D: PHA             ;Save it onto the stack
6B6E: CPX  #$05       ;If the value is for the 80 column screen,
6B70: BEQ  $6BB4      ;Then branch
6B72: BCS  $6BC1      ;If it is greater than 5, then generate
                        ;An 'ILLEGAL QUANTITY' error
6B74: LDA  $6BC4,X    ;Get proper graphics mode flag from the table
6B77: STA  $D8        ;Enable that graphics mode
6B79: BEQ  $6B82      ;If the mode was the text mode, then branch
6B7B: JSR  $9F4F      ;Move BASIC and update the pointers if needed
6B7E: BIT  $D8        ;Check the current Graphics mode
6B80: BVC  $6B89      ;If it is not Graphic 2, then branch
6B82: BIT  $D7        ;Check the screen mode (40/80)
6B84: BPL  $6B89      ;If we are in the 40 column mode, then branch
6B86: JSR  $FF5F      ;Switch to 80 column mode
6B89: JSR  $9E1C      ;Get screen clear flag, if any, into X register
6B8C: CPX  #2         ;If it is greater than 1,
6B8E: BCS  $6BC1      ;Then generate an 'ILLEGAL QUANTITY' error
6B90: TXA             ;Save the flag
6B91: PHA             ;Onto the stack
6B92: LDX  #$14       ;Default split screen row of 20
6B94: JSR  $9E1E      ;Get the split screen row, if any
6B97: CPX  #$1A       ;If the value is greater than 24,
6B99: BCS  $6BC1      ;Then generate an 'ILLEGAL QUANTITY' error
6B9B: TXA             ;Move the value to the accumulator
6B9C: ASL             ;Multiply the row
6B9D: ASL             ;By 8
6B9E: ASL             ;And
6B9F: ADC  #$30       ;Add 48 to calculate the raster line to stop bit
                        ;map and start text
6BA1: STA  $0A34      ;Save as split screen raster Line
6BA4: PLA             ;Get the clear screen flag
6BA5: TAY             ;And move it to the Y register
6BA6: PLA             ;Get the Graphics mode value

```



```

6BA7: TAX                ;And move it to the X register
6BA8: TYA                ;Move the clear screen flag to the accumulator
6BA9: BEQ $6BAE          ;If the screen is not to be cleared, then branch
6BAB: JSR $6A92          ;Clear the screen
6BAE: LDA #0             ;Clear the SCALE
6BB0: STA $116A          ;Mode flag
6BB3: RTS                ;Exit the routine
6BB4: BIT $D7            ;40 or 80 column screen?
6BB6: BMI $6B89          ;If it is the 80 column mode, then branch
6BB8: LDA $D8            ;Get the current screen mode
6BBA: AND #$BF           ;Switch from GRAPHIC 2 to GRAPHIC 1
6BBC: STA $D8            ;Save it
6BBE: JMP $6B86          ;Continue
6BC1: JMP $7D28          ;Generate an 'ILLEGAL QUANTITY' error
    
```

```

*****
*
*                               *
*          TABLE FOR GRAPHICS MODE          *
*
* This is a table of values which are stored in *
* Location $D8 to enable the specified GRAPHICS mode. *
*
*****
    
```

BIT - 76543210 GRAPHIC MODE

```

-----
6BC4: .BYTE #%00000000 ;GRAPHIC 0
6BC5: .BYTE #%00100000 ;GRAPHIC 1
6BC6: .BYTE #%01100000 ;GRAPHIC 2
6BC7: .BYTE #%10100000 ;GRAPHIC 3 and GRAPHIC 5
6BC8: .BYTE #%00110000 ;GRAPHIC 4
    
```

```

*****
*
*                               *
*          BASIC BANK COMMAND          *
*
* Command syntax:  BANK n             *
*
* NOTE:  n = the bank number value (0 - 15) *
*
* This command allows you to select one of the *
* sixteen banks that are in the 128. *
*
*****
    
```

```

6BC9: JSR $87F4          ;Get the BANK number into the X register
6BCC: CPX #16            ;If the BANK number specified is greater than
    
```

```

6BCE: BCS $6BD4 ;15, then generate an 'ILLEGAL QUANTITY' error
6BD0: STX $03D5 ;Set the new BANK NUMBER
6BD3: RTS ;And exit
6BD4: JMP $7D28 ;Generate an 'ILLEGAL QUANTITY' error
    
```

```

*****
*
*                               BASIC SLEEP COMMAND
*
* Command syntax: SLEEP n
*
* NOTE: n = the delay time in seconds (1 - 65535)
*
* This command allows you to create a delay in the
* BASIC program's execution.
*
* This routine sets up the countdown timer at $0A1D-
* $0A1F to the sleeper value specified multiplied by
* 60. The routine then loops, checking the STOP key,
* until the timer has counted down to 0.
*
*****
    
```

```

6BD7: JSR $8812 ;Get the SLEEP value into the Y register
        ;And the accumulator
6BDA: LDX #0 ;Set the initial SLEEP timer MSB to 0
6BDC: SEI ;Stop the SLEEP timer
6BDD: STY $0A1D ;Set timer low
6BE0: STA $0A1E ;Set timer mid
6BE3: STX $0A1F ;Set timer hi
6BE6: JSR $6C0C ;Multiply the timer value by 2
6BE9: JSR $6C16 ;Add original timer value and multiply by 3
6BEC: JSR $6C09 ;Multiply the SLEEP value and counter by 4
6BEF: LDY $0A1D ;Get the timer low
6BF2: LDA $0A1E ;Get the timer mid
6BF5: LDX $0A1F ;Get the timer hi
6BF8: JSR $6C09 ;Multiply the timer by 4
6BFB: JSR $6C16 ;Add the timer values in the A, X, and Y
        ;registers to the timer
6BFE: CLI ;Start timer countdown
6BFF: JSR $4BB5 ;Check the STOP key
6C02: LDX $0A1F ;Get the timer hi
6C05: INX ;And add one to it
6C06: BNE $6BFF ;Loop until the timer has counted down to 0
6C08: RTS ;Exit the routine
    
```

```

*-----*
*
*           MULTIPLY SLEEP TIME
*
* If entered here the routine will multiply the timer
* at $0A1D - $0A1F by 4.
*
*-----*

```

```
6C09: JSR   $6C0C      ;Multiply the SLEEP timer by 2
```

```

*-----*
*
* If entered here the timer will be multiplied by 2.
*
*-----*

```

```

6C0C: ASL   $0A1D      ;Multiply the countdown timer
6C0F: ROL   $0A1E      ;By 2
6C12: ROL   $0A1F
6C15: RTS

```

```

*-----*
*
* This routine adds the values in the Y, A and X reg-
* isters to the SLEEP timer at $0A1D, $0A1E and $0A1F.
*
*-----*

```

```

6C16: PHA           ;Save the timer mid value onto the stack
6C17: TYA           ;Move the timer low value to the Y register
6C18: ADC   $0A1D    ;Add it to the present timer value
6C1B: STA   $0A1D    ;Save it back
6C1E: PLA           ;Get the timer mid value off of the stack
6C1F: ADC   $0A1E    ;Add it to the present timer value
6C22: STA   $0A1E    ;Save it back
6C25: TXA           ;Move the timer hi value to the accumulator
6C26: ADC   $0A1F    ;Add it to the present timer value
6C29: STA   $0A1F    ;Save it back
6C2C: RTS           ;And exit the routine

```

```

*****
*
*           BASIC WAIT STATEMENT
*
* Command syntax:  WAIT location,mask 1[,mask 2]
*

```

```

* The WAIT statement takes the value in the address      *
* that is specified after the WAIT command and          *
* performs a LOGICAL AND operation with the value in   *
* MASK 1. The result from the LOGICAL AND is then     *
* EXCLUSIVE ORed with MASK 2. The value in MASK 1     *
* filters out the unwanted bits you do not want to   *
* test and then the value in MASK 2 makes a mirror    *
* image of the result from the MASK 1.                 *
*                                                       *
*****

```

```

6C2D: JSR   $8803      ;Get the address and value to wait on
6C30: STX   $4B        ;Save MASK 1 value
6C32: LDX   #0         ;Zero the index pointer
6C34: JSR   $0386     ;Get the last character again
6C37: BEQ   $6C3C     ;If there is not a MASK 2 parameter, then branch
6C39: JSR   $8809     ;Check for comma and get the value for MASK 2
6C3C: STX   $4C        ;Save the MASK 2 parameter
6C3E: LDY   #0         ;Zero the index pointer
6C40: LDX   $03D5     ;Get the BANK number for this WAIT STATEMENT
6C43: LDA   #$16      ;Get LSB of the ADDRESS
6C45: JSR   $FF74     ;LDA (ADDRESS),Y from any BANK lo/hi mode
6C48: EOR   $4C        ;EOR the value from the address with MASK 2
6C4A: AND   $4B        ;AND the result with MASK 1
6C4C: BEQ   $6C3E     ;Continue to scan ADDRESS until a 1 is found
6C4E: RTS                ;Exit the routine

```

```

*****
*
*
*           BASIC SPRITE COMMAND
*
* Command syntax:  SPRITE number[,on/off][fclr][,p]
*                  [,X-expand][,Y-expand][,smode]
*
* NOTE: number = the sprite number (1-8)
*
*           on/off = 0 (sprite off)
*                  1 (sprite on)
*
*           fclr = the sprite's foreground color (1-16)
*
*           p = 0 (sprites appear in front of objects)
*              1 (sprites appear in back of objects)
*
*           X-expand = 0 (X expansion off)
*                    1 (X expansion on)
*
*

```

```

*           Y-expand = 0 (Y expansion off)           *
*                   1 (Y expansion on)             *
*
*           smode = 0 (standard sprite)             *
*                   1 (multicolor sprite)         *
*
* This command allows you to turn the sprite on or off,*
* set the sprite's color, set the screen priority for *
* the sprite, and expand the X and Y dimensions of    *
* the sprite, as well as determine whether the sprite *
* is to be a standard or multicolor sprite.         *
*
*****

```

```

6C4F: JSR   $6CBB      ;Get sprite number and store at location $77
6C52: JSR   $9E1E      ;Get sprite enable flag, if any, into the X
                        ;Register
6C55: BCC   $6C5C      ;If none was specified, then branch
6C57: LDY   #$15       ;Pointer to $D015 (sprite enable register)
6C59: JSR   $6C9B      ;Enable or disable the sprite
6C5C: JSR   $9E1E      ;Get the sprite foreground color, if any,
                        ;Into the X register
6C5F: BCC   $6C6F      ;If none was specified, then branch
6C61: DEX
6C62: CPX   #$10       ;Specified was greater than 15,
6C64: BCS   $6C98      ;Then generate an 'ILLEGAL QUANTITY' error
6C66: TXA
6C67: LDX   $77        ;Get the sprite number to use as an index
6C69: JSR   $A845      ;Enable BANK 15
6C6C: STA   $D027,X    ;Set the foreground color of the sprite
6C6F: JSR   $9E1E      ;Get the sprite priority flag, if any,
                        ;Into the X register
6C72: BCC   $6C79      ;If none was specified, then branch
6C74: LDY   #$1B       ;Index to $D01B (sprite priority register)
6C76: JSR   $6C9B      ;Set the sprite's priority
6C79: JSR   $9E1E      ;Get the X expansion flag, if any, into the X
                        ;register
6C7C: BCC   $6C83      ;If none was specified, then branch
6C7E: LDY   #$1D       ;Pointer to $D01D (sprite X expansion register)
6C80: JSR   $6C9B      ;Set or clear the sprite's X expansion register
6C83: JSR   $9E1E      ;Get the Y expansion flag, if any, into the X
                        ;register
6C86: BCC   $6C8D      ;If none was specified, then branch
6C88: LDY   #$17       ;Pointer to $D017 (sprite Y expansion register)
6C8A: JSR   $6C9B      ;Set or clear the sprite's Y expansion register
6C8D: JSR   $9E1E      ;Get sprite mode, if any, into the X register
6C90: BCC   $6C97      ;If none was specified, then exit

```

```

6C92: LDY   #$1C      ;Pointer to $D01C (sprite mode register)
6C94: JSR   $6C9B     ;Set the sprite to standard or multicolor
6C97: RTS                    ;Exit the routine
6C98: JMP   $7D28     ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
* This routine is used to set or clear the proper bit
* in each sprite's register as indexed by the Y
* register. On entry, the X register must hold a one
* to set the proper bit or zero to clear it. Location
* $77 must hold the sprite number whose values you
* wish to change. The Y register must hold the LSB
* of the sprite register. Ex: To change location
* $D01B, store a $1B in the Y register.
*
*****

```

```

6C9B: TXA                    ;Move the flag to the accumulator
6C9C: LSR                    ;Divide by 2, and if the flag was greater than
6C9D: BNE   $6C98           ;1, then generate an 'ILLEGAL QUANTITY' error
6C9F: LDX   $77             ;Get the sprite number and use it as an index
6CA1: LDA   $6CB3,X        ;Get the proper sprite bit value
6CA4: JSR   $A845          ;Enable BANK 15
6CA7: ORA   $D000,Y        ;Set the proper bit in the sprite register
6CAA: BCS   $6CAF          ;If the bit was to be set, then branch
6CAC: EOR   $6CB3,X        ;Else reset (clear) the bit
6CAF: STA   $D000,Y        ;Save the result as the new value
6CB2: RTS                    ;And exit

```

```

*****
*
* This is the table used for the COLLISION interrupts
* and by the PLAY command.
*
*****

```

	HEX		BIT - 76543210	DECIMAL VALUE
	---		-----	-----
6CB3:	.BYTE \$01	;	00000001	1
6CB4:	.BYTE \$02	;	00000010	2
6CB5:	.BYTE \$04	;	00000100	4
6CB6:	.BYTE \$08	;	00001000	8
6CB7:	.BYTE \$10	;	00010000	16
6CB8:	.BYTE \$20	;	00100000	32
6CB9:	.BYTE \$40	;	01000000	64
6CBA:	.BYTE \$80	;	10000000	128



---

```

6CCF: BVC    $6CD4    ;And if so, then branch
6CD1: JMP    $796C    ;If a comma was not found, then generate
                        ;A 'SYNTAX' error message
6CD4: STY    $1135    ;Save the X
6CD7: STY    $1137    ;Destination value
6CDA: STA    $1136    ;In X-DEST and
6CDD: STA    $1138    ;In Y-DEST
6CE0: JSR    $6D9E    ;Get the second parameter and set the
                        ;Flags in location $116E (NUMCNT)
6CE3: BIT    $116E    ;Check to see if a comma was found
6CE6: BVC    $6D49    ;And if it was, then branch
6CE8: BMI    $6D24    ;If a semicolon (;) was found, then branch
6CEA: TYA
6CEB: PHA
                        ;Save it onto the stack
6CEC: LDY    #$04
6CEE: JSR    $9A74    ;Calculate the angle from locations $1135, $1136
6CF1: LDX    $77      ;Get the sprite number
6CF3: LDY    $6DD9,X  ;Get its index value into the descriptor table
6CF6: LDA    #0
6CF8: STA    $117E,Y  ;Zero the first byte
6CFB: INY
6CFC: LDX    #$03
6CFE: LSR    $114A,X
6D01: DEX
6D02: ROR    $114A,X
6D05: DEX
6D06: BPL    $6CFE
6D08: INX
6D09: LDA    $1149,X
6DOC: INY
6DOD: STA    $117E,Y
6D10: CPX    #$04
6D12: BNE    $6D08
6D14: LDA    #0
6D16: INY
6D17: STA    $117E,Y
6D1A: DEX
6D1B: BNE    $6D16
6D1D: PLA
6D1E: AND    #$0F
6D20: STA    $1174,Y
6D23: RTS
6D24: JSR    $8139
6D27: TAY
6D28: TXA
6D29: JSR    $9A77
6D2C: LDX    #$04

```



6D2E: JSR \$9D4A  
6D31: LDX #\$04  
6D33: CLC  
6D34: JSR \$9ACE  
6D37: STA \$1131,X  
6D3A: TYA  
6D3B: STA \$1132,X  
6D3E: INX  
6D3F: INX  
6D40: CPX #\$06  
6D42: BEQ \$6D34  
6D44: ROR \$116E  
6D47: BMI \$6D54  
6D49: STY \$1137  
6D4C: STA \$1138  
6D4F: LDX #\$04  
6D51: JSR \$9D4A  
6D54: LDA \$77  
6D56: TAX  
6D57: ASL  
6D58: TAY  
6D59: LDA \$1137  
6D5C: ASL \$116E  
6D5F: BCC \$6D6A  
6D61: CLC  
6D62: BPL \$6D67  
6D64: EOR #\$FF  
6D66: SEC  
6D67: ADC \$11D7,Y  
6D6A: SEI  
6D6B: STA \$11D7,Y  
6D6E: LDA \$1135  
6D71: ASL \$116E  
6D74: BPL \$6D88  
6D76: CLC  
6D77: ADC \$11D6,Y  
6D7A: STA \$11D6,Y  
6D7D: BCS \$6D82  
6D7F: INC \$1136  
6D82: LDA \$11E6  
6D85: JMP \$6D91  
6D88: STA \$11D6,Y  
6D8B: LDA \$11E6  
6D8E: ORA \$6CB3,X  
6D91: LSR \$1136  
6D94: BCS \$6D99  
6D96: EOR \$6CB3,X





```
* will set the carry flag. The character is returned in*
* the accumulator for which the test was performed. *
*
*****
```

```
6DC6: JSR $0386 ;Get the character TXTPTR is pointing at
6DC9: BEQ $6DD8 ;If there is none, then branch to exit
6DCB: CMP #',' ;Is it a comma?
6DCD: CLC ;Set the flag for comma found
6DCE: BEQ $6DD1 ;If it is a comma, then branch
6DD0: SEC ;Set the flag for a comma not found
6DD1: PHP ;Save the flag onto the stack
6DD2: PHA ;Save the character onto the stack
6DD3: JSR $0380 ;Move TXTPTR over to the next character
6DD6: PLA ;Get the character off of the stack
6DD7: PLP ;Get flag for comma found/not found off stack
6DD8: RTS ;And exit this routine
```

```
*****
*
* SPRITE DATA TABLE
*
* This table contains the pointers to the 11 byte
* sprite descriptors that are set up whenever a
* MOVSPR command with a speed rate is used. The
* descriptors are stored starting at location $117E.
*
*****
```

```
6DD9: .BYTE $00,$0B,$16
6DDC: .BYTE $21,$2C,$37
6DDF: .BYTE $42,$4D
```

```
*****
*
* BASIC PLAY COMMAND
*
* Command syntax: PLAY "Vn,On,Tn,Un,Xn,syms"
*
* NOTE: Vn = 'V' + the voice number
* On = 'O' + the octave number
* Tn = 'T' + the instrument
* Un = 'U' + the volume value
* Xn = 'X' + 1 (filter on)
* 'X' + 0 (filter off)
* syms = the symbols and notes of the music to be
* played
```

```

* This command makes it easier for you to PLAY a      *
* musical arrangement by allowing you to set the      *
* voice, octave, instrument, volume, and other      *
* parameters necessary in developing a musical score. *
*                                                    *
*****

```

```

6DE1: JSR   $877B      ;Get the length of the string in the accumulator
                        ;And place its address in locations $24, $25
6DE4: STA   $FF03      ;Enable BANK 14
6DE7: STA   $77        ;Save the length of the string
6DE9: JSR   $6FCE      ;Clear the SHARP and PITCH flags
6DEC: STA   $78        ;Zero the number of characters
6DEE: LDY   $78        ;Get the number of characters already 'played'
6DF0: CPY   $77        ;Compare it to the total number of characters
                        ;To be 'played'
6DF2: BEQ   $6E01      ;If finished, then exit
6DF4: JSR   $03B7      ;Get the character to be 'played'
6DF7: STA   $FF03      ;Enable BANK 14
6DFA: JSR   $6E02      ;Evaluate the play character
6DFD: INC   $78        ;Add one to the number of characters 'played'
6DFF: BNE   $6DEE      ;And branch to continue 'playing'
6E01: RTS                    ;Exit the routine

```

```

*****
*
* This is the main entry point of the PLAY command by *
* BASIC or MACHINE language, If the character in the *
* ACCUMULATOR is a space this subroutine will return, if*
* it's not then it will fall through to the following *
* routines to further evaluate the character.         *
*                                                    *
*****

```

```

6E02: CMP   #' '      ;If the character
6E04: BNE   $6E07      ;Is not a space, then branch
6E06: RTS                    ;Exit the routine

```

```

*****
*
* This routine checks the character to see if it is one*
* of the notes A - G, and if the character is a note, *
* then a JUMP to $6F1E is done to process that note.  *

```

```

* On entry to this routine, the accumulator contains *
* the ASCII letter which represents the note. If the *
* character is not a valid note, then this routine *
* will fall through to $6E12. *
* *
*****

```

```

6E07: CMP    #'a'      ;If the character is less than'a',
6E09: BCC    $6E12     ;Then check for control characters
6E0B: CMP    #'g'      ;If the character is greater than 'g',
6E0D: BCS    $6E12     ;Then check for control characters
6E0F: JMP    $6F1E     ;Process the note

```

```

*****
*
* This routine checks the accumulator for time duration*
* characters; w, h, q, i, s. If the character in the *
* accumulator is one of the time duration characters, *
* this routine does a JuMP $6F07 to process the dura- *
* tion. If the character is not one of the duration *
* characters, the routine will fall through to $6E1F. *
* *
*****

```

```

6E12: LDX    #4        ;Index to the table of duration characters
6E14: CMP    $6FE7,X   ;Is character one of the duration characters?
6E17: BNE    $6E1C     ;If not, then fall through to $6E1F
6E19: JMP    $6F07     ;Process the duration character
6E1C: DEX                ;Move index to next control character in table
6E1D: BPL    $6E14     ;Continue until 5 control characters tested

```

```

*****
*
* This routine checks the accumulator to see if it *
* contains the character 'R' which indicates a Rest *
* is to be performed. If the character in the *
* accumulator is an 'R', then the routine will JuMP *
* to $6F78 to process the rest. If the accumulator *
* does not contain an 'R', then the routine will check *
* check to see if the character is a '.', which *
* indicates a dotted note. If the character is a '.', *
* then the routine will JuMP to $6F03 to process the *
* dotted note. If the accumulator contains neither *
* character, then the routine will fall through to *
* $6E2D. *
* *
*****

```

```

6E1F:  CMP   #'r'           ;Is it the character for REST?
6E21:  BNE   $6E26         ;If not, then branch
6E23:  JMP   $6F78         ;Process a 'REST'
6E26:  CMP   #'.'           ;Is it the character for a 'dotted' note?
6E28:  BNE   $6E2D         ;If not, then branch
6E2A:  JMP   $6F03         ;Process a 'dotted' note

```

```

*****
*
*   This routine checks the accumulator to see if it
*   contains play options: V, O, T, X, U, or M. If the
*   accumulator contains one of the play options, this
*   routine will JuMP to $6F52 to process it. If the ac-
*   cumulator does not contain any of the play options,
*   then the routine will fall through to $6E3A.
*
*****

```

```

6E2D:  LDX   #$05           ;Index to the table of control characters
6E2F:  CMP   $6FEC,X        ;Check to see if character is in the accumulator
6E32:  BNE   $6E37           ;Is the same as one of the control characters?
                          ;If it is not, then branch
6E34:  JMP   $6F52         ;If it is one of the characters, then branch
6E37:  DEX
6E38:  BPL   $6E2F         ;Then fall through to $6E3A

```

```

*****
*
*   This routine checks the accumulator to see if it
*   contains the flag for a SHARP (#) or a Flat ($).
*   If the accumulator contains either of these charac-
*   ters, this routine will JuMP to $6F69 for a SHARP,
*   or to $6F6C for a FLAT. If neither of the characters
*   is found, then the routine will branch to $6E48.
*
*****

```

```

6E3A:  CMP   #'#'          ;Is it the flag for SHARP?
6E3C:  BNE   $6E41         ;If it is not, then branch
6E3E:  JMP   $6F69         ;If it is the flag for a SHARP, then jump to
                          ;Set the flag for SHARP
6E41:  CMP   #'$'          ;Is it the flag for FLAT?
6E43:  BNE   $6E48         ;If it is not, then branch
6E45:  JMP   $6F6C         ;If it is the flag for a FLAT, then set
                          ;The flag for FLAT

```

```
*****
*
* This routine checks to see if the character in the
* accumulator is an ASCII digit 0 - 9, and if not, it
* will terminate the Play Routine with an 'ILLEGAL
* QUANTITY' error message. If the character in the
* accumulator is an ASCII digit, then this routine
* will check FLAG to see if the previous character
* was the character for the VOICE, OCTAVE, TONE (TUNE),
* FILTER ON/OFF (X), or VOLUME options. If the previous
* character was for one of these options, this routine
* passes the numeric value onto the proper routine
* below that handles the options.If the previous char-
* acter was not for one of these options, then an
* 'ILLEGAL QUANTITY' error message will be generated.
*
*****
```

```
6E48: SEC          ;Subtract 48 from the ASCII
6E49: SBC  #'0'    ;Character to turn it into a hex digit
6E4B: CMP  #10     ;Is it greater than 9?
6E4D: BCC  $6E52   ;No, then branch
6E4F: JMP  $6EFD   ;If it is greater than 9, then generate
                    ;An 'ILLEGAL QUANTITY' error
6E52: ASL  $0126   ;Rotate bit 7 into the carry flag to
6E55: BCS  $6E9D   ;Test for the VOICE parameter in this value
6E57: ASL  $0126   ;If not, then check to see if it is
6E5A: BCS  $6EA8   ;For OCTAVE
6E5C: ASL  $0126   ;If not, then check to see if this value is
6E5F: BCS  $6EB1   ;For TONE
6E61: ASL  $0126   ;If not, then check for 'X' (FILTER)
6E64: BCC  $6EDD   ;If it is not for FILTER, then branch
```

```
*****
*
* This routine is called when the 'X' option is used.
* This routine will first check to ensure the value
* after the 'X' is less than 2 and if it is not, then
* an 'ILLEGAL QUANTITY' error message is generated.
* If the value is less than 2, then this routine will
* turn ON/OFF the filter for the current active VOICE
* (X1 turns the filter ON, X1 turns the filter OFF).
*
*****
```

```
6E66: CMP  #2      ;Is the value less than 2?
6E68: BCC  $6E6D   ;If it is, then branch
```



```

6E6A: JMP    $6EFD      ;If the value is not less than 2,
                        ;Then generate an 'ILLEGAL QUANTITY' error
6E6D: LSR
                        ;If the value was ON, then set the carry flag
6E6E: LDY    $122F     ;Get the current VOICE number being used - 1
6E71: LDX    $6FE4,Y   ;Get the index pointer to that VOICE's MSB for
                        ;Its countdown timer (N -TIME)
6E74: LDA    $1224,X   ;Wait until that VOICE (N-TIME) has counted
6E77: BPL    $6E74     ;Down and the Basic IRQ sets it back to $F0
6E79: LDA    $6CB3,Y   ;Get bit value to turn on this VOICE's filter
6E7C: ORA    $1273     ;And set that bit in the shadow register
                        ;For $D417 (SIDREG23)
6E7F: BCS    $6E84     ;If the carry is set from above, then
                        ;Turn on the filter
6E81: EOR    $6CB3,Y   ;If it is not set, then turn the filter off
6E84: STA    $1273     ;And save the new filter configuration
6E87: LDA    $1274     ;Copy MODE/VOLUME shadow register for $D418 in
                        ;SIDREG24
6E8A: STA    $1275     ;And save it
6E8D: LDX    #3        ;Copy the 4
6E8F: LDA    $1271,X   ;Shadow registers
6E92: JSR    $A845     ;Enable BANK 15
6E95: STA    $D415,X   ;Into the SID chip registers
6E98: DEX
                        ;To set up the filter values, Resonance/Filter
6E99: BPL    $6E8F     ;And Mode/Volume
6E9B: BMI    $6EF7     ;Unconditional branch to exit
    
```

```

*****
*
*                               *
*                               *
*                               *
*                               *
*   If the value after the 'V' is greater than 3, then an*
*   'ILLEGAL QUANTITY' error message is generated. If the*
*   value after the 'V' is less than or equal to 3, then *
*   this value - 1 is placed into location $122F (VOICE).*
*
*
*****
    
```

```

6E9D: TAX                ;Move the numeric number into the X register
6E9E: DEX                ;VOICE number = (VOICE number - 1)
6E9F: CPX    #03        ;Has programmer requested a VOICE number greater
6EA1: BCS    $6EFD     ;Than 2?If so, generate 'ILLEGAL QUANTITY' error
6EA3: STX    $122F     ;Save value as current OCTAVE number in OCTAVE
6EA6: BCC    $6EF7     ;Unconditional branch to exit
    
```

```

*****
*
*                               OCTAVE1
*
* If the value after the 'O' is greater than 6, then an*
* 'ILLEGAL QUANTITY' error message is generated. If the*
* value after the 'O' is less than or equal to 6, then *
* this value is saved into location $122B (OCTAVE).    *
*
*****

```

```

6EAB:  CMP    #7           ;Programmer requested an OCTAVE greater than 6?
6EAA:  BCS    $6EFD        ;If so, generate an 'ILLEGAL QUNATITY' error
6EAC:  STA    $122B       ;Save value as current OCTAVE number in OCTAVE
6EAF:  BCC    $6EF7       ;Unconditional branch to exit

```

```

*****
*
*                               TONE1
*
* This routine takes the value after the 'T' and uses *
* this value to point to the proper ENVELOPE. If 'T' *
* is not used, the system defaults to ENVELOPE zero. *
*
*****

```

```

6EB1:  TAX           ;Move TONE (TUNE) value into the X register to
           ;Be used as index value to the proper ENVELOPE
6EB2:  JSR    $A845     ;Enable BANK 15
6EB5:  LDY    $122F     ;Get the VOICE number 0 - 2 to be used as an
           ;Index value
6EB8:  LDA    $1253,X   ;Get waveform value for this TONE from WAVTAB
6EBB:  STA    $1230,Y   ;And save it in WAVE 0
6EBE:  LDA    $7039,Y   ;Get the index value into the ATtack TABLE
6EC1:  TAY           ;Move the index value into the Y register
6EC2:  LDA    $123F,X   ;Get this tone's attack rate
6EC5:  STA    $D405,Y   ;And pass it onto the SID chip
6EC8:  LDA    $1249,X   ;Give the sustain/release value
6ECB:  STA    $D406,Y   ;To the SID chip
6ECE:  LDA    $125D,X   ;Give the pulse width low
6ED1:  STA    $D402,Y   ;To the SID chip
6ED4:  LDA    $1267,X   ;And the pulse width high
6ED7:  STA    $D403,Y   ;To the SID chip
6EDA:  JMP    $6EF7     ;Then exit this routine

```

```

*****
*
*                               VOLUME1
*
* This routine takes the value after the 'U' and uses
* that value to set the NEW VOLUME in locations $1274,
* $1275 which contain the MODE/VOLUME values with vol-
* ume being the least significant Nybble (Bits 3-0).
*
*
*           'U' value      Volume
*           0                0
*           1                1
*           2                3
*           3                5
*           4                7
*           5                8
*           6               12
*           7               13
*           8               14
*           9               15
*
*****

```

```

6EDD: TAX                ;Move the volume value into the X register
6EDE: LDA $1274          ;Get the MODE/VOLUME
6EE1: AND #$F0           ;Drop off the current volume level
6EE3: ORA $703C,X       ;Add in the requested volume level
6EE6: STA $1274          ;And save it as the NEW MODE/VOLUME
6EE9: LDA $1275          ;Repeat it
6EEC: AND #$F0           ;Again for its
6EEE: ORA $703C,X       ;Duplicate Register
6EF1: JSR $A845          ;Enable BANK 15
6EF4: STA $D418          ;Give it to the SID chip as the NEW
                        ;MODE/VOLUME LEVEL

```

```

-----*
*
*                               CLRFLG
*
*                               CLear FLaG
*
* This routine is to clear the flags for VOICE,
* OCTAVE, TONE, FILTER, and VOLUME to prepare the PLAY
* routines for the next note. The routine then
* returns to the calling routine ($6DFA or $6EFD).
*
*-----*

```

```

6EF7: LDA    #0          ;Set FLAG
6EF9: STA    $0126       ;To zero
6EFC: RTS                      ;And exit
    
```

```

*****
*
*                               PLYERR
*
*                               PLAY ERror
*
* This routine is used by the PLAY Command to generate
* an 'ILLEGAL QUANTITY' error message and to clear
* FLAG to clear out any previous FLAGS.
*
*****
    
```

```

6EFD: JSR    $6EF7       ;Prepare FLAG for the next play character
6F00: JMP    $7D28       ;Generate an 'ILLEGAL QUANTITY' error
    
```

```

*****
*
*                               SETDN
*
*                               SET Dotted Note
*
* This routine will store the value ($2E) to location
* $1233 to indicate that a dotted note is to be
* played. This value will, in effect, increase the
* length the note is to be played by 50%.
*
*****
    
```

```

6F03: STA    $1233       ;Save the value for a dotted note ($2E)
6F06: RTS                      ;Exit the routine
    
```

```

*****
*
*                               SNTIME
*
*                               Set Note TIME
*
* This routine sets up the length of the time each
* note is to be played by using the index register (X)
* to indicate which value is to be defined.
*
*****
    
```

		N-Time	Length	
	<u>INDEX</u>	<u>VALUE</u>	<u>(\$1229,\$122A)</u>	<u>in 4/4 Time</u>
*	0	W(whole)	\$0480	4/4
*	1	H(half)	\$0240	2
*	2	Q(quarter)	\$0120	1
*	3	I(eighth)	\$0090	1/2
*	4	S(sixteenth)	\$0048	1/4
*				

\*\*\*\*\*

```

6F07: LDY    #$80      ;Set up N-Time
6F09: STY    $1229    ;To the Maximum
6F0C: LDY    #$04      ;Time allowed
6F0E: STY    $122A    ;Which is for a whole note
6F11: DEX                ;Index = (Index - 1)
6F12: BMI    $6F1D    ;If the index equals a whole note, then branch
6F14: LSR    $122A    ;Calculate the new
6F17: ROR    $1229    ;N-Time value per index value
6F1A: JMP    $6F11    ;Continue until X = $FF
6F1D: RTS                ;Exit this routine
    
```

\*\*\*\*\*

\*  
 \* Play NOTES A - G \*  
 \*

\*\*\*\*\*

```

6F1E: SEC                ;Set the carry for subtraction
6F1F: SBC    #'a'      ;Subtract value for 'a' to create index value
6F21: TAX                ;Move the index value into the X register
6F22: LDA    $6FF2,X   ;Get the value for the note
6F25: TAX                ;Save it into the X register
6F26: LDA    #6        ;Get the maximum number of octaves
6F28: SEC                ;Set the carry for subtraction
6F29: SBC    $122B    ;Subtract number of octaves requested (Default
                        ;=4) from the maximum allowable octaves
6F2C: TAY                ;Save into the Y register as the
                        ;Actual octave to be played
6F2D: TXA                ;Move the note value back into the accumulator
6F2E: CLC                ;Clear the carry for addition
6F2F: ADC    $122C    ;Add SHARP flag to the note value
                        ;($01 = SHARP, $FF = Flat, $00 = Natural)
6F32: BPL    $6F37    ;If note is SHARP or natural, then branch
6F34: LDA    #11
6F36: INY                ;Add one to the octave
6F37: CMP    #12
6F39: BCC    $6F3E
    
```

```

6F3B: LDA    #0
6F3D: DEY                ;Subtract one from the octave
6F3E: TAX                ;Move note value into X register
6F3F: LDA    $6FF9,X
6F42: STA    $122D
6F45: LDA    $7005,X
6F48: DEY
6F49: BMI    $6F72
6F4B: LSR
6F4C: ROR    $122D
6F4F: JMP    $6F48
    
```

```

*****
*
*                               *
*               SETFLAG          *
*
*               SET command bit in FLAG          *
*
* This routine will check to see if the value in the *
* ACCUMULATOR is the value for 'm' which indicates to *
* wait for the end of this measure, and if it is then *
* it will branch to the routine below. If it is not then*
* it uses the value in the X - register to set a BIT in *
* FLAG ($0126) to indicate what command character(s) are*
* in use. This routine is called by $6E2D.          *
*
*
*****
    
```

```

6F52: CMP    #'m'          ;Is it the control character 'M'?
6F54: BEQ    $6F5D          ;If it is, then branch
6F56: LDA    $9D1C,X        ;Get the proper bit value as a flag (SEE $9D1C)
6F59: STA    $0126          ;And place it in FLAG
6F5C: RTS                    ;Exit this routine
    
```

```

*****
*
*                               *
*               M                *
*
*               Measure           *
*
* This routine will check each of the three voices to *
* ensure they have completed playing their note, and if *
* they have not it will wait for them finish.          *
*
*
*****
    
```

```

6F5D: LDY #5 ;Pointer to voice 3
6F5F: LDA $1223,Y ;Wait until the
6F62: BPL $6F5F ;Current measure is finished
6F64: DEY ;Move to the
6F65: DEY ;Next voice
6F66: BPL $6F5F ;Continue until all 3 voices have finished
6F68: RTS ;And exit the routine

```

```

*****
*
* SHARP
*
* This routine sets the sharp flag in SHARP ($122C).
*
*****

```

```

6F69: LDA #1 ;Flag for SHARP
6F6B: .BYTE $2C ;MASK to fall through to $6F6E

```

```

*****
*
* FLAT
*
* This routine sets the flag to PLAY a flat in SHARP.
*
*****

```

```

6F6C: LDA #$FF ;Flag for a FLAT
6F6E: STA $122C ;Save it
6F71: RTS ;And exit the routine
6F72: STA $122E ;Save the MSB for the frequency of this note
6F75: LDA #0 ;Set the flag to play the note
6F77: .BYTE $2C ;Then fall through to $6F7A

```

```

*****
*
* REST
*
* This routine will set the flag to perform a REST.
*
*****

```

```

6F78: LDA #$FF ;Set the flag to perform a REST
6F7A: PHA ;Save the flag onto the stack
6F7B: LDX $122F ;Get the voice number for this note
6F7E: LDY $6FE4,X ;And get its Index value
6F81: LDA $1224,Y ;And wait for the previous

```

```

6F84: BPL $6F81 ;Note to be played
6F86: LDA $1229 ;Get the duration the note is
6F89: STA $1223,Y ;To be played out of
6F8C: LDA $122A ;Note TIME and place it
6F8F: STA $1224,Y ;Into the proper VOICE address
6F92: LDA $1233 ;If a dotted note is not to
6F95: BEQ $6FAE ;Be played, then branch
6F97: LDA $122A ;Get the current
6F9A: LSR ;Duration this note
6F9B: PHA ;Is to be played
6F9C: LDA $1229 ;(System default is $0120 which
6F9F: ROR ;Is the length of
6FA0: CLC ;A quarter note in 4/4 time)
6FA1: ADC $1223,Y ;Divide that number by two
6FA4: STA $1223,Y ;And add that value to
6FA7: PLA ;VOICE which is used as
6FA8: ADC $1224,Y ;The length this voice is to play
6FAB: STA $1224,Y ;This note VOICE = N-TIME/2 + N-TIME
6FAE: PLA ;Get flag to play or perform a REST. If the
6FAF: BMI $6FCE ;flag to a REST, then branch to clear SHARP/
;FLAT & DOTTED flags--prepare for the next note

6FB1: JSR $A845 ;Enable BANK 15
6FB4: LDY $7039,X ;Get the index value
6FB7: LDA $122D ;Get the frequency for
6FBA: STA $D400,Y ;This note and give it
6FBD: LDA $122E ;To the SID chip for the
6FC0: STA $D401,Y ;VOICE indicated by the index register
6FC3: LDA #$08 ;Turn off the voice that is specified
6FC5: STA $D404,Y ;By the index value ($00 = 1, $07 = 2, $0E = 3)
6FC8: LDA $1230,X ;Then pass the waveform for this
6FCB: STA $D404,Y ;Note onto the SID chip
6FCE: LDA #0 ;Set the flag to
6FD0: STA $122C ;Play a natural note.
6FD3: STA $1233 ;And that there is no dotted note to be played
6FD6: RTS ;Exit this routine

```

```

*****
*
*          BASIC TEMPO COMMAND          *
*
* Command syntax: TEMPO x              *
*
* This command lets you determine the speed at which a *
* song will be played. The value of the TEMPO speed de-*
* fined by 'x' can be between 0 and 255 (default=8).  *
*
*****

```



```

6FD7: JSR   $87F4   ;Get the tempo rate into the X register
6FDA: TXA                   ;Move the tempo rate to the accumulator
6FDB: BEQ   $6FE1   ;If the tempo rate is zero, then generate
                        ;An 'ILLEGAL QUANTITY' error
6FDD: STX   $1222   ;Save it as the new tempo rate
6FE0: RTS                   ;And exit the routine
6FE1: JMP   $7D28   ;Generate an 'ILLEGAL QUANTITY' error
    
```

```

*****
*
* The following table is used as index values to each *
* voice, and is used by the BASIC PLAY command.      *
*
*****
    
```

		<u>VALUE</u>	<u>VOICE</u>	<u>VOICES</u>
6FE4:	.BYTE \$00	; 0	1	\$1223, \$1224
6FE5:	.BYTE \$02	; 2	2	\$1225, \$1226
6FE6:	.BYTE \$04	; 4	3	\$1227, \$1228

```

*****
*
* The following table is used to indicate the duration *
* of a note to be played according to the note's type. *
*
*****
    
```

		<u>DESCRIPTION</u>	<u>BEAT IN 4:4 TIME</u>
6FE7:	TXT 'w'	;Whole note	4/4
6FE8:	TXT 'h'	;Half note	2
6FE9:	TXT 'q'	;Quarter note	1
6FEA:	TXT 'i'	;Eighth note	1/2
6FEB:	TXT 's'	;Sixteenth note	1/4

```

*****
*
* These are the control characters that can be used in *
* a PLAY statement to select various parameters.      *
*
*****
    
```

		<u>DESCRIPTION</u>
6FEC:	TXT 'v'	;Voice
6FED:	TXT 'o'	;Octave
6FEE:	TXT 't'	;Tone (tune) envelope defaults
6FEF:	TXT 'x'	;Filter ON/OFF

```

6FF0: TXT 'u'           ;Volume
6FF1: TXT 'm'           ;M - wait for the end of this measure

```

```

*****
*
* This table contains the values for each of the
* Notes A - G and is used by the routine at $6F1E.
*
*****

```

	<u>VALUE</u>	<u>NOTE</u>
6FF2:	.BYTE \$09	; A
6FF3:	.BYTE \$0B	; B
6FF4:	.BYTE \$00	; C
6FF5:	.BYTE \$02	; D
6FF6:	.BYTE \$04	; E
6FF7:	.BYTE \$05	; F
6FF8:	.BYTE \$0D	; G
6FF9:	.BYTE \$07	
6FFA:	.BYTE \$2F,\$B6,\$83	
6FFD:	.BYTE \$99,\$FC,\$B1	
7000:	.BYTE \$25,\$EF,\$20	
7003:	.BYTE \$BE,\$D1,\$4C	
7006:	.BYTE \$50,\$55	
7008:	.BYTE \$5A,\$5F,\$65	
700B:	.BYTE \$6B,\$72,\$78	
700E:	.BYTE \$80,\$87,\$8F	

```

*****
*
* The following values are the default Attack/Decay
* rates for the ten instrument sounds. The Attack and
* Decay rates are two four bit values that are packed
* together into one byte. The Attack rate occupies
* bits 7 - 4 and the Decay rate bits 3 - 0. For
* example, the first value in the table is 9 which is
* 0000 - Attack rate of 0 and 1001 - Decay rate of 9.
*
*****

```

```

7011: .BYTE $09,$C0,$00
7014: .BYTE $05,$94,$09
7017: .BYTE $09,$09,$89
701A: .BYTE $09

```

```

*****
*
* The following values are the default Sustain/Release *
* rates for the ten instrument sounds. The values are *
* stored in the same format as the Attack/Decay *
* rates listed above. *
*
*****

```

```

701B: .BYTE $00,$C0,$F0
701E: .BYTE $50,$40,$21
7021: .BYTE $00,$90,$41
7024: .BYTE $00

```

```

*****
*
* The following values are the default waveforms for *
* the ten instrument sounds ( $10 = Triangle, *
* $20 = Sawtooth, $40 = Pulse, $80 = Noise ). The *
* '1' that is added to each of these values is the *
* GATE bit. When this bit is stored along with the *
* waveform value, the combined value turns the *
* selected sound on. For instance, to turn the *
* Sawtooth waveform on, the value of $21 is stored *
* in the proper memory location ($20 for the Sawtooth *
* waveform plus one for the GATE bit = $21). *
*
*****

```

```

7025: .BYTE $41,$21,$11
7028: .BYTE $81,$11,$21
702B: .BYTE $41,$41,$41
702E: .BYTE $11

```

```

*****
*
* The following values are the MSBs of the default *
* pulse width values for the ten instrument sounds. *
* The LSB of each of these values is $00. *
*
*****

```

```

702F: .BYTE $06,$00,$00
7032: .BYTE $00,$00,$00
7035: .BYTE $02,$08,$02
7038: .BYTE $00

```

```

*****
*
* The following values are used as the index to the
* voices in the Sound Interface Device (SID).
*
*****

```

```
7039: .BYTE $00,$07,$0E
```

```

*****
*
* The following values are the volume values that are
* used when the control character 'U' is used in the
* 'PLAY' command. For example, the statement
* PLAY 'U7 A' plays the note 'A' with a volume of 12.
*
*****

```

```
703C: .BYTE $00,$01,$03
```

```
703F: .BYTE $05,$07,$08
```

```
7042: .BYTE $0A,$0C,$0E
```

```
7045: .BYTE $0F
```

```

*****
*
*
* BASIC FILTER COMMAND
*
* Command syntax: FILTER [frq][,lwp][,bdp][,hip][,res]
*
* NOTE: frq = the filter cutoff frequency (0 - 2047)
*        lwp = the low-pass filter (1 = on, 0 = off)
*        bdp = the band-pass filter (1 = on, 0 = off)
*        hip = the high-pass filter (1 = on, 0 = off)
*        res = the resonance value (0 - 15)
*
* This command allows you to set the filter
* characteristics of the SID chip. A filter allows some
* frequencies to pass while others are suppressed.
*
* The low-pass filter allows frequencies below the
* cutoff frequency to pass, while suppressing those
* above the cutoff frequency.
*
* The band-pass filter allows frequencies within a
* specified range to pass while suppressing those
* above and below the specified range.
*

```

```

*   The high-pass filter allows frequencies above the      *
*   cutoff frequency to pass while suppressing those      *
*   below the cutoff frequency.                            *
*                                                         *
*****

```

```

7046: PHA                ;Save the first character after the command
                        ;Onto the stack
7047: LDY    #$03        ;Set index to number of bytes to transfer - 1
7049: LDA    $1271,Y    ;Save the current
704C: STA    $1234,Y    ;Filter values
704F: DEY                ;Temporarily
7050: BPL    $7049
7052: PLA                ;Restore the accumulator
7053: CMP    #$2C        ;If no cutoff frequency is specified,
7055: BEQ    $7070        ;Then branch to get the second parameter
7057: JSR    $8812        ;Get the cutoff frequency in the Y register and
                        ;The accumulator
705A: CMP    #$08        ;If the cutoff frequency is greater than 2047,
705C: BCS    $70BE        ;Then generate an 'ILLEGAL QUANTITY' error
705E: STY    $1234        ;Save the LSB
7061: STY    $1235        ;Of the Filter cutoff frequency
7064: LSR                ;Adjust the range
7065: ROR    $1235        ;So the LSB and MSB of
7068: LSR                ;Filter 2047 equals $FFFF
7069: ROR    $1235        ;And Filter 1
706C: LSR                ;Equals $0001
706D: ROR    $1235
7070: LDA    #$10        ;Starting bit location
7072: STA    $1238        ;For the Filter switches
7075: JSR    $9E1E        ;Get the value for the low-pass Filter on or off
7078: BCC    $7091        ;If no value was specified, then branch
707A: CPX    #$01        ;If the value specified
707C: BCC    $7083        ;Is not zero, then branch
707E: BEQ    $7083        ;If the value is not one, then branch
7080: JMP    $7D28        ;Generate an 'ILLEGAL QUANTITY' error
7083: LDA    $1237        ;Get the packed Filter values
7086: ORA    $1238        ;Set the proper bit for Filter type 'on'
7089: BCS    $708E        ;If the Filter was selected, then branch
708B: EOR    $1238        ;If it was not, then clear the bit to turn
                        ;The Filter 'off'
708E: STA    $1237        ;Save the value back
7091: ASL    $1238        ;Shift the counter to the next Filter type
7094: BPL    $7075        ;Continue
7096: JSR    $9E1E        ;Get the resonance value, if any, into X reg.
7099: BCC    $70B2        ;If no value was specified, then branch
709B: CPX    #$10        ;If the resonance value is

```

```

709D: BCS    $70BE    ;Greater than 15, then generate
                          ;An 'ILLEGAL QUANTITY' error
709F: TXA
70A0: ASL
70A1: ASL    ;To the upper nybble (bits 0-3) to (bits 7-4)
70A2: ASL
70A3: ASL
70A4: STA    $1239    ;Save it
70A7: LDA    $1236    ;Get the packed Resonance/Voice input register
70AA: AND    #$0F     ;Drop the old resonance value
70AC: ORA    $1239    ;Replace it with the new value
70AF: STA    $1236    ;Save it back
70B2: LDY    #$03     ;Set index to number of bytes to transfer - 1
70B4: LDA    $1234,Y  ;Move the
70B7: STA    $1271,Y  ;New filter
70BA: DEY
70BB: BPL    $70B4    ;Back
70BD: RTS
70BE: JMP    $7D28    ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
*          BASIC ENVELOPE COMMAND
*
* Command syntax:  ENVELOPE x[,atk][,dec][,sus][,rel]
*                  [,wve][,psw]
*
* NOTE:    x = envelope number
*          atk = attack rate (0 - 15)
*          dec = decay rate (0 - 15)
*          sus = sustain value (0 - 15)
*          rel = release rate (0 - 15)
*          wve = waveform value
*
*          where:  0 = triangle
*                  1 = sawtooth
*                  2 = variable pulse width (square)
*                  3 = noise
*                  4 = ring modulation
*
*          psw = pulse width (0 - 4095)
*
* This command allows you to set up the waveform
* characteristics of a sound that is to be produced.
* There are ten predefined instrument sounds in the
* 128. They are; piano, accordion, calliope, drum,
* flute, guitar, harpsichord, organ, trumpet, and
*

```

```

* xylophone. These instruments correspond to envelope *
* numbers 0 - 9 respectively. If you wish to use one *
* of the predefined instruments, just use the envelope *
* number without specifying any other parameters. *
* *
*****

```

```

70C1: JSR $87F4 ;Get the envelope number into the X register
70C4: CPX #10 ;If the value is between
70C6: BCC $70CB ;0 and 9, then continue
70C8: JMP $7D28 ;If not, generate an 'ILLEGAL QUANTITY' error
70CB: STX $123A ;Save the envelope number
70CE: LDA $123F,X ;Using envelope number as an index, get the old
;Attack/Decay rate
70D1: STA $123B ;Save it as a default value
70D4: LDA $1249,X ;Get the old Sustain/Release rate
70D7: STA $123C ;Save it as a default value
70DA: LDA $1253,X ;Get the old waveform value
70DD: STA $123D ;Save it
70E0: LDX #$00 ;Set the initial index to zero
70E2: STX $123E ;For Attack/Decay values
70E5: JSR $9E1E ;Get the Attack or Sustain value into the X reg.
70E8: BCC $7100 ;If no value was specified, then branch
70EA: TXA ;Move the value into the accumulator
70EB: ASL ;Shift the value
70EC: ASL ;4 times
70ED: ASL ;To place the Attack or Sustain rate
70EE: ASL ;Into the high nybble of the byte
70EF: STA $1239 ;Save it
70F2: LDX $123E ;Get the index to the appropriate register
70F5: LDA $123B,X ;Get old Attack/Decay or Sustain/Release value
70F8: AND #$0F ;Drop the old Attack or Sustain value
70FA: ORA $1239 ;Replace it with the new value
70FD: STA $123B,X ;Save it back
7100: JSR $9E1E ;Get the Decay or Release rate, if any,
;Into the X register
7103: BCC $7119 ;If no value was specified, then branch
7105: TXA ;Move the value into the Accumulator
7106: AND #$0F ;Drop bits 7 - 4
7108: STA $1239 ;Save it temporarily
710B: LDX $123E ;Get the index to the proper storage area
710E: LDA $123B,X ;Get old Attack/Decay or Sustain/Release value
7111: AND #$F0 ;Drop the old Decay or Release value
7113: ORA $1239 ;Replace it with the new value and
7116: STA $123B,X ;Save the new Attack/Decay or
;Sustain/Release value
7119: LDX $123E ;Get the current index

```

```

711C: INX                ;Add one
711D: CPX    #$01        ;If the index equals 1,
711F: BEQ    $70E2       ;Then branch back to set Sustain/Release values
7121: JSR    $9E1E       ;Get the waveform value, if any, into the X reg.
7124: BCC    $7136       ;If no value was specified, then branch
7126: LDA    #$15        ;Set the initial value to ring modulation
7128: CPX    #$04        ;If the waveform specified is 4
712A: BEQ    $7133       ;For ring modulation, then branch
712C: BCS    $70C8       ;If the value is greater than 4, then generate
                          ;An 'ILLEGAL QUANTITY' error
712E: LDA    $6CB7,X     ;Get appropriate value for specified waveform
7131: ORA    #$01        ;And add one to set the GATE bit
7133: STA    $123D       ;Save the waveform
7136: JSR    $9E06       ;Get the pulse width, if any, into the Y
                          ;Register and the accumulator
7139: BCC    $714E       ;If no value was specified, then branch
713B: TAX                ;Move the MSB of the pulse width into the X reg.
713C: LDA    $123D       ;Get the waveform value
713F: AND    #$40        ;Check if the pulse waveform was selected
7141: BEQ    $714E       ;If not, then branch
7143: TXA                ;Move the MSB back into the accumulator
7144: LDX    $123A       ;Get the envelope number
7147: STA    $1267,X     ;Save the MSB of the pulse width
714A: TYA                ;Move the LSB of the pulse width into the acc.
714B: STA    $125D,X     ;And save the LSB of the pulse width
714E: LDX    $123A       ;Use the envelope number as an index
7151: LDA    $123B       ;Get the packed Attack/Decay rate
7154: STA    $123F,X     ;And save it
7157: LDA    $123C       ;Get the packed Sustain/Release rate
715A: STA    $1249,X     ;And save it
715D: LDA    $123D       ;Get the waveform,
7160: STA    $1253,X     ;Save it,
7163: RTS                ;And exit

```

```

*****
*
*          BASIC COLLISION COMMAND          *
*
* Command syntax: COLLISION coltyp [,action] *
*
* NOTE: coltyp = the collision type         *
*
*       where: 1 = sprite to sprite        *
*              2 = sprite to display data  *
*              3 = light pen               *
*

```



```

*          action = the line number for the action to      *
*          be taken when the collision type                *
*          specified by 'coltyp' occurs                   *
*
* This command allows you to dictate what 'action'      *
* will be taken when the collision specified by          *
* 'coltyp' occurs.                                     *
*
*****
7164: JSR  $87F4      ;Get the collision type into the X register
7167: DEX                    ;If the collision type
7168: CPX  #$03      ;Value is greater than 2,
716A: BCS  $718D     ;Then generate an 'ILLEGAL QUANTITY' error
716C: STX  $1280     ;Save the collision type
716F: JSR  $9E06     ;Get the Line number to GOSUB to, if any,
                    ;Into the Y register and the accumulator
7172: PHP                    ;Save the status register
                    ;(Carry set = a line number was specified)
7173: LDX  $1280     ;Use the collision type as an index
7176: STA  $127C,X   ;Save the LSB of the line number
7179: TYA                    ;Move the MSB of the line number into the acc.
717A: STA  $1279,X   ;Save the MSB of the line number to GOSUB
717D: LDA  $127F     ;Get the interrupt status
7180: ORA  $6CB3,X   ;Set the proper bit for the specified interrupt
7183: PLP                    ;Restore the status register
7184: BCS  $7189     ;If a line number was specified, then branch
7186: EOR  $6CB3,X   ;If not, turn off the specified interrupt check
7189: STA  $127F     ;Save the interrupt value
718C: RTS                    ;And exit
718D: JMP  $7D28     ;Generate an 'ILLEGAL QUANTITY' error

*****
*
*          BASIC SPRCOLOR COMMAND                        *
*
* Command syntax:  SPRCOLOR [smc-1][,smc-2]           *
*
* NOTE:  smc-1 = the Multicolor 1 for all the sprites *
*        smc-2 = the Multicolor 2 for all the sprites *
*
* This command allows you to set the Multicolor 1 and *
* Multicolor 2 colors for all of the sprites.        *
*
*****

```

```

7190: CMP   #$2C      ;If there is no first parameter,
7192: BEQ   $71A2     ;Then branch to get the second parameter
7194: JSR   $87F4     ;Get the first parameter into the X register
7197: DEX                   ;If it is greater
7198: CPX   #$10      ;Than 16,
719A: BCS   $71B3     ;Then generate an 'ILLEGAL QUANTITY' error
719C: JSR   $A845     ;Enable BANK 15
719F: STX   $D025     ;Set the Multicolor 1 value
71A2: JSR   $9E1E     ;Get the second parameter into the X register
71A5: BCC   $71B2     ;If there is no second parameter, then exit
71A7: DEX                   ;If the second parameter is
71A8: CPX   #$10      ;Greater than 16,
71AA: BCS   $71B3     ;Then generate an 'ILLEGAL QUANTITY' error
71AC: JSR   $A845     ;Enable BANK 15
71AF: STX   $D026     ;And set the Multicolor 2 value
71B2: RTS                   ;Exit
71B3: JMP   $7D28     ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
*          BASIC WIDTH COMMAND          *
*
* Command syntax:  WIDTH x              *
*
* NOTE:  x = the line width value      *
*
*          where: 1 = a single width line *
*                  2 = a double width line *
*
* This command allows you to set the width of the *
* lines that are drawn by the BASIC graphics commands. *
*
*****

```

```

71B6: JSR   $87F4     ;Get the width value into the X register
71B9: DEX                   ;If the value is
71BA: CPX   #$02      ;Greater than 2,
71BC: BCS   $71C2     ;Then generate an 'ILLEGAL QUANTITY' error
71BE: STX   $116B     ;Save the width value
71C1: RTS                   ;And exit
71C2: JMP   $7D28     ;Generate an 'ILLEGAL QUANTITY' error

```

```
*****
*
*                               *
*          BASIC VOLume COMMAND *
*                               *
* Command syntax:  VOL  volval  *
*                               *
* NOTE:  volval = the sound volume value (0 - 15) *
*                               *
* This command allows you to set the volume of the SID *
* chip to minimum (0), maximum (15), or any level in *
* between (0 - 15). *
*                               *
*****
```

```
71C5: JSR  $87F4      ;Get the Volume value into the X register
71C8: CPX  #$10      ;If it is greater than 16,
71CA: BCS  $71E9      ;Then generate an 'ILLEGAL QUANTITY' error
71CC: STX  $77        ;Save the Volume value
71CE: LDA  $1274      ;Get the old Volume value
71D1: AND  #$F0       ;Clear the old Volume
71D3: ORA  $77        ;Replace it with the new value
71D5: STA  $1274      ;Save it back
71D8: LDA  $1275      ;Get the old Volume value
71DB: AND  #$F0       ;Clear the old value
71DD: ORA  $77        ;Replace it with the new Volume value
71DF: STA  $1275      ;Save it
71E2: JSR  $A845      ;Enable BANK 15
71E5: STA  $D418      ;Set the SID chip to the specified Volume
71E8: RTS                ;And exit
71E9: JMP  $7D28      ;Generate an 'ILLEGAL QUANTITY' error
```

```
*****
*
*                               *
*          BASIC SOUND COMMAND *
*                               *
* Command syntax:  SOUND vce,frq,dur[,dir]][,mfq] *
*                  [,stv][,wve][,psw] *
*                               *
* NOTE:  vce = voice (1 - 3) *
*         frq = frequency (0 - 65535) *
*         dur = duration (0 - 32767) *
*         dir = step direction *
*                               *
*         where:  0 = up *
*                 1 = down *
*                 2 = oscillate *
*                               *
```

```

*          mfq = minimum frequency if the sweep is          *
*              used (0 - 65535)                              *
*          stv = step value for sweep  (0 - 32767)          *
*          wve = waveform                                    *
*
*              where:  0 = triangle                          *
*                    1 = sawtooth                          *
*                    2 = variable pulse width (square)    *
*                    3 = noise                              *
*                    4 = ring modulation                   *
*
*          psw = pulse width (0 - 4095)                     *
*
* This command allows you to generate a SOUND whose       *
* characteristics are specified by the parameters that   *
* follow the SOUND command.                               *
*
*****

```

```

71EC: JSR  $87F4      ;Get the voice number into the X register
71EF: DEX
71F0: CPX  #$03      ;Is the voice number greater than 3
71F2: BCC  $71F7      ;If not, then branch
71F4: JMP  $7D28      ;If the voice number is greater than 3, then
                    ;Generate an 'ILLEGAL QUANTITY' error
71F7: STX  $1281      ;Save the voice number
71FA: JSR  $880F      ;Check for a comma and get the frequency value
                    ;Into the Y register and the accumulator
71FD: STY  $12A5      ;Save the LSB
7200: STA  $12A6      ;And the MSB of the frequency
7203: STY  $12AC      ;Again
7206: STA  $12AD
7209: JSR  $880F      ;Check for a comma and get the duration value
                    ;Into the Y register and the accumulator
720C: CMP  #$80      ;If the duration value is greater than 32767,
720E: BCS  $71F4      ;Then generate an 'ILLEGAL QUANTITY' error
7210: STY  $12A3      ;Save the LSB and
7213: STA  $12A4      ;The MSB of the duration value
7216: JSR  $9E1C      ;Check for a comma and get the step direction,
                    ;If any, into the X register
7219: CPX  #$03      ;If the direction value is greater than 2,
721B: BCS  $71F4      ;Then generate an 'ILLEGAL QUANTITY' error
721D: TXA
                    ;Move the direction value to the accumulator
721E: STA  $12A9      ;And save it
7221: AND  #$01      ;Drop all bits except bit 0 to check for a
                    ;Direction of down
7223: PHP
                    ;Save the status register

```

```

7224: JSR   $9E06      ;Get the minimum sweep frequency, if any,
7227: STY   $12A7      ;Save the LSB and
722A: STA   $12A8      ;The MSB of the sweep frequency
722D: JSR   $9E06      ;Get the sweep step value into the Y register
                          ;And the accumulator
7230: PLP                      ;Restore the status register
7231: BEQ   $7240      ;If the sweep direction was not down, branch
7233: PHA                      ;Save the MSB of the step value
7234: TYA                      ;Save the LSB of the step value onto the stack
7235: EOR   #$FF        ;Generate the twos complement
7237: CLC                      ;Of the value
7238: ADC   #$01        ;And move the result
723A: TAY                      ;Back into the Y register
723B: PLA                      ;Restore the accumulator
723C: EOR   #$FF        ;Generate the twos complement
723E: ADC   #$00        ;Of the MSB of the step value
7240: STA   $12AB      ;And save it
7243: TYA                      ;Save the LSB
7244: STA   $12AA      ;Of the step value
7247: LDX   #$02        ;Default waveform
7249: JSR   $9E1E      ;Get the waveform, if any, into the X register
724C: CPX   #$04        ;If the waveform specified is greater than 3,
724E: BCS   $701B      ;Then generate an 'ILLEGAL QUANTITY' error
7250: LDA   $6CB7,X    ;Get the appropriate waveform value,
7253: ORA   #$01        ;Set the GATE bit,
7255: STA   $12B0      ;And save it
7258: JSR   $9E06      ;Get the pulse width, if any, into the Y
                          ;Register and the accumulator
725B: BCS   $7261      ;If a value was specified, then branch
725D: LDA   #$08        ;Set the default pulse width
725F: LDY   #$00        ;Value to 2048
7261: CMP   #$10        ;If the pulse width value is greater than 4095,
7263: BCS   $71F4      ;Then generate an 'ILLEGAL QUANTITY' error
7265: STY   $12AE      ;Save the LSB and
7268: STA   $12AF      ;The MSB of the pulse width
726B: LDA   $12A3      ;If both the MSB
726E: ORA   $12A4      ;And the LSB of the duration value
7271: BEQ   $72B9      ;Are zero, then branch
7273: LDX   $1281      ;Get the voice number
7276: TXA                      ;Move it into the accumulator
7277: ASL                      ;Multiply it by 2
7278: TAY                      ;And use the result as an index
7279: LDA   $1224,Y    ;Wait until
727C: BPL   $7279      ;Both counters
727E: LDA   $1285,X    ;Have counted
7281: BPL   $727E      ;Down to $FF
7283: LDY   #$00        ;Zero the index

```

```

7285: LDA $12A5,Y ;Get the frequency value for the specified voice
7288: STA $1288,X ;Save it as the Sound Minimum and Maximum value
728B: INX ;Move the index
728C: INX ;To the next register
728D: INX ;To update
728E: INY ;Move the index to the next frequency value
728F: CPY #$09 ;Continue until all are
7291: BNE $7285 ;Transferred
7293: LDX $1281 ;Use the voice number as an index
7296: LDY $7039,X ;Get the appropriate index for that voice
7299: JSR $A845 ;Enable BANK 15
729C: LDA #$08 ;Turn off
729E: STA $D404,Y ;The appropriate voice
72A1: LDA #$00 ;Set the Attack/Decay
72A3: STA $D405,Y ;Rate to 0
72A6: LDA #$F0 ;Set the Sustain rate to MAXIMUM
72A8: STA $D406,Y ;And the Release rate to 0
72AB: LDX #$00 ;Zero the index
72AD: LDA $12AC,X ;Get the Sound values
72B0: STA $D400,Y ;And store them in the SID chip
72B3: INY ;Increment the SID index
72B4: INX ;Continue until
72B5: CPX #$05 ;Five values have been
72B7: BNE $72AD ;Transferred
72B9: LDX $1281 ;Use the voice number as an index
72BC: LDY $12A3 ;Get the LSB and
72BF: LDA $12A4 ;The MSB of the duration value
72C2: SEI ;Stop all background jobs
72C3: STA $1285,X ;Save the MSB of the duration value
72C6: TYA ;Move the LSB to the accumulator
72C7: STA $1282,X ;Save the LSB of the duration value
72CA: CLI ;Restart background jobs
72CB: RTS ;And exit

```

```

*****
*
*
*          BASIC WINDOW COMMAND
*
* Command syntax: WINDOW tlc,tlr,brc,brr[,clr]
*
* NOTE: tlc = top left column value
*       tlr = top left row value
*       brc = bottom right column value
*       brr = bottom right row value
*       clr = clear the window (1 = yes, 0 = no)
*

```

```

* This command allows you to define a window on the *
* screen to which the screen display will be output. *
* *
*****

```

```

72CC: JSR   $87F4      ;Get the top left column of the window into the
                        ;X register
72CF: CPX   #$28      ;Check if it is greater than 39
72D1: BIT   $D7       ;Check if we are in the 40 or the 80 column mode
72D3: BPL   $72D7     ;If we are in the 40 column mode, then branch
72D5: CPX   #$50      ;Check if the value is greater than 79
72D7: BCS   $7332     ;If it is greater than 39 in the 40 column mode
                        ;Or 79 in the 80 column mode, then generate
                        ;An 'ILLEGAL QUANTITY' error
72D9: STX   $12B3     ;Save the top left column value
72DC: JSR   $8809     ;Get the top left row value
72DF: CPX   #$19      ;If it is greater than 24
72E1: BCS   $7332     ;Then generate an 'ILLEGAL QUANTITY' error
72E3: STX   $12B4     ;Save the top left row value of the window
72E6: JSR   $8809     ;Get the bottom right column value
72E9: CPX   #$28      ;Check if it is greater than 39
72EB: BIT   $D7       ;Check if we are in the 40 or the 80 column mode
72ED: BPL   $72F1     ;If we are in the 40 column mode
72EF: CPX   #$50      ;Check if it is greater than 79
72F1: BCS   $7332     ;If the value is greater than 39 in the 40
                        ;Column mode, or greater than 79 in the 80
                        ;column mode, then generate an
                        ;'ILLEGAL QUANTITY' error
72F3: STX   $12B5     ;Save the bottom right column of the window
72F6: CPX   $12B3     ;If it is less than
72F9: BCC   $7332     ;The top left column value, then generate an
                        ;'ILLEGAL QUANTITY' error
72FB: JSR   $8809     ;Get the bottom right row value of the window
72FE: CPX   #$19      ;If it is greater than 24
7300: BCS   $7332     ;Then generate an 'ILLEGAL QUANTITY' error
7302: STX   $12B6     ;Save the bottom right row of the window
7305: CPX   $12B4     ;If it is less than
7308: BCC   $7332     ;The top left row value, then generate an
                        ;'ILLEGAL QUANTITY' error
730A: JSR   $9E1C     ;Get the clear flag
730D: CPX   #$02      ;If it is greater than 1
730F: BCS   $7332     ;Then generate an 'ILLEGAL QUANTITY' error
7311: TXA                   ;Save the clear window flag
7312: PHA                   ;Onto the stack
7313: LDX   $12B3     ;Get the top left column of the window
7316: LDA   $12B4     ;Get the top left row of the window
7319: CLC                   ;Clear the carry to set the top left row column

```

```

731A: JSR  $C02D      ;Set up the top left row column of window
731D: LDX  $12B5      ;Get the bottom right column of the window
7320: LDA  $12B6      ;Get the bottom right row of the window
7323: SEC                      ;Set the carry to set bottom right row column
7324: JSR  $C02D      ;Set up the bottom right row, column of window
7327: LDX  #$13       ;Value for 'HOME'
7329: PLA                      ;Get the clear window flag
732A: BEQ  $732E      ;If the window is not to be cleared, then
                          ;Just output 'HOME'
732C: LDX  #$93       ;Value for 'CLR' clear screen
732E: TXA                      ;Move the characters to the accumulator
732F: JMP  $9269      ;And output
7332: JMP  $7D28      ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
*          BASIC BOOT COMMAND          *
*
* Command syntax:                      *
*
* BOOT "filename"[,DR] [<ON,>DV] [,BN][,P] *
*
* NOTE:  DR = 'D' + the drive number   *
*        DV = 'U' + the device number  *
*        BN = 'B' + the bank number to load in *
*        P  = 'P' + the alternate starting address *
*
* Example:  10 A$="EXAMPLE":A=2816     *
*          20 BOOT(A$),D0,U8,B15,P(A)  *
*
* When you RUN the example program listed above, the *
* program named "EXAMPLE" will be BLOADED into memory *
* starting at address 2816 and then SYSed to at that *
* starting address to activate the program.          *
*
* The following examples achieve the same results:   *
*
* Example 1: 10 BOOT "EXAMPLE",P2816,U8,D0,B15      *
*
*          OR          *
*
* Example 2: 10 BOOT "EXAMPLE",P(DEC"0B00")),U8,D0,B15 *
*
* WARNING: This command only works with a 1571 disk *
*          drive that is in the 1571 mode. If this *
*          routine is used with something other than *
*          a 1571 disk drive in the 1571 mode, the *

```



```

*           computer will crash with a probable           *
*           'BREAK' into the MONITOR.                     *
*                                                                 *
* NOTE:      If you know the address to SYS to, you can *
*            POKE this address into locations $AC, $AD *
*            as LSB, MSB format.                          *
*                                                                 *
*****

```

```

7335: LDA   #$E6           ;Ensure that the only parameters specified are
7337: LDX   #$FC           ;The Filename, Drive number, Device number
7339: JSR   $A3C3          ;The Bank number, and the New starting address
733C: LDA   $80           ;Get the bit value of the parameters used
733E: LSR                   ;Shift the Filename bit into the carry
733F: BCC   $735E          ;If there is no Filename, go to the boot sector
7341: JSR   $A21F          ;Perform a 'BLOAD'
7344: BCS   $736F          ;If there is an error, then branch
7346: LDX   $03D5          ;BANK for SYS, POKE, PEEK set by BANK command
7349: LDA   $81           ;Check if a BANK
734B: LSR                   ;Number was specified in BOOT command statement
734C: BCC   $7351          ;If not, then branch
734E: LDX   $011F          ;Get the BASIC BANK number
7351: STX   $02           ;Save it
7353: LDA   $AC           ;Get the LSB of the starting address
7355: STA   $04           ;Save it as the LSB of the address to JSR to
7357: LDA   $AD           ;Get the MSB of the starting address
7359: STA   $03           ;Save it as the MSB of the address to JSR to
735B: JMP   $FF6E          ;Call JSRFAR to execute the program

```

```

*-----*
*
* This routine will load Track 1, Sector 0 into $0B00
* in an attempt to locate the required information.
*
* NOTE: For more information on this routine, refer
* to the owner's manual.
*
*-----*

```

```

735E: LDA   $0112          ;Get the drive number
7361: ORA   #$30           ;Create an ASCII digit
7363: LDX   $011C          ;Get the current device number
7366: JSR   $A845          ;Enable BANK 15
7369: JSR   $FF53          ;Check for the Boot sector and the Boot File
736C: BCS   $736F          ;If there is an error, then branch
736E: RTS                   ;Exit

```

```

*-----*
*
*   Jump to the BASIC/KERNAL error message handler.
*
*-----*

```

```
736F: JMP    $90D0      ;Generate a BASIC error message for the KERNAL
```

```

*****
*
*           BASIC SPRDEF COMMAND
*
*   Command syntax:  SPRDEF
*
*   This command allows you to enter the sprite
*   definition utility which is part of the 128's
*   operating system.  This utility allows you to create
*   and edit a sprite on the screen that can be called
*   and used later by the SPRITE commands.
*
*****

```

```

7372: JSR    $9F4F
7375: JSR    $A845      ;Enable BANK 15
7378: LDA    #$D0      ;Set the page number
737A: STA    $1168     ;Of the character set to be used
737D: LDA    #$20      ;Turn on the HIRES
737F: STA    $D8       ;Bit map
7381: JSR    $6B30     ;Clear the bit map
7384: LDY    #$80      ;Set bit 7
7386: STY    $113D     ;So that reversed characters are printed
7389: LDY    #$18      ;Starting column for the border
738B: LDA    #$20      ;Value to be outputted
738D: LDX    #$00      ;Starting row
738F: JSR    $68DB     ;Output a space
7392: INX
7393: CPX    #$15      ;Have we done twenty-one rows yet?
7395: BCC    $738F     ;If not, then branch
7397: JSR    $68DB     ;If we have done twenty-one rows, then
739A: DEY
739B: BPL    $7397     ;Continue until the bottom border is completed
739D: JSR    $A845     ;Enable BANK 15
73A0: LDA    $F1       ;Color code under cursor for the character
                        ;output
73A2: PHA
                        ;Save it onto the stack
73A3: LDA    $D021     ;Get the current background color
73A6: STA    $F1       ;Save it as the character color

```

```

73A8: LDA    #'+'      ;Value to output
73AA: LDX    #$00      ;Starting row of 0
73AC: STX    $113D     ;Clear bit 7 to print normal characters
73AF: LDY    #$00      ;Starting column to 0
73B1: JSR    $68DB     ;Print the value to the HIRES screen
73B4: INY                    ;Move to the next column
73B5: CPY    #$18      ;Continue until twenty-four columns
73B7: BCC    $73B1     ;Are filled with '+'
73B9: INX                    ;Move to the next row
73BA: CPX    #$15      ;Continue until
73BC: BCC    $73AF     ;Twenty-one rows are filled with '+'
73BE: PLA                    ;Get the character color back
73BF: STA    $F1       ;Restore the character color
73C1: JSR    $76D4     ;Color the Sprite definition block to hide '+'

```

```

*****
*
*          PRINT 'SPRITE NUMBER?' to the screen.
*
*****

```

```

73C4: LDY    #$02      ;Index to the message and to the starting column
73C6: LDX    #$17      ;Row value to print to
73C8: LDA    $766C,Y   ;Get a character of the message
73CB: BEQ    $73D3     ;If it is the delimiter, then branch
73CD: JSR    $68DB     ;Output the character to the HIRES screen
73D0: INY                    ;Move to the next character
73D1: BNE    $73C8     ;Continue output
73D3: JSR    $A845     ;Enable BANK 15
73D6: JSR    $FFE4     ;Get a character from the keyboard
73D9: BEQ    $73D6     ;Continue scanning until a key is depressed
73DB: CMP    #13       ;<CR>?
73DD: BNE    $73E7     ;If not, then branch
73DF: JSR    $6B30     ;Clear the bit map screen
73E2: LDA    #0        ;Return to the
73E4: STA    $D8       ;Text mode
73E6: RTS                    ;Exit

```

```

*****
*
*          TEST FOR DIGITS 1 - 8
*
*****

```

```

73E7: SEC                    ;Set the carry for subtraction
73E8: SBC    #$31       ;Create the HEX value - 1
73EA: STA    $12FC     ;Save the sprite number - 1

```

```

73ED:  CMP    #$08      ;If the value entered was greater than 8,
73EF:  BCS    $73D6     ;Then go back to the input loop
73F1:  TAX
73F2:  ASL
73F3:  TAY
73F4:  LDA    $6CB3,X   ;Get the sprite bit number
73F7:  STA    $116D     ;And save it
73FA:  AND    $D01C     ;Check if the selected sprite is multicolor
73FD:  BEQ    $7401     ;If not, then branch
73FF:  LDA    #$80      ;Value for multicolor sprite
7401:  STA    $12FA     ;For multicolor (128 = multicolor)
7404:  LDA    #$08      ;Set the sprite's
7406:  STA    $11D6,Y   ;X coordinate LSB to 8
7409:  LDA    #$4A      ;Set the sprite's
740B:  STA    $11D7,Y   ;Y coordinate to 74
740E:  LDA    $116D     ;Get the sprite's bit number
7411:  ORA    $11E6     ;Set the MSB of that sprite's X coordinate to 1
7414:  STA    $11E6     ;Store it back to set the sprite's X coordinate to 264
7417:  LDA    $116D     ;Get the sprite ON/OFF configuration
741A:  STA    $D015     ;Enable the sprite
741D:  LDX    $12FC     ;Use the sprite number - 1 as an index
7420:  LDY    $6DD9,X   ;Get the offset to the speed value of the sprite
7423:  LDA    #$00      ;Set the currently displayed
7425:  STA    $117E,Y   ;Sprite's speed to zero
7428:  TXA
7429:  LDY    #$11      ;Move the sprite number - 1 into the accumulator
742B:  LDX    #$17      ;Column to start printing at
742D:  CLC
742E:  ADC    #$31      ;Clear the carry flag for addition
7430:  JSR    $68DB     ;Convert the sprite number to ASCII
7433:  JSR    $A845     ;Output the digit to the HIRES screen
7436:  LDA    $12FC     ;Enable BANK 15
7439:  LSR
743A:  ROR
743B:  ROR
743C:  STA    $4B      ;As a base
743E:  LDY    #$0E     ;Calculate the proper
7440:  BCC    $7443     ;Address for the sprite
7442:  INY
7443:  STY    $4C      ;Data
7445:  JSR    $75D1     ;And store the
7448:  LDY    #$3F     ;Address
744A:  LDA    ($4B),Y   ;In $4B,$4C
744C:  STA    $12B7,Y   ;Adjust the sprite's color
744F:  DEY
7450:  BPL    $744A     ;Load the Y register with the number of bytes
                          ;To transfer
744A:  LDA    ($4B),Y   ;Transfer the sprite data
744C:  STA    $12B7,Y   ;To be used by the SPRDEF routine
744F:  DEY
7450:  BPL    $744A     ;Move to the next byte
                          ;Continue until 64 bytes have been transferred

```

```

7452: LDX    #$00      ;Set the starting
7454: STX    $115F     ;Column to zero
7457: STX    $115E     ;And the starting row to zero
745A: JSR    $764A     ;Turn on the cursor by toggling the RVS bit
745D: JSR    $FFE4     ;Get a key from the keyboard
7460: BEQ    $745D     ;Wait until a key is pressed
7462: PHA                    ;Save the key value on to the stack
7463: JSR    $764A     ;Turn the cursor off by toggling the RVS bit
7466: PLA                    ;Get the key value off of the stack
7467: LDX    #$10      ;Set up an index for the 16 colors
7469: CMP    $76B4,X   ;Check the table for the proper color value
746C: BNE    $747B     ;If it is not the same, then branch
746E: DEX                    ;Decrement the index by one
746F: TXA                    ;Move it into the accumulator
7470: LDX    $12FC     ;Get the current sprite number - 1
7473: STA    $D027,X   ;Set the color of the proper sprite
7476: JSR    $75D1     ;Adjust the sprite's color
7479: BCS    $745A     ;Return back to the waiting loop
747B: DEX                    ;Move to the next element in the table
747C: BNE    $7469     ;Continue checking
747E: LDX    #$11      ;Set up an index for 18 keys
7480: CMP    $767F,X   ;Compare the key that was pressed to the table
7483: BEQ    $748A     ;If they are the same, then branch
7485: DEX                    ;If they are not the same, then move the index
                          ;To the next table element
7486: BPL    $7480     ;Continue checking
7488: BMI    $745A     ;If there was no match, then return to the wait loop
748A: TXA                    ;Move the index to the accumulator
748B: TAY                    ;And to the Y register also
748C: ASL                    ;Multiply the index by 2
748D: TAX                    ;And move it back into the X register
748E: LDA    $7691,X   ;Get the MSB of the routine's address
7491: PHA                    ;And save it onto the stack
7492: LDA    $7691+1,X ;Get the LSB of the address - 1
7495: PHA                    ;And save it onto the stack
7496: RTS                    ;And RTS to execute the routine on the stack

```

```

*****
*
* This routine calculates the proper color value when *
* the keys 1 - 4 are pressed. It then stores the *
* proper value for that color into the sprite data. *
*
* On entry to this routine, the Y register contains *
* the key value - 1 (0 - 3). *
*
*****

```

---

```

7497: TYA                ;Move the key index into the accumulator
7498: STA    $8F        ;And save it
749A: JSR    $7610     ;Calculate the color value
749D: PHA                ;Save it onto the stack temporarily
749E: LDY    $115E     ;Get the current column
74A1: LDX    $115F     ;And the current row values
74A4: JSR    $76C5     ;Calculate the color memory address
74A7: PLA                ;Get the color value back off of the stack
74A8: JSR    $763F     ;Set the proper point
74AB: LDY    $115E     ;Get the current column value
74AE: TYA                ;Move it into the accumulator
74AF: AND    #$07      ;Calculate the bit offset in the byte
74B1: TAX                ;Move the result into the X register
74B2: TYA                ;Move the column value into the accumulator
74B3: LSR                ;And divide
74B4: LSR                ;The value
74B5: LSR                ;By 8
74B6: CLC                ;Clear the carry for addition
74B7: ADC    $115F     ;And add the result
74BA: ADC    $115F     ;To the current row value
74BD: ADC    $115F     ;To generate an index
74C0: TAY                ;Move the index to the Y register
74C1: LDA    ($4B),Y   ;Get the value of the specified byte
74C3: BIT    $12FA     ;Check for the multicolor mode
74C6: BPL    $74E0     ;If not in the multicolor mode, then branch
74C8: STA    $8E        ;Save the value temporarily
74CA: LDA    $9D1C,X   ;Get the proper bit value
74CD: ORA    $9D1D,X   ;Set the bit next to it also
74D0: PHA                ;Save the result onto the stack
74D1: ORA    $8E        ;Combine it with the byte value
74D3: STA    $8E        ;And save it
74D5: PLA                ;Get the bit value off of the stack
74D6: LDX    $8F        ;Get the index to the color source
74D8: AND    $9F25,X   ;Drop the bits from the multicolor table
74DB: EOR    $8E        ;Toggle the bits that were calculate above
74DD: JMP    $74EA     ;Jump to store the result
74E0: ORA    $9D1C,X   ;Set the proper bit
74E3: ASL    $8F        ;Check to see if the bit was to be
74E5: BNE    $74EA     ;Set or cleared
74E7: EOR    $9D1C,X   ;Clear the bit
74EA: STA    ($4B),Y   ;Store the result in the sprite data
74EC: BIT    $12FA     ;Check to see if the Auto Cursor movement is
                          ;On or off
74EF: BVC    $753F     ;If the option is on, then move the cursor
74F1: JMP    $745A     ;Else, return to the waiting loop

```

```
*****
*
* This routine is used to transfer the sprite data
* from the work area at $12B7 - $12F6 to the proper
* sprite data area whose address is contained in
* locations $4B, $4C. The routine then goes to $73C4
* which restarts the editor by prompting the user for
* another sprite number.
*
*****
```

```
74F4: LDY   #$3F      ;Index to the number of bytes to transfer (64)
74F6: LDA   $12B7,Y   ;Get the data from the work area
74F9: STA   ($4B),Y   ;And save it as the new sprite data
74FB: DEY                   ;Continue until
74FC: BPL   $74F6     ;All 64 bytes have been transferred
74FE: LDA   #$00      ;Disable the sprite
7500: STA   $D015     ;That is being displayed
7503: JMP   $73C4     ;And go to get the next sprite number
```

```
*****
*
* This routine is executed whenever the letter 'M' is
* depressed to toggle the multicolor bit (bit 7) in
* location $12FA. The routine toggles the appropriate
* bit in location $D01C to select the
* Standard/Multicolor mode and then adjusts the
* display for the standard or multicolor sprite
* editing.
*
*****
```

```
7506: LDA   $12FA     ;Get the multicolor flag
7509: EOR   #$80      ;Toggle the multicolor bit
750B: STA   $12FA     ;And save it back
750E: JSR   $75D1     ;Adjust the sprite's color
7511: LDA   $115E     ;Adjust the cursor
7514: AND   #$FE      ;Position
7516: STA   $115E     ;And save it back
7519: LDY   #$1C      ;Index for the sprite multicolor mode register ($D01C)
751B: .BYTE $2C      ;MASK
```

```
*****
*
* This routine is executed whenever the 'Y' key is
* depressed to toggle the 'Y' expansion of the sprite
* that is currently being displayed. The routine
*
```

```

* exits by returning to the input waiting loop at      *
* $745A.                                               *
*                                                       *
*****

```

```

751C: LDY  #$17      ;Index for the Sprite Y expand register ($D017)
751E: .BYTE $2C      ;MASK

```

```

*****
*
* This routine is executed whenever the 'X' key is      *
* depressed to toggle the 'X' expansion of the sprite *
* that is currently being displayed. The routine exits *
* by returning to the input waiting loop at $745A.     *
*                                                       *
*****

```

```

751F: LDY  #$1D      ;Index for the Sprite X expand register ($D01D)
7521: LDA  $D000,Y    ;Get the current value
7524: LDX  $12FC      ;Get the current sprite number
7527: EOR  $6CB3,X    ;Set/Clear the proper bit for this sprite
752A: STA  $D000,Y    ;And save the new value
752D: JMP  $745A      ;Return to the waiting loop

```

```

*****
*
* This routine clears the 64 bytes of sprite data      *
* which is addressed by locations $4B, $4C. The        *
* routine then sets the work area on the screen to    *
* the background color to hide the '+' and then exits *
* by returning to the input waiting loop at $745A.     *
*                                                       *
*****

```

```

7530: LDY  #$3F      ;Index to the number of bytes to clear (64)
7532: LDA  #$00      ;Value to clear a byte
7534: STA  ($4B),Y    ;Clear the sprite data
7536: DEY                ;Move to the next byte
7537: BPL  $7534      ;Continue until 64 bytes have been cleared
7539: JSR  $76D4      ;Set the definition screen to the Background color
753C: JMP  $7452      ;Set the cursor coordinates to 0,0 and then
                    ;Return to the waiting loop

```

```

*****
* This routine is executed whenever the Cursor Right *
* or Cursor Left keys are depressed. The 'cursor' is *
* moved by erasing the Reverse bit of the current    *

```



```

* location, moving to the next byte, and setting the *
* Reverse bit at that location. *
* *
*****

```

```

753F: LDA  #$01      ;Flag for Cursor Right
7541: .BYTE $2C      ;MASK
7542: LDA  #$FF      ;Flag for Cursor Left
7544: BIT  $12FA     ;Check for the multicolor mode
7547: BPL  $754A     ;If not in the multicolor mode, then branch
7549: ASL           ;Multiply the value by 2
754A: CLC           ;Clear the carry for addition
754B: ADC  $115E     ;Add the result to the current column value
754E: BMI  $7559     ;If the cursor is to 'wrap-around', then branch
7550: CMP  #$18      ;Compare the value to the maximum column + 1
7552: BCS  $7581     ;If it is greater than 23, then branch
7554: STA  $115E     ;Save the new column value
7557: BCC  $757E     ;And branch
7559: LDX  #$17      ;Maximum column value for the non-multicolor mode
755B: BIT  $12FA     ;Check the multicolor mode flag
755E: BPL  $7561     ;If not in the multicolor mode, then branch
7560: DEX           ;If in the multicolor mode, then decrement the maximum
                          ;Column value by one
7561: STX  $115E     ;And save it as the new column value

```

```

*****
* *
* This routine is executed whenever the Cursor Up *
* or Cursor Down keys are depressed. The 'cursor' is *
* moved by erasing the Reverse bit of the current *
* location, moving to the next byte, and setting the *
* Reverse bit at that location. *
* *
*****

```

```

7564: LDA  #$FF      ;Flag for Cursor Up
7566: .BYTE $2C      ;MASK
7567: LDA  #$01      ;Flag for Cursor Down
7569: CLC           ;Clear the carry for addition
756A: ADC  $115F     ;Add the flag to the current row value
756D: CMP  #$15      ;And compare it to the maximum row value
756F: BCS  $757E     ;If the value is too large, then exit
7571: STA  $115F     ;Save the new row value
7574: BCC  $757E     ;And branch to return to the waiting loop

```

```
*****
*
* This routine is executed whenever the 'A' key is *
* depressed. The routine toggles the Auto-Cursor *
* movement bit (bit 6) in location $12FA. The *
* routine exits by returning to the input waiting loop.*
*
*****
```

```
7576: LDA $12FA ;Get the Auto Cursor flag
7579: EOR #$40 ;Toggle bit 6 to Enable/Disable the Auto
;Cursor movement
757B: STA $12FA ;And save the flag back
757E: JMP $745A ;Return to the waiting loop
```

```
*****
*
* This routine is executed whenever the RETURN key is *
* pressed to move the 'cursor' down one line and all *
* the way to the left of the screen. The routine *
* exits by returning to the input waiting loop. *
*
*****
```

```
7581: LDA #$00 ;Set the column
7583: STA $115E ;Value to zero
7586: BEQ $7567 ;And move the cursor down one line
```

```
*****
*
* This routine is called whenever the 'C' key is *
* pressed. It outputs the message 'COPY FROM?' to *
* the screen and then handles the copying of the data *
* from another area to the current data area. *
*
*****
```

```
7588: LDY #$02 ;Index to message
758A: LDX #$18 ;Row to start printing at
758C: LDA $7661,Y ;Get a byte of the message
758F: BEQ $7597 ;If it is equal to zero, then branch
7591: JSR $68DB ;Output the character
7594: INY ;Move to the next byte in the message
7595: BNE $758C ;Continue printing the message
7597: JSR $A845 ;Enable BASIC BANK 15
759A: JSR $FFE4 ;Get a key from the keyboard
759D: BEQ $759A ;Wait until a key is pressed
```

```

759F:  CMP    #$0D    ;Is the key a carriage return?
75A1:  BEQ    $75C2    ;If it is, then branch
75A3:  SEC                    ;Set the carry for subtraction
75A4:  SBC    #'1'      ;Check for
75A6:  CMP    #$08    ;Digits 1 - 7
75A8:  BCS    $759A    ;If the digit entered is not a digit between 1 and 7,
                    ;Then return to the waiting loop

75AA:  LSR                    ;Calculate
75AB:  ROR                    ;The LSB value of the
75AC:  ROR                    ;Sprite data address
75AD:  STA    $8E      ;And save the result
75AF:  LDY    #$0E    ;Store the starting MSB in the Y register
75B1:  BCC    $75B4    ;If the sprite number was 4 or greater,
75B3:  INY                    ;Then increment the MSB by one
75B4:  STY    $8F      ;And save it
75B6:  LDY    #$3F    ;Set up an index to the number of bytes to transfer
75B8:  LDA    ($8E),Y    ;Get the sprite data
75BA:  STA    ($4B),Y    ;And save it for the sprite editor
75BC:  DEY                    ;Move to the next byte of data
75BD:  BPL    $75B8    ;Continue until 64 bytes have been transferred
75BF:  JSR    $75D1    ;Adjust the sprite's color
75C2:  LDA    #$00    ;Value to clear to
75C4:  TAY                    ;Set up an index
75C5:  STA    $3E00,Y   ;Clear the prompt from the
75C8:  DEY                    ;HIRES screen
75C9:  BNE    $75C5    ;Continue until 256 bytes have been cleared
75CB:  JMP    $745A    ;Return to the waiting loop
75CE:  .BYTE  $FF,$FF   ;Spare bytes
75D0:  .BYTE  $FF      ;Spare byte

```

```

*****
*
* This routine is used to change the color of the
* whole sprite on the screen work area.
*
*****

```

```

75D1:  LDX    #$00    ;Clear
75D3:  STX    $1160    ;An index
75D6:  STX    $12FB    ;Set the starting row to zero
75D9:  JSR    $76C5    ;Calculate the color memory address
75DC:  LDY    #$00
75DE:  LDX    #$08    ;Bit counter
75E0:  STY    $116E    ;Set the starting column
75E3:  LDY    $1160    ;Get the byte index
75E6:  LDA    ($4B),Y   ;Get a byte of the sprite data
75E8:  INC    $1160    ;Increment to the next byte

```

```

75EB: LDY   $116E   ;Get the column index
75EE: ASL                   ;Shift bit 7 out
75EF: BIT   $12FA   ;Check for the multicolor mode
75F2: BPL   $75F6   ;If not in the multicolor mode, then branch
75F4: ROL                   ;If in the multicolor mode, then shift out one more bit
75F5: DEX                   ;Decrement the bit counter by one
75F6: PHA                   ;And save the result
75F7: ROL                   ;Shift out another bit
75F8: JSR   $7610   ;Calculate the proper color value
75FB: JSR   $763F   ;Set the proper point
75FE: INY                   ;Index to the next column
75FF: PLA                   ;Get the shifted value back
7600: DEX                   ;Decrement the bit counter by one
7601: BNE   $75EE   ;Continue until all 8 bits are done
7603: CPY   #$18     ;Check to see if this row is done
7605: BCC   $75DE   ;If it is not, then branch and continue
7607: LDX   $12FB   ;Else, get the current row value
760A: INX                   ;Move to the next row
760B: CPX   #$15     ;Have all of the rows been done yet?
760D: BCC   $75D6   ;If not, then branch and continue
760F: RTS                   ;Exit the routine

```

```

*****
*
*   This routine gets the proper color value from the
*   color source number (0 - 3) which is in the
*   accumulator.  The result is saved to location $8E.
*
*****

```

```

7610: AND   #$03         ;Drop all bits except, bits 0 and 1
7612: LSR                   ;Shift bits 0 and 1
7613: ROR                   ;Out to check for the color source
7614: BEQ   $7625   ;Unconditional branch
7616: BIT   $12FA   ;Check for the multicolor mode
7619: BPL   $762A   ;If not in the multicolor mode, then branch
761B: LDA   $D025   ;Get the Multicolor 0 value
761E: BCC   $7634   ;If the color source, Multicolor 0 was selected,
                    ;Then branch
7620: LDA   $D026   ;Get the Multicolor 1 value
7623: BCS   $7634   ;If the color source, Multicolor 1, was selected,
                    ;Then branch
7625: LDA   $D021   ;Get the Background color value
7628: BCC   $7634   ;If the color source, Background color, was selected,
                    ;Then branch
762A: STX   $8E     ;Save the index temporarily
762C: LDX   $12FC   ;Get the current sprite number

```

```

762F: LDA  $D027,X    ;Get the color of the specified sprite
7632: LDX  $8E        ;Get the index back
7634: AND  #$0F       ;Drop the high nybble of the color value
7636: STA  $8E        ;And save the result
7638: ASL                ;Shift the
7639: ASL                ;Lower nybble
763A: ASL                ;Into the
763B: ASL                ;High nybble
763C: ORA  $8E        ;Combine the two values
763E: RTS                ;And exit

```

```

*****
*
* This routine changes the color of a point in the
* work area according to the value that is in the
* accumulator. The address of the point is contained
* in locations $8C, $8D.
*
*****

```

```

763F: STA  ($8C),Y    ;Set the color at the proper point
7641: BIT  $12FA      ;Check for the multicolor mode
7644: BPL  $7649      ;If not in the multicolor mode, then branch
7646: INY                ;If in the multicolor mode, then
7647: STA  ($8C),Y    ;Set two points
7649: RTS                ;And exit the routine

```

```

*****
*
* This routine is used to calculate the address of
* the 'cursor' according to the row ($115F) and the
* column ($115E) values. The routine then moves the
* 'cursor' to the proper location.
*
*****

```

```

764A: LDX  $115F      ;Get the current row value
764D: JSR  $76C5      ;Calculate the color memory address in $8C, $8D
7650: LDY  $115E      ;Get the current column address
7653: BIT  $12FA      ;Check for the multicolor mode
7656: BPL  $765B      ;If not in the multicolor mode, then branch
7658: JSR  $765B      ;If in the multicolor mode, then do two spaces
765B: LDA  ($8C),Y    ;Get the current color
765D: EOR  #$80       ;Toggle the ReVerSe bit of the cursor
765F: STA  ($8C),Y    ;And save it back
7661: INY                ;Increment to the next byte
7662: RTS                ;And exit the routine

```

```

*-----*
*
* The following is the text used by the SPRDEF command.*
*
*-----*
    
```

```

7663: TXT 'copy from?'
766D: .BYTE $00 ;End of text flag
766E: TXT 'sprite number? '
767E: .BYTE $00 ;End of text flag
    
```

```

*****
*
* The following table contains the key values that
* are recognized by the SPRDEF routine.
*
*****
    
```

```

767F: TXT '1234' ; Keys 1 - 4
7683: .BYTE $03,$8D ; RUN/STOP key, Shifted RETURN
7685: .BYTE $58,$59 ; X, Y
7687: .BYTE $4D,$9D ; M, Cursor Left
7689: .BYTE $1D,$91 ; Cursor Right, Cursor Up
768B: .BYTE $11,$93 ; Cursor Down, Clear Screen
768D: .BYTE $13,$41 ; Home, A
768F: .BYTE $0D,$43 ; Carriage RETURN, C
    
```

```

*****
*
* The following table contains the addresses - 1 of the*
* routines that are to be executed whenever the corres-
* ponding key from the table at $767F is encountered. *
*
*****
    
```

```

7691: .BYTE $74,$96 ; 1
7693: .BYTE $74,$96 ; 2
7695: .BYTE $74,$96 ; 3
7697: .BYTE $74,$96 ; 4
7699: .BYTE $74,$F3 ; RUN/STOP
769B: .BYTE $74,$FD ; Shifted RETURN
769D: .BYTE $75,$1E ; X
769F: .BYTE $75,$1B ; Y
76A1: .BYTE $75,$05 ; M
76A3: .BYTE $75,$41 ; Cursor Left
76A5: .BYTE $75,$3E ; Cursor Right
76A7: .BYTE $75,$63 ; Cursor Up
    
```

```

76A9: .BYTE $75,$66      ; Cursor Down
76AB: .BYTE $75,$2F      ; Clear the work area
76AD: .BYTE $74,$51      ; Home
76AF: .BYTE $75,$75      ; A
76B1: .BYTE $75,$80      ; Carriage RETURN
76B3: .BYTE $75,$87      ; C

```

```

*****
*
* The following table contains the color values that
* are recognized by the SPRDEF routine.
*
*****

```

```

76B5: .BYTE $90,$05      ; Black,      White
76B7: .BYTE $1C,$9F      ; Red,       Cyan
76B9: .BYTE $9C,$1E      ; Purple,    Green
76BB: .BYTE $1F,$9E      ; Blue,     Yellow
76BD: .BYTE $81,$95      ; Orange,    Brown
76BF: .BYTE $96,$97      ; Light Red, Dark Gray
76C1: .BYTE $98,$99      ; Gray,     Light Green
76C3: .BYTE $9A,$9B      ; Light Blue, Light Gray

```

```

*****
*
* This routine is used to calculate the color memory
* address by the row value in the X register. On
* entry, the X register should hold the row value
* (from 0 to 24). On exit, locations $8C, $8D will
* hold the LSB, MSB respectively of the corresponding
* color memory address which starts at $1C00 when a
* Graphic screen is allocated.
*
*****

```

```

76C5: LDA  $C033,X      ;Get the LSB of the screen address
76C8: STA  $8C          ;Save it to be used as the color memory address
76CA: LDA  $C04C,X      ;Get the MSB of the screen address
76CD: AND  #$03         ;Drop all bits but 0 and 1
76CF: ORA  #$1C         ;Add $1C to point to the color memory,
76D1: STA  $8D          ;Save it,
76D3: RTS              ;And exit

```

```

*****
*
* This routine is used by the SPRDEF routine to color
* the 24 column by 21 row sprite definition block
* which is displayed on a HIRES screen.
*
*****

76D4: LDA  $D021      ;Get the current background color
76D7: JSR  $7634      ;Put the color in both the high and low nybble
76DA: LDX  #$14       ;Value for 21 rows - 1
76DC: PHA                    ;Save the color value
76DD: JSR  $76C5      ;Calculate the line start address for color
76E0: PLA                    ;Get the color value back
76E1: LDY  #$17       ;Load the Y register with the number of columns
                          ;To fill per line - 1
76E3: STA  ($8C),Y    ;Save the color value
76E5: DEY                    ;Move to the next column
76E6: BPL  $76E3      ;Continue until 24 columns are done
76E8: DEX                    ;Move to the next row
76E9: BPL  $76DC      ;Continue until 21 rows are done
76EB: RTS                    ;Exit

*****
*
*                               BASIC SPRSAV COMMAND
*
* Command syntax:  SPRSAV source, destination
*
* NOTE:           source = the origin of the sprite data
*                  to be transferred
*                  destination = where the sprite data is to be
*                  transferred to
*
* This command allows you to move sprite data from
* a string into the sprite storage area or from the
* sprite storage area into a string.  You can also
* transfer sprite data from one sprite to another.
*
*****

76EC: JSR  $777C      ;Check the first character after the SPRSAV
                          ;command
76EF: BCS  $7720      ;If the carry is set, then it is a
                          ;String to sprite transfer
76F1: STA  $4B        ;If the carry is cleared, then it is either a
                          ;Sprite to string transfer or

```



```

;Sprite to sprite transfer
76F3: STY $4C ;Save the address to SPDATA
76F5: LDY #$3E ;Set the counter for 63 bytes
76F7: LDA ($4B),Y ;Move the sprite data
76F9: STA $12B7,Y ;To a temporary storage area
76FC: DEY ;Continue until
76FD: BPL $76F7 ;64 bytes are transferred
76FF: INY
7700: STY $12F7 ;Clear the MSB of the Y coordinate
7703: STY $12F9 ;And the X coordinate
7706: LDA #$17 ;Load the accumulator with the value for the LSB
;of the X coordinate
7708: STA $12F6 ;Save it
770B: LDA #$14 ;Load the accumulator with the value for the LSB
;of the Y coordinate
770D: STA $12F8 ;Save it
7710: LDX #$B7 ;Load the X register with the value for the LSB
;Of the address of the sprite data
7712: LDY #$12 ;Load the Y register with the value for the MSB
;Of the address of the sprite data
7714: STX $70 ;Save the address of the string
7716: STY $71 ;In STRING1 for the next JSR
7718: LDA #$43 ;Number of bytes = 63 for the
;Sprite + 4 for the coordinate
771A: JSR $86CC ;Transfer the data from sprite to string
771D: JSR $7799 ;Get the next string
7720: STX $03DB ;Get the length of the string
7723: STA $03DC ;Get the LSB and
7726: STY $03DD ;The MSB of the string address
7729: JSR $795C ;Check for a comma and if it is not found, then
;Generate a 'SYNTAX' error
772C: LDA $3D ;Get the LSB of TXTPTR
772E: STA $03E0 ;And save it
7731: LDA $3E ;Get the MSB of TXTPTR
7733: STA $03E1 ;And save it
7736: JSR $777C ;Check the next character
7739: BCS $7760 ;If the carry is set, then it is a string
773B: STA $8C ;Save the address of the sprite data
773D: STY $8D ;In GRAPNT
773F: LDA $03DC ;Get the LSB of the address to the string
7742: STA $4B ;And save it
7744: LDA $03DD ;Get the MSB of the address to the string
7747: STA $4C ;And save it
7749: LDY #$00
774B: CPY $03DB ;Compare the length of the string to zero
774E: BEQ $775F ;If the length is equal to zero, then exit
7750: LDA #$4B ;Load the accumulator with the value of the LSB

```

```

;For the string's address
7752: JSR  $03AB      ;Get a byte of the string
7755: STA  $FF03      ;Enable BANK 14
7758: STA  ($8C),Y    ;Save the data to the sprite
775A: INY                ;Increment to the next byte
775B: CPY  #$3F      ;Continue saving the data to the sprite
775D: BNE  $774B      ;Until 63 bytes are done
775F: RTS                ;And Exit
7760: LDA  $03E0      ;Get the LSB of TXTPTR back
7763: STA  $3D        ;And save it
7765: LDA  $03E1      ;Get the MSB of TXTPTR back
7768: STA  $3E        ;And save it
776A: JSR  $7AAF      ;Get the variable pointed to by CHRGET
776D: STA  $4B        ;Save the LSB and
776F: STY  $4C        ;The MSB of the address of the variable
7771: LDA  #$DB       ;Place the address of the
7773: STA  $66        ;String's descriptor
7775: LDA  #$03       ;Into
7777: STA  $67        ;Locations $66, $67
7779: JMP  $5405      ;Assign the value to the string

*****
*
* This routine checks the first character after the
* SPRSAV command and then acts on the sprite number.
*
*****

777C: JSR  $77EF      ;Evaluate the expression after the command
777F: BIT  $0F        ;Check the DATA Flag
7781: BMI  $7799      ;If the data type is a string, then branch
7783: JSR  $87F7      ;Get the sprite number into the X register
7786: DEX                ;If the sprite number
7787: CPX  #$08       ;Is greater than - 1,
7789: BCS  $7796      ;Then generate an 'ILLEGAL QUANTITY' error
778B: TXA
778C: LSR                ;Divide by 2
778D: ROR                ;Shift the carry
778E: ROR                ;Into bit 6 in the accumulator
778F: LDY  #$0E       ;Load the Y register with the value of the MSB
;To the sprite definition area
7791: BCC  $7794      ;If an overflow has not occurred, then branch
7793: INY                ;If an overflow has occurred, then increment the
;MSB
7794: CLC                ;Clear the carry for numeric
7795: RTS                ;And exit
7796: JMP  $7D28      ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
* This routine acts on the string that follows the
* SPRSAV command.
*
* NOTE: When the routine exits, the accumulator will
* contain the length of the string and the
* X and Y registers the LSB and MSB of the
* string's address.
*
*****

```

```

7799: LDA $66 ;Get the address of
779B: LDY $67 ;The string's descriptor
779D: JSR $87E0 ;And delete it from the temporary string stack
77A0: LDY #$00 ;Zero the index pointer
77A2: JSR $42E7 ;Get the length of the string from its
77A5: TAX ;Descriptor, and save it in the X register
77A6: INY ;Increment the index to the next byte
77A7: JSR $42E7 ;Get the LSB of the string's address
77AA: PHA ;And save it onto the stack
77AB: INY ;Increment the index to the next byte
77AC: JSR $42E7 ;Get the MSB of the string's address
77AF: TAY ;And place it in the Y register
77B0: PLA ;Place the LSB in the accumulator
77B1: SEC ;Set the carry to indicate a string
77B2: RTS ;Exit the routine

```

```

*****
*
* BASIC FAST COMMAND
*
* Command syntax: FAST
*
* This command allows you to speed up the operation
* of the 128 by increasing the clock speed to 2 MHz.
*
* NOTE: When the FAST command is executed, the 40
* column screen is automatically turned off
* (blanked).
*
*****

```

```

77B3: JSR $A845 ;Enable BANK 15
77B6: LDA $D011 ;Get VIC register 17 configuration
77B9: AND #$6F ;Turn the 40 column screen off
77BB: STA $D011 ;Reconfigure the register

```

```

77BE: LDA #1 ;Set the clock speed to 2MHz (FAST MODE)
77C0: STA $D030 ;Set the speed to FAST MODE
77C3: RTS ;Exit the routine

```

```

*****
*
* BASIC SLOW COMMAND
*
* Command syntax: SLOW
*
* This command allows you to return to the slow mode
* by decreasing the clock speed to 1 MHz.
*
* NOTE: When the SLOW command is executed, the 40
* column screen is automatically turned back on.*
*
*****

```

```

77C4: JSR $A845 ;Enable BANK 15
77C7: LDA #0 ;Set the clock speed to 1MHz (SLOW MODE)
77C9: STA $D030 ;Set the speed to SLOW MODE
77CC: LDA $D011 ;Get VIC register 17 configuration
77CF: AND #$7F ;Clear the raster line MSB
77D1: ORA #$10 ;Set bit 4 to turn the 40 column screen on
77D3: STA $D011 ;Reconfigure the register
77D6: RTS ;Exit the routine

```

```

*****
*
* FRMNUM
*
* FoRMula evaluate with the result
* of NuMeric expression
*
* This routine will evaluate the numeric expression by
* calling FRMEVL and then ensure that the result is
* actually numeric and not string. If the expression
* resulted in a string, a 'TYPE MISMATCH' error is
* generated.
*
*****

```

```

77D7: JSR $77EF ;Evaluate expression

```

```

*****
*
*                               *
*               CHKNUM         *
*                               *
*               CheCk for NUMeric *
*                               *
* This routine checks the VALTYP ($OF) to ensure that *
* the current data type is numeric. If the data type *
* is not numeric, then a 'TYPE MISMATCH' error is *
* generated. *
*                               *
*****

```

```

77DA: CLC                ;Flag for NUMERIC
77DB: BCC    $77DE        ;Unconditional branch past the check for string

```

```

*****
*
*                               *
*               CHKSTR         *
*                               *
*               CheCk for STRing *
*                               *
* This routine checks the VALTYP ($OF) to ensure that *
* the current data type is a string. If the data type *
* is not a string, then a 'TYPE MISMATCH' error is *
* generated. *
*                               *
*****

```

```

77DD: SEC                ;Flag for STRING
77DE: BIT    $OF          ;Is the value a string?
77E0: BMI    $77E5        ;If it is a string, then branch

```

```

*-----*
*
* If the carry flag was set to indicate the value was a*
* string but the VALTYP ($OF) indicates that the value *
* is numeric, a 'TYPE MISMATCH' error is generated. *
*
*-----*

```

```

77E2: BCS    $77E7        ;Generate a 'TYPE MISMATCH' error
77E4: RTS                                ;Exit the routine
77E5: BCS    $77E4        ;If the search is for a string, then branch
77E7: LDX    #22          ;Error number for a 'TYPE MISMATCH' error
77E9: .BYTE $2C          ;Mask to fall thru to $77EC to bypass the next
                          ;error

```

---

```
77EA: LDX  #$19      ;Error number for a 'FORMULA TOO COMPLEX' error
77EC: JMP  $4D3C     ;Jump to the error message routine
```

```
*****
*
*                      FRMEVL                      *
*
*                      FoRMula EVaLuation          *
*
* This routine is used by many other routines that *
* require the processing of an expression. This    *
* routine's main purpose is to take the BASIC     *
* ASCII text of the expression and divide it into its *
* individual parts, check those parts for errors,  *
* perform the individual operations, and combine the *
* results to obtain a single value that can be used *
* by the BASIC program.                           *
*
*****
```

```
77EF: LDX  $3D      ;Get the LSB of the BASIC TXTPTR
77F1: BNE  $77F5     ;If it is not equal to zero, then branch
77F3: DEC  $3E      ;Decrement the MSB of TXTPTR
77F5: DEC  $3D      ;Decrement the LSB of TXTPTR
77F7: LDX  #0       ;Set the priority code to zero initially
77F9: .BYTE $2C     ;MASK (bit)
77FA: PHA           ;If not masked, then save the MSB of
                  ;TXTPTR onto the stack
77FB: TXA           ;Get the LSB of TXTPTR into the accumulator
77FC: PHA           ;And save it onto the stack
77FD: TSX          ;Transfer the stack pointer into the X register
77FE: CPX  #$63     ;If the stack pointer is less than $63,
7800: BCC  $77EA     ;Then generate a 'FORMULA TOO COMPLEX' error
7802: JSR  $78D7     ;Get the next character and convert it to
                  ;Floating Point format
7805: LDA  #$00
7807: STA  $4F
7809: JSR  $0386     ;Get the character that was last accessed
780C: SEC          ;Set the carry for subtraction
780D: SBC  #$B1     ;Subtract the token value for greater than '>'
780F: BCC  $7828     ;If it is less than the value for '>', then
                  ;branch
7811: CMP  #$03     ;If it is greater than the
7813: BCS  $7828     ;Token value for less than '<', then branch
7815: CMP  #$01     ;Is it the token value for equals '='
7817: ROL
```

---

```

7818: EOR   #$01      ;Make the bits equal (0 = '>', 1 = '=', and
                    ;2 = '<')
781A: EOR   $4F      ;Add it to the existing comparison flag
781C: CMP   $4F      ;If the comparison is used more than once,
781E: BCC   $7881    ;Then generate a 'SYNTAX' error
7820: STA   $4F      ;Save the comparison flag (MASK)
7822: JSR   $0380    ;Get the next character
7825: JMP   $780C    ;And restart the loop
7828: LDX   $4F      ;Get the comparison MASK
782A: BNE   $7858    ;If it is not equal to zero, then branch
782C: BCS   $78AC    ;If the character is a letter and not
                    ;A digit, then branch

782E: ADC   #$07      ;Add the index
7830: BCC   $78AC    ;And move FAC1 to FAC2
7832: ADC   $0F      ;Add the string flag
7834: BNE   $7839
7836: JMP   $870D    ;Add the two strings together
7839: ADC   #$FF      ;Add the index
783B: STA   $24      ;Save it
783D: ASL           ;Multiply by 2
783E: ADC   $24      ;And add once to make a total of * 3
7840: TAY           ;Use the value as an index
7841: PLA           ;Get the accumulator back off of the stack
7842: CMP   $4828,Y  ;And compare it to the priority code
7845: BCS   $78B1    ;If the value is higher than the priority
                    ;Code, then branch

7847: JSR   $77DA    ;Check to see if it is numeric and if it is not,
                    ;Then generate a 'TYPE MISMATCH' error
784A: PHA           ;Place the priority code onto the stack
784B: JSR   $7871    ;Place the operator address onto the stack
784E: PLA           ;Restore the priority code
784F: LDY   $4D      ;Get the index to the operator
7851: BPL   $786A    ;Exit if done
7853: TAX           ;Exit if the
7854: BEQ   $78AF    ;Priority code equals zero
7856: BNE   $78BA    ;If it does not equal zero, then process it
7858: LSR   $0F      ;Set the pointer for process
785A: TXA           ;Move the priority code back to the accumulator
785B: ROL           ;And multiply by two to obtain an index
785C: LDX   $3D      ;Subtract
785E: BNE   $7862    ;One from
7860: DEC   $3E      ;TXTPTR
7862: DEC   $3D
7864: LDY   #$1B     ;Load the Y register with the offset priority
                    ;code flag

7866: STA   $4F      ;Save the operator code
7868: BNE   $7841    ;Restart the loop

```

---

```

786A: CMP    $4828,Y    ;Compare the value in the accumulator with
                        ;The priority code
786D: BCS    $78BA      ;If it is equal to or greater than, then exit
786F: BCC    $784A      ;Continue to loop
7871: LDA    $482A,Y    ;Save the address
7874: PHA                    ;Of the operator
7875: LDA    $4829,Y    ;Onto the
7878: PHA                    ;Stack
7879: JSR    $7884      ;Round off the left operand
787C: LDA    $4F        ;Get the comparison code ( <, =, > )
787E: JMP    $77FA      ;And evaluate the expression
7881: JMP    $796C      ;Generate a 'SYNTAX' error
7884: LDA    $68        ;Get the sign of FAC1
7886: LDX    $4828,Y    ;Get the priority code of the operand
7889: TAY                    ;Save the sign of FAC1 in the Y register
788A: CLC                    ;Clear the carry flag
788B: PLA                    ;Save the return
788C: ADC    #$01       ;Address of the calling
788E: STA    $24        ;Routine into locations
7890: PLA                    ;$24, $25
7891: ADC    #$00       ;The values are pulled
7893: STA    $25        ;Off the stack
7895: TYA                    ;Move the sign of FAC1 into the accumulator
7896: PHA                    ;And save the sign onto the stack
7897: JSR    $8C47      ;Round off FAC1
789A: LDA    $67        ;Save M4
789C: PHA                    ;Onto the stack
789D: LDA    $66        ;Save M3
789F: PHA                    ;Onto the stack
78A0: LDA    $65        ;Save M2
78A2: PHA                    ;Onto the stack
78A3: LDA    $64        ;Save M1
78A5: PHA                    ;Onto the stack
78A6: LDA    $63        ;Save the exponent
78A8: PHA                    ;Onto the stack
78A9: JMP    ($0024)    ;And JUMP to the calling routine
78AC: LDY    #$FF      ;Load the Y register with the flag for negative
                        ;(minus)
78AE: PLA                    ;Get the priority code from the stack
78AF: BEQ    $78D4      ;If it is equal to zero, then exit
78B1: CMP    #$64      ;Compare it with the priority code for
                        ;less than '<'
78B3: BEQ    $78B8      ;If it is the priority code for '<', then branch
78B5: JSR    $77DA      ;Check to see if the value is numeric
78B8: STY    $4D        ;
78BA: PLA                    ;Get the string flag from the stack
78BB: LSR                    ;Shift it to the left

```



```

78BC: STA $14
78BE: PLA ;Transfer the values
78BF: STA $6A ;For FAC1
78C1: PLA ;That are on
78C2: STA $6B ;The stack
78C4: PLA ;Into FAC2
78C5: STA $6C
78C7: PLA
78C8: STA $6D
78CA: PLA
78CB: STA $6E
78CD: PLA
78CE: STA $6F
78D0: EOR $68
78D2: STA $70
78D4: LDA $63
78D6: RTS ;Exit
    
```

```

*****
*
*                               EVAL
*
* This routine is used to convert a single arithmetic
* term from ASCII to its Floating Point equivalent.
* If the term was a variable, then the value of the
* variable is returned to FAC1. If the term was an
* ASCII digit, the digit is converted to its Floating
* Point equivalent and stored in FAC1. Upon entry to
* this routine, TXTPTR must point to the ASCII
* number to be converted.
*
*****
    
```

```

78D7: JMP ($030A) ;JUMP thru the EVAL vector at $030A ($78DA)
78DA: LDA #0 ;Set the VALTYP ($0F)
78DC: STA $0F ;To numeric
78DE: JSR $0380 ;Get the next character
78E1: BCS $78E8 ;And if it is a letter, then branch
78E3: LDX #0 ;Set the flag to obtain the value from RAM
;BANK 0
78E5: JMP $8D22 ;Convert the ASCII numerical string to
;A Floating Point number in FAC1
78E8: JSR $7B3C ;Check to see if the character is a letter
78EB: BCC $78F0 ;If the character is not a digit, then branch
78ED: JMP $7978 ;Get the numeric value of the digit
78F0: CMP #$FF ;Is the value the code for PI?
78F2: BNE $7903 ;If it is not the code for PI, then branch
    
```

```

78F4: LDA  #$FE      ;Load the accumulator and the Y register with
78F6: LDY  #$78      ;The LSB and MSB of the address that points to
                          ;The value for PI
78F8: JSR  $8BD4     ;Move the five byte floating point value of  $\pi$ 
                          ;Into FAC1
78FB: JMP  $0380     ;Get the next character

```

```

*****
*
*                               PIVAL
*
*                               PI VALue
*
* This is the five byte Floating Point value for  $\pi$ .
*
*****

```

```

          EX M1 M2 M3 M4
          -- -- -- -- --
78FE: .BYTE $82,$49,$0F,$DA,$A1 ;PI = 3.14159265
7903: CMP  #'.'      ;Is it a decimal point?
7905: BEQ  $78E5     ;If it is, then branch to convert ASCII to
                          ;Floating Point
7907: CMP  #'-'      ;Is it a minus sign '-' for subtraction?
7909: BEQ  $7971     ;If it is, then branch
790B: CMP  #'+'      ;Is it a plus sign '+' for addition?
790D: BEQ  $78DE     ;If it is, then branch
790F: CMP  #'"'      ;Is it a quote mark '"'?
7911: BNE  $7928     ;If it is not a quote mark, then branch to check
                          ;For numeric functions
7913: LDA  $3D       ;Get the
7915: LDY  $3E       ;TXTPTR
7917: ADC  #$00      ;And add the carry if any
7919: BCC  $791C     ;If no overflow occurred then branch
791B: INY
                          ;Else add one to the MSB of TXTPTR
791C: JSR  $869A     ;Set up the pointers to a string in memory
791F: LDX  $72       ;Get the LSB and the MSB
7921: LDY  $73       ;Of the address of the string
7923: STX  $3D       ;And save it to
7925: STY  $3E       ;TXTPTR
7927: RTS
                          ;Exit

```

```

*****
*
*                               CHKNOT
*
*                               CHecK for NOT token
*

```

```

* This routine checks the accumulator for the BASIC      *
* NOT token and if it's found then the BASIC NOT      *
* command is executed ($7930). If it's not found then *
* this routine branches to CHKFN.                      *
*                                                       *
*****

```

```

7928: CMP   #$A8      ;Is the value the token for 'NOT'?
792A: BNE   $7942     ;If it is not the token for 'NOT', then branch
                        ;To check for the FN token
792C: LDY   #$18      ;Load the Y register with the offset of priority
                        ;codes
792E: BNE   $7973     ;For the NOT command

```

```

*****
*                                                       *
*           BASIC NOT FUNCTION (OPERATOR)              *
*                                                       *
* The NOT operator returns the 'TWO'S COMPLEMENT' of an *
* integer number. For example PRINT NOT 10 would return *
* a value of -11. This routine will invert the number   *
* and adds one to it and places the result in FAC1 in  *
* Floating Point format.                                *
*                                                       *
*****

```

```

7930: JSR   $84B4     ;Convert the Floating Point number
                        ;To signed integer format
7933: LDA   $67       ;Get the twos complement
7935: EOR   #$FF      ;Of the LSB
7937: TAY                       ;And place it into the Y register
7938: LDA   $66       ;Get the twos complement
793A: EOR   #$FF      ;Of the MSB

```

```

*****
*                                                       *
*           GIVAYF                                     *
*                                                       *
* This routine converts a signed 16 bit integer to a   *
* Floating Point number. Enter with the MSB in the   *
* accumulator the LSB in the Y register of the value. *
*                                                       *
*****

```

```

793C: JSR   $84E5     ;Set up for the conversion
793F: JMP   $8C70     ;Convert the signed integer to a Floating Point
                        ;number

```

```

*****
*
*                               *
*                               *
*                               *
*                               *
*                               *
*                               *
*****

7942:  CMP    #$A5      ;Is it the token for the 'FN" function?
7944:  BNE    $7949     ;If it is not the token for 'FN', then branch
7947:  JMP    $853B     ;JUMP to the routine that handles the 'FN' token

*****
*
*                               *
*                               *
*                               *
* This routine will check the accumulator for the token*
* value for SGN and if then token value is less than *
* the value for SGN then the expression enclosed in *
* parenthesis is evaluated, and if the token value is *
* equal to or greater then this routine calles $4BF7 to *
* text for the string functions. *
* *
*****

7949:  CMP    #$B4      ;If the value is less than the token for 'SGN',
794B:  BCC    $7950     ;Then branch to evaluate the expression
                          ;Between parentheses
794D:  JMP    $4BF7     ;Check for a string function

*****
*
*                               *
*                               *
*                               *
*                               *
*                               *
* This routine evaluates the expression within the *
* parentheses, by first calling CHKOPN to check for *
* an opening parenthesis, and then calling FRMEVL to *
* evaluate the expression inside the parentheses. *
* *
*****

7950:  JSR    $7959     ;CHKCLS Check for/ skip the opening parenthesis
                          ;If the opening parenthesis is not found,
                          ;Then generate a 'SYNTAX' error
7953:  JSR    $77EF     ;FRMEVL Evaluate the expression

```

```
*****
*
* Note: All the following routines will check for the
* specified character and if it is found, the routine
* will get the next character following it via CHRGET.
*
*****
```

```
*****
*
*                               CHKCLS
*
*       CHecK CLoSing parenthesis and skip
*
* This routine checks for and skips the closing paren-
* thesis and generates a 'SYNTAX' error if not found.
*
*****
```

```
7956: LDA  #' ) '      ;Check for the closing parenthesis
7958: .BYTE $2C      ;MASK to fall through to $795E
```

```
*****
*
*                               CHKOPN
*
*       CHecK OPeNing Parenthesis and skip
*
* This routine is used to check for an opening
* parenthesis and skip it. However, if one is not
* found, then a 'SYNTAX' error is generated.
*
*****
```

```
7959: LDA  #' ( '      ;Check for opening parenthesis
795B: .BYTE $2C      ;MASK to fall through to $795E
```

```
*****
*
*                               CHKCOM
*
*       CHecK for a COMma and skip
*
* This routine is used to check for a comma and if it
* is not found, then a 'SYNTAX' error is generated.
*
*****
```

```
795C: LDA #' , ' ;Check for a comma
```

```
*****
*
*                               CHKACC
*
*                               CHecK ACCumulator
*
*   If this routine is entered here, you must have the
*   value you wish to check for in the accumulator.
*   The routine will check for that value and generate
*   a 'SYNTAX' error if it is not found.
*
*****
```

```
795E: LDY #0 ;Zero the index pointer
7960: STA $79 ;Save the character we are searching for
7962: JSR $03C9 ;Get the next character
7965: CMP $79 ;Is it the same as the character we're looking
;for?
7967: BNE $796C ;If not, then branch to create a 'SYNTAX' error
7969: JMP $0380 ;If there is a match, then skip the character
;And get the next character
```

```
*****
*
*                               SNERR
*
*                               SyNtax ERRor generator
*
*   This routine is used to generate a 'SYNTAX ERROR'
*   message.
*
*****
```

```
796C: LDX #11 ;Error number for a 'SYNTAX' error
796E: JMP $4D3C ;JUMP to the error message routine

7971: LDY #$15 ;Load the Y register with the priority code
;For unary minus (Negate the Exponent)
7973: PLA ;Pull the return address of the calling routine
7974: PLA ;Off of the stack
7975: JMP $784B ;Evaluate the operand
```

```
*****
*
*                               ISVAR                               *
*
* This routine will get the value of a variable.                    *
*
*****
```

```
7978: JSR  $7AAF      ;Search for the variable's descriptor
797B: STA  $66        ;Save its address in
797D: STY  $67        ;FAC1
797F: LDX  $47        ;Get the first character of the variable name
7981: LDY  $48        ;Get the second character of the variable name
7983: LDA  $0F        ;Get the flag for numeric ($00), or string ($FF)
7985: BEQ  $79EA     ;If the value is numeric, then branch to check
                          ;For a system variable
7987: LDA  #0         ;Set the flag for string
7989: STA  $71        ;String comparison
```

```
*****
*
*                               CHKTI$                             *
*
*                               CHecK for Time $                   *
*
* This routine performs two checks. One of the checks *
* is to ensure that the variable name is for TI$. If *
* the variable name is TI$, then the second check *
* is made to ensure that PTRGET sets its descriptor *
* to the 'Dummy' address of $03D2. The routine then *
* returns the value for TI$. If the first character *
* of the variable name is not a 'T', then the routine *
* exits by branching to the routine that checks for *
* the variable name of DS. *
*
*****
```

```
798B: CPX  #'t'      ;Check to see if the first character is a 't'
                          ;for 'TI$'
798D: BNE  $79B4     ;If it is not, then branch to check for DS$
798F: CPY  #'I'      ;Check to see if the second character is an 'I'
7991: BNE  $79B3     ;If it is not, then branch to exit
7993: LDA  $66        ;Get the LSB of the descriptor
7995: CMP  #$D2      ;Address is the dummy flag for TI?
7997: BNE  $79B3     ;No, then exit
7999: LDA  $67        ;Get the descriptor's MSB
799B: CMP  #$03      ;Is it the dummy flag MSB?
```

```

799D: BNE $79B3 ;If not, then exit
799F: JSR $7A1A ;Read the time
79A2: STY $60 ;Zero $60
79A4: DEY ;Decrement the Y register thereby, generating a
;Resultant value of $FF
79A5: STY $72 ;Store $FF into $72 to be used as an index value
79A7: LDY #6 ;String length of six characters for TI$
79A9: STY $5F ;Save it
79AB: LDY #36 ;Index over 36 bytes to
79AD: JSR $8ECD ;Convert the time into a string
79B0: JMP $85B8 ;Give it a temporary descriptor
79B3: RTS ;Exit the routine

```

```

*****
*
*                               CHK DS$
*
*                               CHeck for Disk Status $
*
* This routine checks to see whether the variable's
* name is DS$. If it is not TI$ (above) or DS$, then
* then the routine exits, because, the variable is not
* a system variable. If the variable name is DS$,
* then the routine calls OBTDS in order to handle the
* DS$ assignment. The routine then allocates the
* string in RAM BANK 1 to prevent the garbage
* collection routine from deallocating the string.
*
*****

```

```

79B4: CPX #'d' ;Is the first character a 'd'
79B6: BNE $79B3 ;If not, then exit the routine
79B8: CPY #'S' ;Is the second character an 'S'
79BA: BNE $79B3 ;If it is not DS, then exit
79BC: JSR $79E3 ;Handle the updating of DS$ so that the
79BF: LDY #$FF ;Nnext 'INY' will zero the index pointer
79C1: INY ;Increment the index pointer
79C2: LDA #$7B ;Get a byte from the string assigned
79C4: JSR $03AB ;To DS$ (pointed to by $7B,$7C)
79C7: CMP #$00 ;Delimiter of zero?
79C9: BNE $79C1 ;If not, keep counting the number of characters
79CB: TYA ;Save the length of DS$ into the accumulator
79CC: JSR $8688 ;Create the DS$ descriptor in $63, $64, $65
79CF: TAY ;Move the length back into the accumulator
79D0: BEQ $79E0 ;Branch to move the descriptor
79D2: DEY ;Decrement the length of the string
79D3: LDA #$7B ;Get a byte from the string

```



```

79D5: JSR    $03AB      ;That is pointed to by locations $7B, $7C
79D8: STA    ($37),Y    ;Move it into the new string area
79DA: TYA                      ;Move the length into the accumulator
79DB: BNE    $79D2      ;If not equal to zero, then continue
79DD: JSR    $8771      ;Add the length of the string to its address
79E0: JMP    $86E3      ;Move the descriptor to the temporary string
                          ;stack
    
```

```

*****
*
*
*           OBTD$$
*
*           OBTain DS$
*
*
* This routine checks location $7A to see if any
* BASIC Serial Bus routines have been used, such as,
* OPEN, DCLOSE, etc... since the last time that DS$
* was updated. If location $7A does not equal zero,
* then no BASIC Serial Bus routines have been used
* since the last time that DS$ was updated.
*
*
* NOTE: Always remember that any Serial Bus access
* will deallocate DS$. Therefore, be sure to as-
* sign DS$ to another variable temporarily.
* For example, A$ = DS$.
*
*****
    
```

```

79E3: LDA    $7A        ;Disk Drive operation since last update?
79E5: BNE    $7A27      ;If not, then branch to exit
79E7: JMP    $A778      ;Update DS$
    
```

```

*-----*
*
* This routine checks to see if the numeric variable
* is an Integer or a Floating Point Number.
*
*-----*
    
```

```

79EA: BIT    $10        ;Does INTFLG indicate that the variable is FP?
79EC: BPL    $79FD      ;If not, then branch to handle an integer
    
```

```

*****
*
*                      OBT INT                      *
*
*                      OBTain INTEger              *
*
* This routine will get the two byte integer value *
* that is pointed to by locations $66, $67 and    *
* convert it to Floating Point Format in FAC1.    *
*
* NOTE: The integer value must be located in RAM *
*        BANK 1.                                  *
*
*****

```

```

79EE:  LDY   #$00           ;Zero the index pointer
79F0:  JSR   $42E7          ;Get the first character (digit) from DS$
79F3:  TAX                   ;Save the first character in the X register
79F4:  INY                   ;Increment the index pointer
79F5:  JSR   $42E7          ;Get the second character (digit) from DS$
79F8:  TAY                   ;Move it into the Y register
79F9:  TXA                   ;Move the first character into the accumulator
79FA:  JMP   $793C          ;Convert it to integer format

```

```

*-----*
*
* This routine checks the variable's descriptor *
* address to see if PTRGET at $7AAF has set the *
* address to $03D2 which indicates that this could be *
* a system variable. If the address is $03D2, then *
* the variable name that is in the X and Y registers *
* (X = first character, Y = second character) is *
* checked for a match against the system variables to *
* find out which of the system variables it is. When *
* a match has been found, the routine handles the *
* system variable accordingly. However, if the *
* 'Dummy' variable is not found, then the routine *
* exits.
*
*-----*

```

```

79FD:  LDA   $67             ;Get the MSB of the descriptor's address
79FF:  CMP   #$03             ;Is it the dummy address?
7A01:  BNE   $7A81          ;If not, then branch to move the FPN to FAC1
7A03:  LDA   $66             ;Get the LSB of the descriptor
7A05:  CMP   #$D2            ;Is it the dummy variable flag?
7A07:  BNE   $7A81          ;If not, then it is not TI

```

```

*****
*
*                               CHKTI
*
*                               CHeck for TIme
*
* This routine checks for the system variable TI
* (Floating Point Format).  If the variable is found
* to be TI, then the routine will read the current
* time and convert it to Floating Point Format in FAC1.
*
*****

```

```

7A09: CPX  #'t'      ;Is it 't' as in TI?
7A0B: BNE  $7A28    ;If it is not a 't', then do not check for the
                ;'I', just branch
7A0D: CPY  #'I'      ;Is it an 'I' as in TI?
7A0F: BNE  $7A81    ;If not, then branch since there are more
                ;Variable names that are possible starting with
                ;'t' and therefore the variable must be
                ;Processed as a normal variable
7A11: JSR  $7A1A    ;It is TI so GETTIM
7A14: TYA                ;Zero the Y register and the Accumulator
7A15: LDX  #$A0     ;Exponent
7A17: JMP  $8C7B    ;Convert TI to a Floating Point Number

```

```

*****
*
*                               GETTIM
*
*                               GET TIME
*
* This routine reads the system clock by calling
* RDTIM at $FFDE and then stores the result at
* locations $65, $66, and $67.
*
*****

```

```

7A1A: JSR  $FFDE    ;Get the time
7A1D: STX  $66     ;Save the system clock value Mid
7A1F: STY  $65     ;High
7A21: STA  $67     ;Low
7A23: LDY  #$00    ;Clear
7A25: STY  $64     ;M1
7A27: RTS                ;Exit the routine

```

```

*****
*
*                               CHKST
*
*                               CHECK for SStatus
*
* This routine will check the X and Y registers
* which contain the first and second character of the
* variable name to obtain the value for.  If the
* variable's name is the Floating Point Variable ST,
* then the routine will return the value for ST.
* However, if the variable's name is not ST, then
* the routine branches to check for another system
* variable or to process the variable as a normal
* variable.
*
*****

```

```

7A28: CPX   #'s'           ;Is it an 's' as in ST
7A2A: BNE   $7A36         ;If not, then branch to check for DS
7A2C: CPY   #'t'           ;Since, it is an 's', is it a 'T'
7A2E: BNE   $7A81         ;If not, then branch to process the variable as
                          ;A normal variable
7A30: JSR   $FFB7         ;Get the I/O STATUS and assign it to ST
7A33: JMP   $8C68         ;Change the byte in the accumulator to Floating
                          ;Point format and leave it in FAC1

```

```

*****
*
*                               CHKDS
*
*                               CHECK Disk Status
*
* This routine takes the first two characters from DS$
* and convert them to a Floating Point Number in FAC1.
*
* NOTE: DS% is a valid variable that is not
*        associated with the disk drive and is
*        handled in a different manner.
*
*****

```

```

7A36: CPX   #'d'           ;Is it a 'D' as in DS
7A38: BNE   $7A60         ;If not, then branch to check for ER
7A3A: CPY   #'s'           ;Since it is a 'D', is it an 'S' as in DS
7A3C: BNE   $7A81         ;If it is not DS, then branch to process it as a
                          ;Normal variable

```

```

*****
*
*                               GETDS
*
*                               GET Disk Status
*
* This routine will update the new DS$ if it is
* required. Then it will convert the first two
* ASCII characters to a numeric value in FAC1.
*
*****

```

```

7A3E: JSR   $79E3      ;Handle DS$
7A41: LDY   #0         ;Set the indirect address
7A43: LDA   #$7B      ;For DS to $007B
7A45: JSR   $03AB     ;Get the first digit of DS
7A48: AND   #$0F      ;Mask off the upper nybble
7A4A: ASL                   ;Convert DS
7A4B: STA   $11       ;To numeric
7A4D: ASL                   ;And call
7A4E: ASL                   ;A routine
7A4F: ADC   $11       ;In the
7A51: STA   $11       ;BASIC SGN function
7A53: INY                   ;To convert
7A54: LDA   #$7B      ;The value
7A56: JSR   $03AB     ;To a Floating Point
7A59: AND   #$0F      ;Number
7A5B: ADC   $11       ;In FAC1
7A5D: JMP   $8C68     ;Convert it to a Floating Point Number in FAC1

```

```

*****
*
*                               CHKER
*
*                               CHecK for an ERror
*
* This routine checks for the system reserved word
* ER and if it is found, then the routine branches
* to GETER below.
*
*****

```

```

7A60: CPX   #'e'      ;Is it an 'E' as in ER
7A62: BNE   $7A81     ;If not, then process it as a normal variable
7A64: CPY   #'r'      ;Since it is an 'E', is it and 'R' as in ER
7A66: BEQ   $7A78     ;If it is, then branch
7A68: CPY   #'l'      ;Since it is an 'E', is it an 'L' as in EL

```

7A6A: BNE \$7A81 ;If not, then process it as a normal variable

```

*****
*
*                               GETEL
*
*                               GET Error Line
*
* This routine will take the line number the error
* occurred on and convert it to Floating Point Format
* in FAC1.
*
*****

```

7A6C: STA \$FF03 ;Enable Bank 14  
 7A6F: LDA \$120A ;Get the line number that the error occurred  
 7A72: LDY \$1209 ;On from ERRLIN  
 7A75: JMP \$84C9 ;And convert it to Floating Point Format in FAC1

```

*****
*
*                               GETER
*
*                               GET Error
*
* This routine will take the BASIC ERROR number from
* ERNUM ($1208) and convert it to Floating Point
* Format in FAC1.
*
*****

```

7A78: STA \$FF03 ;Enable BASIC BANK 14  
 7A7B: LDA \$1208 ;Get the error number from ERRNUM  
 7A7E: JMP \$8C68 ;And convert the value to a Floating Point  
 ;Number in FAC1

```

*****
*
* This routine is used to get a Floating Point value
* from BANK 1 and move it to FAC1. If entered here,
* the address of the floating point value in BANK 1
* must be stored in locations $66, $67 (LSB,MSB).
*
*****

```

7A81: LDA \$66 ;Get the LSB  
 7A83: LDY \$67 ;And the MSB of the address

```

*****
*
*                               MOVFRM
*
*           MOVE a Floating point numbeR from Memory
*
* If this routine is entered here, the accumulator
* and the Y register must hold LSB and MSB
* respectively of the address in BANK 1 of the
* Floating Point value that you wish to move to FAC1.
*
* NOTE:  The address in the X and Y registers must be
*         pointing to nominal byte 2, the exponent.
*
*****

```

```

7A85: STA  $24      ;Save the LSB
7A87: STY  $25      ;And the MSB of the address
7A89: LDY  #$00     ;Starting index of zero
7A8B: JSR  $03B7    ;Get the exponent of the value in memory
7A8E: STA  $63      ;Save it in FAC1
7A90: STY  $71      ;Clear the rounding byte
7A92: INY                ;Move to the next value
7A93: JSR  $03B7    ;Get M1 from memory
7A96: STA  $68      ;Save as the sign of FAC1
7A98: ORA  #$80     ;Ensure that bit 7 is set
7A9A: STA  $64      ;Save it in FAC1
7A9C: INY                ;Move to the next value
7A9D: JSR  $03B7    ;Get M2 from memory
7AA0: STA  $65      ;Save in FAC1
7AA2: INY                ;Move to next value
7AA3: JSR  $03B7    ;Get M3 from memory
7AA6: STA  $66      ;Save in FAC1
7AA8: INY                ;Move to the next value
7AA9: JSR  $03B7    ;Get M4 from memory
7AAC: STA  $67      ;Save in FAC1
7AAE: RTS                ;Exit

```

```

*****
*
*                               PTRGET
*
*                               PoinTeR GET
*
* Note:  The Basic DIM command enters PTRGET at $7AB4
* to prevent the flag for 'Don't Dim' from being set.
* The X register and accumulator contain the first

```

```

* character of the variable name. This routine will *
* take the character that TXTPTR is pointing to and *
* place it in VARNAM ($47) as the first character in *
* the variable name. Then the routine will ensure *
* that the character is an ASCII letter A-Z and if it *
* is not, a 'SYNTAX' error message will be generated. *
* If the character is an ASCII letter, various zero *
* page flags are set to indicate to the system NOT TO *
* DIMension an array because, this variable is numeric *
* and a Floating Point Number (FPN). The routine will *
* then index over to the next character to obtain the *
* second character of the variable name that is to *
* be located or created (now pointed to by TXTPTR). *
* The second character is then placed into VARNAM + 1 *
* ($47). After this has been accomplished, the routine *
* will check to see if the character following the *
* first or second character of the variable name is *
* the flag for string ($), or integer '%'. If neither *
* symbol is present, a floating point number is *
* assumed to be present. If one of the symbols is *
* present, various zero page flags as well as bit *
* seven of one or both characters of the variable *
* name in VARNAM ($47,$48) are set. If the character *
* following the flag denoting the variable type is *
* an opening parenthesis, then location $12 is tested *
* to see if arrays are allowed. If they are not *
* allowed, then the index value in parentheses is *
* ignored and the previous variable name is used. If *
* arrays are allowed, then ISARY is called to DIM *
* the array. The check on arrays is performed because, *
* the DEFINE FuNction command does not allow arrays *
* in its syntax. If an array is not going to be *
* processed, this routine will fall through to *
* PTRGET1 ($7BOB). *
* *
*****

```

```

7AAF: LDX #0 ;Flag for Don't DIM
7AB1: JSR $0386 ;Get the first character of the variable name
7AB4: STX $0E ;Set the flag for Don't DIM
7AB6: STA $47 ;Save the first character of the variable name
7AB8: JSR $0386 ;This will return the current character that
;TXTPTR is Pointing to. This JSR is here in case
;the Basic Dim Command called this routine
7ABB: JSR $7B3C ;Check for an ASCII letter
7ABE: BCS $7AC3 ;If it is a letter, then branch
7AC0: JMP $796C ;Generate a 'SYNTAX' error message

```



---

```

7AC3: LDX    #0           ;Flag to
7AC5: STX    $0F         ;Indicate numeric and not a string
7AC7: STX    $10         ;Floating point number not an integer
7AC9: JSR    $0380       ;Get the next character
7ACC: BCC    $7AD3       ;If it is a DIGIT 0 - 9, then branch
7ACE: JSR    $7B3C       ;Check to make sure it is a letter
7AD1: BCC    $7ADE       ;If it is not a letter, then branch
7AD3: TAX                    ;Save the second character of the variable name
                          ;Into the X register
7AD4: JSR    $0380       ;Get the next character
7AD7: BCC    $7AD4       ;If it is a DIGIT 0 - 9, then branch
7AD9: JSR    $7B3C       ;Check to make sure it is a letter
7ADC: BCS    $7AD4       ;If it is a letter, then skip over the rest
                          ;Of the variable name
7ADE: CMP    #'$'        ;Is it the symbol ($) for a string?
7AE0: BNE    $7AE8       ;If it is not, then branch
7AE2: LDA    #$FF        ;Flag for a string
7AE4: STA    $0F         ;Set the flag
7AE6: BNE    $7AF8       ;Branch past the test for an integer (%)
7AE8: CMP    #'%'        ;Is it an integer?
7AEA: BNE    $7AFF       ;If it is not, then branch to process a
                          ;Floating Point or an Array
7AEC: LDA    $12         ;Check SubFlag to see if integers are allowed
7AEE: BNE    $7AC0       ;If not, generate a 'SYNTAX' error
7AF0: LDA    #$80        ;Set the flag for an integer
7AF2: STA    $10         ;Set bit 7 in INTFLG
7AF4: ORA    $47         ;Set bit seven in the first character of the
7AF6: STA    $47         ;Variable name and save the character in VARNAM
7AF8: TXA                    ;Move the second character of variable name into
                          ;The Accumulator
7AF9: ORA    #$80        ;Set Bit 7 for strings and integers
7AFB: TAX                    ;Move the second character into the X register
7AFC: JSR    $0380       ;Get the next character
7AFF: STX    $48         ;Save the second character of variable name
7B01: SEC                    ;Set the carry for subtraction
7B02: ORA    $12         ;Combine the first character after the variable
                          ;type
                          ;Flag with the Integer allowed/not allowed flag
                          ;which
                          ;Will be checked later for an opening
                          ;parenthesis
7B04: SBC    #'('        ;If the character is the opening parenthesis
7B06: BNE    $7B0B       ;And ARRAYs are not allowed, then branch
7B08: JMP    $7CAB       ;Jump to ISARY to handle the ARRAY

```

```

*****
*
*                               PTRGET1
*
* This routine will use VARTAB to obtain the address
* of where the start of the variable descriptor table
* is located in Ram Bank 1. Unless you have changed
* it, this table is usually starts at $0400. The
* routine will use VARTAB as a starting position by
* placing it into locations $61, $62. The first two
* bytes of this variable descriptor are then checked
* against the variable name that is stored in
* locations $47, $48 to see if it is the one we are
* searching for. If it is not, then the routine will
* add seven to the address in $61, $62 to move to the
* next variable descriptor. The check is then made on
* this variable descriptor. The process continues
* until either the variable name is found or until
* locations $61, $62 equal ARYTAB. If locations $61,
* $62 equal ARYTAB, this indicates that all the
* variable descriptors have been checked and the
* variable name we were searching for was not found.
* Therefore, this routine will branch to the NOTFNS
* routine below to allocate room for the new variable
* that is in $47, $48. However, if during the search
* the variable name was found, then this routine will
* call the MOVVARPNT routine below to leave $61,$62
* pointing to nominal byte 2 of the variable descriptor*
* and place that address in $49,$4A (VARPNT).
*
*****

```

```

7BOB: LDY  #$00      ;Clear the Integer/Arrays
7BOD: STY  $12      ;Not allowed flag
7BOF: LDA  $2F      ;Get the address from VARTAB of
7B11: LDX  $30      ;Where the storage area of RAM BANK 1
7B13: STX  $62      ;Is for the VARIABLE descriptors
7B15: STA  $61      ;And save it in locations $61, $62
7B17: CPX  $32      ;Check the MSB of $62 against the MSB of ARYTAB
7B19: BNE  $7B1F    ;to see if all the descriptors have been
                          ;Searched through. If they have not, then branch
7B1B: CMP  $31      ;Check the LSB to see if ARYTAB = $61, $62
7B1D: BEQ  $7B46    ;If all of the descriptors have been searched,
                          ;Then branch to create a new descriptor
7B1F: JSR  $4300    ;Get the first character of the variable name
                          ;from the
                          ;Variable descriptor pointed to by $61, $62

```

```

7B22: CMP    $47           ;Is it the same character as the
7B24: BNE    $7B32        ;Variable name we are searching for?
                                ;If not, then branch
7B26: INY                                ;If the first character matchs move over to the
                                ;Second character of the variable name
7B27: JSR    $4300        ;Get the second character of the name from
                                ;The descriptor
7B2A: CMP    $48           ;Is it the same as the second character we are
                                ;Looking for?
7B2C: BNE    $7B31        ;If not, then move over to the next descriptor
7B2E: JMP    $7C57        ;If both characters of the variable name match,
                                ;Then branch to point VARPNT to nominal byte 2
                                ;Of the descriptor

```

```

*****
*
*                               PTRGET2
*
* This routine will add seven to $61, $62 to move it
* over to the first character of the variable name
* of the next variable descriptor.
*
*****

```

```

7B31: DEY           ;Move the index pointer back to the first
7B32: CLC           ;Character in the descriptor then
7B33: LDA    $61     ;Add seven to the current descriptor
7B35: ADC    #$07     ;Being checked to move us over
7B37: BCC    $7B15   ;To the next descriptor's first character of the
7B39: INX           ;Variable name then restart
7B3A: BNE    $7B13   ;By returning back to the main loop

```

```

*****
*
*                               PTRGET3
*
* This subroutine will check the character in the
* accumulator to see if it is an ASCII letter A - Z.
* If it is, the carry flag is set and if it is not an
* ASCII letter A - Z, the carry flag is cleared.
*
*****

```

```

7B3C: CMP    #'A'       ;Check to see if the character 'A' and if it
7B3E: BCC    $7B45        ;Is less than the character 'A', then branch
                                ;With the carry flag cleared
7B40: SBC    #'['        ;If the character is greater than 'Z',

```

```

7B42: SEC                ;Then the carry flag is cleared and the carry
7B43: SBC    #$A5        ;Flag is set for characters A thru Z
7B45: RTS                ;Exit this routine

```

```

*****
*
*                               *
*                   NOTFNS     *
*
*                   did NOT FiNd deScriptor
*
* This routine is used to CREATE a new variable
* descriptor as long as the calling routine is not
* the BASIC POINTER COMMAND, or the ISVAR routine.
*
*****

```

```

7B46: TSX                ;Place the stack pointer into the X register
7B47: LDA    $0102,X     ;Get the MSB of the Calling Routine
7B4A: CMP    #$83        ;Is it the Pointer Routine ($8305)
7B4C: BEQ    $7B52       ;If it is, then branch
7B4E: CMP    #$79        ;Is it the ISVAR Routine ($7978)
7B50: BNE    $7B7C       ;If it is not, then branch
7B52: LDA    #$D2        ;
7B54: LDY    #$03        ;Dummy return address to indicate TI$
7B56: RTS                ;Exit the routine

```

```

*-----*
*
*                               *
*                   NOTFNS2    *
*
*                   did NOT FiNd deScriptor2
*
*                   SEE THE COMMENTS UNDER NOTFNS1
*
*-----*

```

```

7B57: CPY    #'I'        ;Is it an 'I' as in TI$?
7B59: BEQ    $7B52       ;Yes, then generate a 'SYNTAX' error
7B5B: CPY    #'i'        ;Is it 'i' as in TI
7B5D: BNE    $7B90       ;No, then branch to create the new descriptor
7B5F: BEQ    $7B79       ;If it is, then generate a 'SYNTAX' error
7B61: CPY    #'S'        ;Is it 'S' as in DS?
7B63: BEQ    $7B79       ;Yes, then generate a 'SYNTAX' error
7B65: CPY    #'s'        ;Is it 's' as in DS?
7B67: BNE    $7B90       ;No, then branch to create the new descriptor
7B69: BEQ    $7B79       ;If it is, then generate a 'SYNTAX' error
7B6B: CPY    #'t'        ;Is it a 't' as in ST?

```

```

7B6D: BNE   $7B90      ;No, then branch to create the new descriptor
7B6F: BEQ   $7B79      ;If it is, then generate a 'SYNTAX' error
7B71: CPY   #'r'       ;Is it the 'r' as in ER or EL
7B73: BEQ   $7B79      ;If so, then branch to generate a 'SYNTAX' error
7B75: CPY   #'l'       ;Is it 'l' as in EL
7B77: BNE   $7B90      ;No, then branch to create the new descriptor
7B79: JMP   $796C      ;Generate a 'SYNTAX' error message
    
```

```

*****
*
*                               NOTFNS1
*
*                               did NOT FiNd deScriptor1
*
* This routine will ensure the variable name that a new *
* descriptor is to be created for is not one of the sys-*
* tem reserved variables (ER, EL, DS, DS$, ST, TI, or *
* TI$) and if it is one of the reserved variables (ex- *
* cept TI$ for which a DUMMY descriptor address ($03D2) *
* is created) a SYNTAX ERROR MESSAGE will result. If the*
* variable's name for which the new descriptor is to be *
* created is not one of the system's reserved variables,*
* then this routine will fall through to NOTFNS3.
*
*****
    
```

```

7B7C: LDA   $47        ;Get the variable name the new
7B7E: LDY   $48        ;Descriptor to be created for
7B80: CMP   #'t'       ;Is it a 't' as in TI?
7B82: BEQ   $7B57      ;If so, then check for the 'I'
7B84: CMP   #'s'       ;Is it an 's' as in ST?
7B86: BEQ   $7B6B      ;If so, then check for the 't'
7B88: CMP   #'e'       ;Is it an 'e' as in ER?
7B8A: BEQ   $7B71      ;If so, the check for the 'r' or the 'l'
7B8C: CMP   #'d'       ;Is it a 'd' as in DS?
7B8E: BEQ   $7B61      ;If so, then check for the 's'
    
```

```

*****
*
*                               NOTFNS3
*
*                               did NOT FiNd deScriptor3
*
* This routine actually creates the new descriptor for *
* the variables name which is in $47,$48 (VARNAM).
*
*****
    
```

---

```

7B90: LDA  $31      ;Move ARYTAB into $61,$62
7B92: LDY  $32      ;Note: ARYTAB is pointing to the end of
7B94: STA  $61      ;The variable
7B96: STY  $62      ;Descriptor (table)
7B98: LDA  $33      ;Move STREND into $5C,$5D
7B9A: LDY  $34      ;Note: STREND is pointing to the end of
7B9C: STA  $5C      ;The ARRAY storage area plus 1 and doubles as
7B9E: STY  $5D      ;The address the last string can be stored into
7BA0: CLC                ;Clear carry for subtraction
7BA1: ADC  #$07      ;Then add seven to STREND to make
7BA3: BCC  $7BA6     ;Room for the new variable descriptor
7BA5: INY                ;Which increases variable storage area by 7
                          ;And decreases room available to strings by 7
7BA6: STA  $5A      ;Save the address of the last byte of the
7BA8: STY  $5B      ;Descriptors in $5A, $5B
7BAA: JSR  $7C66     ;Shift the ARRAY storage
                          ;Area up seven bytes
7BAD: LDA  $5A      ;Get the address after
7BAF: LDY  $5B      ;The move
7BB1: INY                ;Increment the LSB by one
7BB2: STA  $31      ;Save it as the new ARYTAB
7BB4: STY  $32      ;Address
7BB6: STA  $5A      ;And save it in the
7BB8: STY  $5B      ;Temporary work area
7BBA: LDA  $5A      ;Check the
7BBC: LDX  $5B      ;New ARYTAB
7BBE: CPX  $34      ;Address with STREND
7BC0: BNE  $7BC8     ;To see if the
7BC2: CMP  $33      ;Descriptor exists
7BC4: BNE  $7BC8     ;If they are not the same, then branch
7BC6: BEQ  $7C40     ;Else place the variable name into the
                          ;descriptor
                          ;And fill nominal bytes 2 - 6 with zeros
7BC8: STA  $24      ;Save the address of the variable's
7BCA: STX  $25      ;Descriptor
7BCC: LDY  #$00      ;Zero the index value
7BCE: JSR  $03B7     ;Get the variable's name
7BD1: TAX                ;Save it into the X register
7BD2: INY                ;Move over to nominal byte 1
7BD3: JSR  $03B7     ;Get the second character of the variable's name
7BD6: PHP                ;Save the processor's status onto the stack
7BD7: INY                ;Move over to nominal byte 2
7BD8: JSR  $03B7     ;Get the address of the next variable
7BDB: ADC  $5A      ;Descriptor and add it to ARYTAB to
7BDD: STA  $5A      ;Derive a new ARYTAB MSB
7BDF: INY                ;Move over to nominal byte 3
7BE0: JSR  $03B7     ;Obtain a

```

---

```

7BE3:  ADC   $5B      ;New ARYTAB
7BE5:  STA   $5B      ;LSB.
7BE7:  PLP
7BE8:  BPL   $7BBA    ;Get the flag for the variable type in the 2nd
                          ;Character of the variable's name and if it is a
7BEA:  TXA                          ;Floating Point Number or Function, then branch
                          ;Move the first character of the variable's name
                          ;Into the accumulator
7BEB:  BMI   $7BBA    ;If it is an Integer, then branch
7BED:  INY
7BEE:  JSR   $03B7
7BF1:  LDY   #$00
7BF3:  ASL
7BF4:  ADC   #$05
7BF6:  ADC   $24
7BF8:  STA   $24
7BFA:  BCC   $7BFE
7BFC:  INC   $25
7BFE:  LDX   $25
7C00:  CPX   $5B
7C02:  BNE   $7C08
7C04:  CMP   $5A
7C06:  BEQ   $7BBE
7C08:  LDY   #$00
7C0A:  JSR   $03B7
7C0D:  BEQ   $7C33
7C0F:  STA   $79
7C11:  INY
7C12:  JSR   $03B7
7C15:  CLC
7C16:  ADC   $79
7C18:  STA   $5C
7C1A:  INY
7C1B:  JSR   $03B7
7C1E:  ADC   #$00
7C20:  STA   $5D
7C22:  LDY   #$00
7C24:  JSR   $42E2
7C27:  ADC   #$07
7C29:  STA   ($5C),Y
7C2B:  INY
7C2C:  JSR   $42E2
7C2F:  ADC   #$00
7C31:  STA   ($5C),Y
7C33:  LDA   #$03
7C35:  CLC
7C36:  ADC   $24
7C38:  STA   $24

```





```

7C59: CLC           ;Clear the carry flag for addition
7C5A: ADC    #2     ;Add two
7C5C: LDY    $62    ;Get the MSB of the descriptor's address
7C5E: BCC    $7C61  ;If no overflow occurred when 2 was added to the
                    ;LSB then branch to prevent adding 1 to the MSB
7C60: INY           ;Add one to the MSB of the descriptor's address
7C61: STA    $49    ;And save the result in VARPNT
7C63: STY    $4A    ;As the new descriptor address
7C65: RTS           ;Exit this routine
    
```

```

*****
*                                                                 *
*                               BLTU                               *
*                                                                 *
*                BLock Transfer Utility                          *
*                                                                 *
* When a descriptor is being created, this routine is          *
* called to create room for the new descriptor in                *
* memory.                                                         *
*                                                                 *
* On entry to this routine, the accumulator and the            *
* Y register will contain the number of bytes that              *
* need to be allocated in memory in LSB/MSB format.            *
* The routine then falls through to BLTU1 below.              *
*                                                                 *
*****
    
```

```

7C66: JSR    $5017  ;Check to ensure there is enough room for the
                    ;Descriptor which is about to be created
7C69: STA    $33    ;Save the address in
7C6B: STY    $34    ;STREND
    
```

```

*****
*                                                                 *
*                               BLTU1                             *
*                                                                 *
*                BLock Transfer Utility 1                        *
*                                                                 *
* This routine is used to insert data into a specified         *
* section of memory for the variable routines in order         *
* to make room for the new variable descriptor. The           *
* parameters for this routine are as follows:                   *
*                                                                 *
* POINTER : DESCRIPTION                                         *
* ----- *
* $61,$62 : Starting address of the BLOCK to be moved         *
    
```

```

* ----- *
* POINTER : DESCRIPTION *
* ----- *
* $5C,$5D : Starting address of the BLOCK + the number *
*           : of bytes to move *
* ----- *
* $5A,$5B : Destination address + the number of bytes *
*           : to move *
* ----- *
*
* The above information can be put to good use. For *
* example, if you wish to move 8 bytes of data starting*
* at $2008 in RAM BANK 1 to $3008, you would place the *
* following values in the pointers mentioned above: *
*
* POINTER : VALUE *
* ----- *
* $61,$62 : 2008 Start of the BLOCK *
* ----- *
* $5C,$5D : 2010 Start of the BLOCK + number of bytes *
*           : $2008 + $08 = $2010 *
* ----- *
* $5A,$5B : 3010 Destination address + number of bytes *
*           : $3008 + $08 = $3010 *
* ----- *
*
* With the above information as a guide, the *
* development of word processing programs or any other *
* program that requires the movement of data or text *
* around in memory will be made easier. *
*
* *****

```

```

7C6D: SEC           ;Set the carry for subtraction
7C6E: LDA   $5C     ;Get the LSB of the end of the
                   ;block to be moved +1
7C70: SBC   $61     ;And subtract the LSB of the start of the
                   ;block to be
7C72: STA   $24     ;Moved-save the result in location $24
7C74: TAY                   ;And also place it into the Y register
7C75: LDA   $5D     ;Get the MSB of the end of the block
                   ;to be moved + 1
7C77: SBC   $62     ;And subtract the MSB of the start of the
                   ;block to be
7C79: TAX                   ;Moved to and save the result in the X register
7C7A: INX                   ;Add one to the MSB
7C7B: TYA                   ;Move the LSB into the accumulator

```

```

7C7C: BEQ   $7CA3      ;If the number of bytes to move is a multiple of
                        ;256, then branch
7C7E: LDA   $5C        ;Get the LSB of the end of the block address
7C80: SEC                       ;Set the carry flag for subtraction
7C81: SBC   $24        ;Subtract the odd number of bytes from it
7C83: STA   $5C        ;And save it as the new LSB
7C85: BCS   $7C8A      ;If there was no overflow, then branch
7C87: DEC   $5D        ;Subtract one from the MSB
7C89: SEC                       ;Set the carry flag for subtraction
7C8A: LDA   $5A        ;Get the MSB of the end of the block address
7C8C: SBC   $24        ;And subtract the odd number of bytes from it
7C8E: STA   $5A        ;And save it as the new MSB
7C90: BCS   $7C9B      ;If an overflow occurred, then branch
7C92: DEC   $5B        ;Decrement the LSB of the end of block by one
7C94: BCC   $7C9B      ;If there was no overflow, then exit
7C96: JSR   $42E2      ;Get a byte from the starting block
7C99: STA   ($5A),Y    ;And save it in the target block
7C9B: DEY                       ;Decrement the index pointer
7C9C: BNE   $7C96      ;Continue looping until done
7C9E: JSR   $42E2      ;Get the last byte (nominal byte 0)
7CA1: STA   ($5A),Y    ;And save it
7CA3: DEC   $5D        ;Decrement the MSB of the starting block
7CA5: DEC   $5B        ;Decrement the MSB of the destination address
7CA7: DEX                       ;Decrement the page counter by one
7CA8: BNE   $7C9B      ;If all the pages are not done then branch
7CAA: RTS                       ;Exit this routine

```

```

*****
*
*                               ISARY                               *
*
*                               ISolate ARraY                       *
*
* This routine will search through the ARRAY storage area searching *
* for the array whose name is held in locations $47, $48 (VARNAM) *
* and with TXTPTR pointing to the opening parenthesis. If the *
* array is found, then this routine will exit to RAERR below. *
* If the array is not found, then this routine will exit to *
* ALARY below to create the array.
*
* NOTE: When these routines exit, you are left in BANK 14 with *
* RAM BANK 1 enabled.
*
*****

```

---

```

7CAB: LDA   $0E           ;Combine the DIMFLG ($0E) and INTFLG ($10)
7CAD: ORA   $10           ;Onto the stack to preserve their
7CAF: PHA                   ;Flags
7CB0: LDA   $0F           ;Save VALTYP
7CB2: PHA                   ;Onto the stack
7CB3: LDY   #$00          ;Set the number of dimensions to Zero
7CB5: TYA                   ;And save
7CB6: PHA                   ;It onto the stack
7CB7: LDA   $48           ;Place the second character of the
7CB9: PHA                   ;Variable name onto the stack
7CBA: LDA   $47           ;Place the first character of the
7CBC: PHA                   ;Variable name onto the stack
7CBD: JSR   $84A7        ;Get the next character, convert it to Integer
                          ;Format in locations $66, $67 (LSB, MSB format)
7CC0: PLA                   ;Get the first character of the variable
7CC1: STA   $47           ;Name off of the stack and save it in $47
7CC3: PLA                   ;Get the second character of the variable name
7CC4: STA   $48           ;And place it into $48
7CC6: PLA                   ;Get the number to dimension
7CC7: TAY                   ;And save it in the Y register
7CC8: TSX                   ;Place the stack pointer into the X register
7CC9: LDA   $0102,X       ;Make room for the
7CCC: PHA                   ;Number of elements
7CCD: LDA   $0101,X       ;By moving the bottom flags which were
                          ;VALTYP and DIMFLG/INTFLG, which were combined,
7CD0: PHA                   ;From the bottom to the top of the stack
7CD1: LDA   $66           ;Save the
7CD3: STA   $0102,X       ;Number of elements
7CD6: LDA   $67           ;Save the number of elements
7CD8: STA   $0101,X       ;To dimension in their place
7CDB: INY                   ;Increment the number of dimensions by one
7CDC: STY   $0D           ;Save the number of dimensions
7CDE: JSR   $0386        ;Get the character after the dimension
7CE1: LDY   $0D           ;Get the number of dimensions
7CE3: CMP   #','         ;Is there a comma (multidimensional ARRAY)?
7CE5: BEQ   $7CB5        ;If so, then branch to process the next value
7CE7: JSR   $7956        ;Check for the closing parenthesis and if it is
                          ;Not found, then generate a 'SYNTAX' error
7CEA: PLA                   ;Restore the flag for string or
7CEB: STA   $0F           ;Numeric in VALTYP
7CED: PLA                   ;Restore the flag for integer or
7CEE: STA   $10          ;Floating point into INTFLG
7CF0: AND   #$7F         ;Drop bit 7 which could have held
7CF2: STA   $0E           ;The flag for string and save the first
                          ;Character of the name in DIMFLG

```

```

*****
*
*                               SRCHARY
*
*                               SeaRCH for ARRAy
*
* This routine will search through the ARRAY storage
* area searching for the array whose name is held in
* locations $47, $48 (VARNAM). If the array is
* found, then this routine will exit to RAERR to
* ensure that this is indeed the correct array.
*
*****

```

```

7CF4: LDX  $31      ;Get the Start of Array Storage address from
7CF6: LDA  $32      ;ARYTAB
7CF8: STX  $61      ;And save it in
7CFA: STA  $62      ;Locations $61, $62
7CFC: CMP  $34      ;Check to see if the address in the X register
7CFE: BNE  $7D04    ;and the accumulator is equal to STREND, which
7D00: CPX  $33      ;Is the end of the ARRAY storage, and
7D02: BEQ  $7D46    ;If it is, then branch to create the ARRAY since
                          ;Since it does not already exist
7D04: LDY  #$00     ;Zero the index pointer
7D06: JSR  $4300    ;Get the first character of the
7D09: INY                      ;Variable descriptor that is being
7D0A: CMP  $47      ;Searched through and if they
7D0C: BNE  $7D15    ;Do not match, then branch
7D0E: JSR  $4300    ;Get the second character from the current
7D11: CMP  $48      ;Descriptor and if it equals the character
7D13: BEQ  $7D2D    ;That we are searching for, then the ARRAY has
                          ;Already been dimensioned, therefore branch
7D15: INY                      ;Move over to the length
7D16: JSR  $4300    ;Get the LSB of the next
7D19: CLC                      ;Descriptor
7D1A: ADC  $61      ;And add it to ARYTAB
7D1C: TAX                      ;Save the result in the X register
7D1D: INY                      ;Move over to the MSB of the next
7D1E: JSR  $4300    ;Available descriptor and add that
7D21: ADC  $62      ;Value to the MSB of ARYTAB
7D23: BCC  $7CF8    ;If there was no overflow, then branch, else

```

```

*****
*
*                               BSERR
*
*                               Bad Subscript ERROR
*
*   Generate an 'BAD SUBSCRIPT' error message.
*
*****

```

```

7D25: LDX  #18      ;Generate a 'BAD SUBSCRIPT' error
7D27: .BYTE $2C    ;Mask to fall through to $7D2A

```

```

*****
*
*                               IQERR
*
*                               Illegal Quantity ERROR
*
*   Generate an 'ILLEGAL QUANTITY ' error message.
*
*****

```

```

7D28: LDX  #14      ;Generate an 'ILLEGAL QUANTITY' error
7D2A: JMP  $4D3C    ;JUMP to the error message routine

```

```

*****
*
*                               RAERR
*
*                               Redimensioned Array ERROR
*
*   This routine will first check the DIMFLG to see if
*   this array, which has already been dimensioned, is
*   to be dimensioned. If it is to be dimensioned, a
*   'REDIM'D ARRAY' error will occur. If the array is
*   not to be dimensioned, then this routine will ensure
*   that the number of dimensions in the ARRAY in memory
*   and the one to be found are the same. If they are
*   not the same, then a 'BAD SUBSCRIPT' error will be
*   generated. If they are the same, then this routine
*   will exit to ARYELM to process the array.
*
*****

```

```

7D2D: LDX  #19      ;Generate a 'REDIM'D ARRAY' error message
7D2F: LDA  $0E      ;Is the ARRAY to be dimensioned?

```

```

7D31: BNE $7D2A ;If so, then generate a 'REDIM'D ARRAY' error
7D33: JSR $7E71 ;Get the address of the array's descriptor
7D36: LDY #$04 ;Get the number of
7D38: JSR $4300 ;Dimensions from the descriptor
7D3B: STA $79 ;Save it into SYNTMP
7D3D: LDA $0D ;Get the number of dimensions for this ARRAY
;To be modified
7D3F: CMP $79 ;If it is not the same,
7D41: BNE $7D25 ;Then generate a 'BAD SUBSCRIPT' error
7D43: JMP $7DD2 ;Process the array

```

```

*****
*
*
*           ALARY
*
*           Allocate ARRAy
*
* This routine is used to create a new array
* descriptor. Upon entry to this routine, locations
* $47, $48 (VARNAM) will contain the variable's name
* for which this array is to be created. Combined
* with the two characters of the variable's name are
* the flags that indicate what type of array is to be
* created (String, Integer, or Floating Point).
* Locations $61, $62 will contain the address of
* byte 0 of the array descriptor for the array. If
* the DIMFLG is not set, then this routine will create
* the array with the default dimension value of 11.
*
* NOTE: For information on the different flags that
* are combined with the characters of the
* variable name, see the section on variable
* storage in this book.
*
*****

```

```

7D46: JSR $7E71 ;Get the address of the first ARRAY element
7D49: JSR $5017 ;Ensure that there is enough room in string
;For this ARRAY
7D4C: LDY #$00 ;Zero the MSB of the
7D4E: STY $73 ;Length of the variable's descriptor
7D50: LDX #5 ;Set the index to five
7D52: LDA $47 ;Get the first character of the variable name
7D54: STA $FF04 ;Enable BASIC BANK 14 with RAM BANK 1 enabled
7D57: STA ($61),Y ;Place the first character of the variable name
;In byte 0 of the ARRAY descriptor
7D59: BPL $7D5C ;If the variable type is INTEGER, then branch

```

```

7D5B: DEX                ;Subtract 1 from the length of the ARRAY
                          ;descriptor
7D5C: INY                ;Move over to Byte 1 of the ARRAY's descriptor
7D5D: LDA $48            ;Get the second character of the variable name
7D5F: STA ($61),Y       ;Place it into Byte 1 of the ARRAY's descriptor
7D61: BPL $7D65         ;If the variable type is a string, then branch
7D63: DEX                ;Subtract two from the length of the ARRAY
                          ;element, hereby making the length of the element
7D64: DEX                ;3 for Strings, 2 for Integers
                          ;And 5 for Floating Point
7D65: STX $72           ;Save the length of this ARRAY's element
7D67: LDA $0D           ;Get the number of dimensions of the ARRAY - 1
7D69: INY                ;Move the index pointer over to
7D6A: INY                ;Nominal Byte 4 of the
7D6B: INY                ;ARRAY descriptor
7D6C: STA ($61),Y       ;Save the number of dimensions into Byte four of
                          ;The ARRAY's descriptor
7D6E: LDX #11           ;System default value for dimensioning LSB of
7D70: LDA #0            ;Limit in case the branch below holds true
7D72: BIT $0E           ;See if the 1st character of the variable
7D74: BVC $7D7E        ;Name is in DIMFLG and if not, then branch to
                          ;Bypass obtaining the number of elements off the
                          ;Stack and use the system default value of 11
7D76: PLA                ;Get the LSB of the number of dimensions
7D77: CLC                ;Off of the stack
7D78: ADC #1            ;Add one to it
7D7A: TAX                ;And move it into the X register
7D7B: PLA                ;Get the MSB of the number of elements off
7D7C: ADC #0            ;The stack and add the carry to it
7D7E: INY                ;Move the index over to nominal Byte 5 of
                          ;The variable descriptor
7D7F: STA ($61),Y       ;Save the number of ARRAY descriptor elements
7D81: INY                ;Move the index over to nominal Byte 6 of
                          ;The variable descriptor
7D82: TXA                ;Move it into the accumulator
7D83: STA ($61),Y       ;Save the number of ARRAY descriptor elements
7D85: JSR $7E3E         ;Get the size of the multidimensional ARRAY
7D88: STX $72           ;Save the size in HULP in
7D8A: STA $73           ;LSB, MSB format
7D8C: LDY $24           ;Get the index into the ARRAY descriptor
7D8E: DEC $0D           ;Subtract one from the number of dimensions
7D90: BNE $7D6E        ;If there are more to process, then branch to
                          ;Obtain the total number of elements
7D92: ADC $5B           ;Add the index to the MSB of the ARRAY element
                          ;address
7D94: BCS $7DFD        ;If it is out-of-range, then branch to generate
                          ;An 'OUT OF MEMORY' error

```



```

7D96: STA    $5B      ;Save the End of the ARRAY descriptor
7D98: TAY                      ;Save it into the Y register
7D99: TXA                      ;Move the LSB into the Accumulator
7D9A: ADC    $5A      ;Add it to the ARRAY element address
7D9C: BCC    $7DA1    ;If there was no overflow, then branch
7D9E: INY                      ;Add one to the MSB and if it is above $FFxx,
7D9F: BEQ    $7DFD    ;Generate an 'OUT OF MEMORY' error message
7DA1: JSR    $5017    ;Check to ensure enough memory
                          ;(Y = MSB, A = LSB of number of bytes needed)
7DA4: STA    $33      ;Save the address
7DA6: STY    $34      ;Of the last array
7DAB: LDA    #$00      ;Fill the area with zeros
7DAA: INC    $73      ;Add one to the page pointer (number of pages to
                          ;fill)
7DAC: LDY    $72      ;Get the offset value in the Y register
7DAE: BEQ    $7DB5    ;If 0, skip this page and do the next one
7DB0: DEY                      ;Subtract one from the index register
7DB1: STA    ($5A),Y    ;Fill the page pointed to by $5A/$5B with zero's
7DB3: BNE    $7DB0    ;Continue until this page is done
7DB5: DEC    $5B      ;Continue filling the new array storage area
                          ;with
7DB7: DEC    $73      ;Zeros until the page pointer reaches zero
7DB9: BNE    $7DB0    ;Not done, then branch
7DBB: INC    $5B      ;Decrement the MSB by one
7DBD: SEC                      ;Set the carry for subtraction
7DBE: LDA    $33      ;Get the end of ARRAY storage and subtract the
7DC0: SBC    $61      ;Start of the array from it
7DC2: LDY    #$02      ;Index over to the third nominal byte in the
                          ;descriptor
7DC4: STA    ($61),Y    ;And save the address of the next ARRAY
                          ;Descriptor in nominal byte 3 in the descriptor
7DC6: LDA    $34      ;Which when added to ARraY TABLE
7DC8: INY                      ;It will point to the
7DC9: SBC    $62      ;Next available descriptor for
7DCB: STA    ($61),Y    ;STREND
7DCD: LDA    $0E      ;Was this routine called by the
7DCF: BNE    $7E3D    ;BASIC DIM command? If so, then branch to exit
    
```

```

*****
*
*
*           ARYELM
*
*           ARraY ELeMent
*
*
* This routine will check the ARRAY descriptor
* that is pointed to by locations $61, $62 with the
* array that we are searching for by comparing the
    
```

```

* number of DIMENSIONS of the array we are searching *
* for to the ARRAY we are currently pointing to. *
* If the ARRAY element we are searching for is *
* greater, then a 'BAD SUBSCRIPT' error will result. *
* However, if the ARRAY element we are searching for *
* is less than or equal to, then ELMADR will be *
* executed to calculate the actual element's address *
* in RAM BANK 1. *
* *
*****

```

```

7DD1: INY                ;Add one to the index register
7DD2: JSR $4300          ;Get the number of dimensions
7DD5: STA $0D           ;For the array from the descriptor and save it
7DD7: LDA #$00          ;Clear the
7DD9: STA $72           ;Offset to
7ddb: STA $73           ;The Array element
7DDD: INY                ;Add one to the index register
7DDE: PLA               ;Get the index value from the stack
7DDF: TAX               ;And save it into the X register
7DE0: STA $66           ;And in the FAC
7DE2: JSR $4300          ;Get the MSB of the Index value
7DE5: STA $79           ;From the Array Descriptor
7DE7: PLA               ;Get the MSB of the Index value off of the stack
7DE8: STA $67           ;Get the MSB of the Index value
7DEA: CMP $79           ;That was specified in the ARRAY statement
7DEC: BCC $7E00         ;If it is less than, then get the address of the
                          ;ARRAY element
7DEE: BNE $7DFA         ;If it is greater than, then
                          ;Generate a 'BAD SUBSCRIPT' error
7DF0: INY                ;If it is equal to, then
7DF1: JSR $4300          ;Get the number of dimensions from the ARRAY
7DF4: STA $79           ;Descriptor and save it. Then compare it to the
7DF6: CPX $79           ;Element in the ARRAY statement
7DF8: BCC $7E01         ;If it is less than, then get the address of the
                          ;ARRAY Element, else
7DFA: JMP $7D25         ;Generate a 'BAD SUBSCRIPT' error
7DFD: JMP $4D3A         ;Generate an 'OUT OF MEMORY' error

```

```

*****
* *
* ELMADR *
* *
* Evaluate eLeMent AdDress *
* *
* This routine is used to actually obtain the address *
* of the ARRAY ELEMENT and then places that address *

```

```

* in locations $49, $4A (VARPNT). The routine then *
* exits with the address in the Accumulator and the *
* Y register in LSB, MSB format. *
* *
*****

```

```

7E00: INY                ;Move over to the next byte and
7E01: LDA    $73        ;Check to see if the offset
7E03: ORA    $72        ;To the ARRAY element
7E05: CLC                ;Is zero and if it is,
7E06: BEQ    $7E12      ;Then branch to skip multiplication
7E08: JSR    $7E3E      ;Get the size of the ARRAY
7E0B: TXA                ;Move the LSB of the size of the ARRAY and add
7E0C: ADC    $66        ;It to the index
7E0E: TAX                ;Move the LSB into the X register and
7E0F: TYA                ;The MSB into the accumulator
7E10: LDY    $24        ;Restore the index to the
7E12: ADC    $67        ;ARRAY descriptor
7E14: STX    $72        ;And save as the new index value
7E16: DEC    $0D        ;Decrement the number of dimensions
7E18: BNE    $7DDB      ;If it is not equal to zero, then branch
7E1A: STA    $73        ;Save the MSB of the offset
7E1C: LDX    #5         ;Set the default length of the element to 5 for
                          ;Floating Point Numbers
7E1E: LDA    $47        ;Get the first character of the variable name
7E20: BPL    $7E23      ;If the variable type is not Integer, branch
7E22: DEX                ;Subtract one from the length of the ARRAY
                          ;Element (the result is 4)
7E23: LDA    $48        ;Get the second character of the variable name
7E25: BPL    $7E29      ;If the variable type is a String, then branch
7E27: DEX                ;Subtract 2 from the length of the ARRAY element
                          ;Which would make the length of the elements for
                          ;3 for Strings, 2 for Integers, and 5 for
                          ;Floating Point Numbers
7E28: DEX                ;Elements 2
7E29: STX    $2A        ;Save the length of the ARRAY element
7E2B: LDA    #$00       ;Make the MSB of the length equal to zero
7E2D: JSR    $7E49      ;Get the address of the ARRAY element
7E30: TXA                ;Add the address of the ARRAY's
7E31: ADC    $5A        ;Element to the
7E33: STA    $49        ;Address of the
7E35: TYA                ;ARRAY's descriptor
7E36: ADC    $5B        ;And then
7E38: STA    $4A        ;Save it as the address of the element we were
                          ;Looking for
7E3A: TAY                ;Place the MSB of the array element in the Y
                          ;register

```

```

7E3B: LDA    $49      ;Place the LSB of the array element in the
                        ;accumulator,
7E3D: RTS          ;Then exit this routine with the address left in
                        ;Locations $49, $4A (VARPNT)

```

```

*****
*
*                               UMULT                               *
*
*                               Utility MULTplication                *
*
* This routine computes the size of a multidimensional *
* ARRAY and returns the result in the X and Y *
* registers in LSB, MSB format. *
*
*****

```

```

7E3E: STY    $24      ;Save the index to the array descriptor
7E40: JSR    $4300    ;Get the MSB of the limit
7E43: STA    $2A      ;And save it
7E45: DEY
7E46: JSR    $4300    ;Get the LSB of the array descriptor
7E49: STA    $2B      ;And save it
7E4B: LDA    #16      ;Set the number of bits
7E4D: STA    $5F      ;To multiply (16)
7E4F: LDX    #$00
7E51: LDY    #$00
7E53: TXA
7E54: ASL
7E55: TAX
7E56: TYA
7E57: ROL
7E58: TAY
7E59: BCS    $7DFD    ;If there was an overflow, then generate
                        ;An 'OUT OF MEMORY' error
7E5B: ASL    $72      ;Shift the high bit
7E5D: ROL    $73      ;Out of the length of the ARRAY
7E5F: BCC    $7E6C    ;If there was no overflow, then branch to
                        ;Process the next multiplication
7E61: CLC
7E62: TXA
7E63: ADC    $2A      ;Of the limit to 'X'
7E65: TAX
7E66: TYA
7E67: ADC    $2B      ;Add the MSB of the limit to 'Y'
7E69: TAY
7E6A: BCS    $7DFD    ;If there was an overflow, then generate

```

```

;An 'OUT OF MEMORY' error
7E6C: DEC   $5F      ;Continue until all
7E6E: BNE   $7E53    ;16 bits are done
7E70: RTS                      ;Exit the routine

```

```

*****
*
*                               *
*           FSTELM              *
*
*           find the FirST ELeMent
*
* This routine will take the value that is stored in
* location $0D, which contains the number of
* dimensions minus one that are contained in the
* ARRAY, multiply that value by two, and add five,
* which is the number of actual bytes from the first
* byte of the array descriptor to the number of
* dimensions in the array. The routine then adds the
* result of the above calculations to the address of
* the ARRAY descriptor and saves the result to
* locations $5A, $5B. This routine will exit with
* the Accumulator and the Y register containing the
* address of the FIRST ARRAY element.
*
* For example, if the following BASIC statement were
* used, the following calculations would result:
*
*           10 CLR:DIM A$(10,10)
*
* Since there are two dimensions in the ARRAY, a
* value of two would be found in location $0D.
* That value is then multiplied by 2 thereby resulting
* in a product of 4. Next, 5 is added resulting in a
* sum of 9. The sum of this addition is then added to
* the address of the ARRAY descriptor. The result
* is the address that would point to the first array
* element and is saved to locations $5A and $5B.
*
*****

```

```

7E71: LDA   $0D      ;Get the number of dimensions
7E73: ASL                      ;Multiply it by two
7E74: ADC   #$05     ;Add five to it
7E76: ADC   $61      ;And add the address of the array
7E78: LDY   $62
7E7A: BCC   $7E7D    ;If there was no overflow, then branch
7E7C: INC                      ;Add one to the MSB

```

```

7E7D: STA $5A ;Save the point to the array's
7E7F: STY $5B ;First element
7E81: RTS
7E82: .BYTE $FF ;From this point until $7FFF, these locations
7E7D: STA $5A ;Save the pointer to the array's
7E7F: STY $5B ;First element
7E81: RTS ;Exit the routine
7E82: .BYTE $FF ;From this point until $7FFF, these locations
7FFD: .BYTE $FF ;Are filled with a value of $FF

```

```

*****
*
*
*          BASICVER
*
* This location contains the BASIC ROM version number
* and currently contains a one at the time of the
* printing of this book. The extent of the changes
* between version one and zero are unknown at this time.*
*
*****

```

```

7FFE: .BYTE $01 ;Version number
7FFF: .BYTE $FF ;Fill value

```

```

*****
*
*          BASIC FRE FUNCTION
*
* Function syntax: FRE(X)
*
* NOTE: If X equals zero, then the value returned
* represents the number of bytes available for
* BASIC programs.
*
* If X equals one, then the value returned
* represents the number of bytes available for
* BASIC variable storage.
*
*****

```

```

8000: JSR $87F7 ;Convert ASCII to HEX and return the value
;In the X register
8003: CPX #1 ;For RAM BANK 1?
8005: BCC $800C ;Branch if for RAM BANK 0
8007: BEQ $803A ;Branch if for RAM BANK 1
8009: JMP $7D28 ;Generate an 'ILLEGAL QUANTITY' error

```

```
*****
*
*          BASIC FRE(0) FUNCTION          *
*
* This routine subtracts the end of the BASIC program *
* from the highest address available to BASIC in RAM *
* BANK 0.                                           *
*
*****
```

```
800C: SEC          ;Set the carry for subtraction
800D: LDA $1212    ;Subtract the TEXT TOP pointers from the
8010: SBC $1210    ;MAX MEM 0. In other words, subtract
8013: TAY          ;The end of your program's address from the
8014: LDA $1213    ;MAXimum MEMory available, which is normally
8017: SBC $1211    ;Set to $FF00
801A: BCS $8047    ;Always branch, unless MAX MEM 0 is set
                    ;To an address that is less than the address
                    ;Of the BASIC program. If this condition
                    ;Does exist, then you will see the TRASH CBM
                    ;Generates below.
```

```
*****
*
* This is useless code that CBM left in the ROMs and *
* will only be executed if MAX MEM 0 is set to an *
* address that is less than the address of the BASIC *
* program.                                           *
*
*****
```

```
801C: LDX $35      ;Useless code, will not be executed
801E: INX          ;Unless the conditions as explained
801F: INY          ;Above are met
```

```
*****
*
* This routine will print the message left by CBM. *
* Enter this routine with the following values: *
* A = 123, X = 45, and Y = 6.                   *
*
*****
```

```
8020: STA $70      ;Save the key value
8022: TYA          ;Transfer the second key to the accumulator
8023: SEC          ;And subtract 5
8024: SBC #$05     ;From it
```

```

8026: STA  $71      ;And save it
8028: LDA  $71      ;Get the second key
802A: EOR  $AE37,X  ;Generate first seed by 'EOR'ing with memory
802D: EOR  $70      ;Generate the ASCII value by 'EOR'ing
                        ;With the first key
802F: BEQ  $8075    ;If the value is equal to zero, then finished
8031: JSR  $FFD2    ;Print the character
8034: INC  $71      ;Increment the second key by one
8036: INX                    ;Continue until a value of zero is
8037: BNE  $8028    ;Found or until the X register overflows
8039: RTS                    ;Exit

```

```

*****
*
*                               *
*          BASIC FRE(1) FUNCTION *
*                               *
* This routine subtracts the address that represents *
* the end of the BASIC variables that are stored,   *
* from the highest address available for BASIC      *
* variable storage to get the amount of free memory *
* that is in RAM BANK 1.                            *
*
*****

```

```

803A: JSR  $92EA    ;Perform a GARBAGE COLLECTION to free up any
                        ;Unused strings etc..
803D: SEC                    ;Set the carry flag for subtraction
803E: LDA  $35      ;Subtract the end of the BASIC ARRAYS from
8040: SBC  $33      ;The bottom of the string storage area.
8042: TAY
8043: LDA  $36
8045: SBC  $34
8047: JMP  $84C9    ;Convert the value into a floating point number

```

```

*****
*
*                               *
*          BASIC VAL FUNCTION  *
*                               *
* Function syntax:  VAL (X$)   *
*
* NOTE: X$ = a string consisting of numbers, *
*         characters, or both  *
*
* This routine will take the string that is in *
* parentheses and place a delimiter of zero after the *
* string. It will then convert the string into a *
* Floating Point number in FAC1 and place the address *

```



```

* of that number in locations $24, $25.  If the string *
* does not contain any numeric characters then a value *
* of zero is returned by the VAL function.           *
*                                                     *
*****

```

```

804A: JSR   $866E      ;Get the length of string in the accumulator
804D: BNE   $8052      ;If the length is not zero, then branch
804F: JMP   $88D6      ;If the length of string is zero, then VAL=0
8052: CLC                          ;Add the length of the string
8053: ADC   $24        ;To the address of string that is in $24, $25
8055: STA   $72        ;And save it in locations $72, $73
8057: LDA   $25        ;Which will have $72, $73 pointing
8059: ADC   #0         ;One character past the string
805B: STA   $73        ;Which is the LSB of the address of the
                        ;String's descriptor
805D: LDY   #0         ;Index to the LSB of the string descriptor
805F: LDA   #$72       ;Get the pointer to the address
                        ;Of the string's descriptor
8061: JSR   $03AB      ;Get the LSB of the string descriptor and
8064: PHA                          ;Save it onto the stack
8065: TYA                          ;Transfer the delimiter of zero to the
8066: STA   ($72),Y     ;Accumulator and save it temporarily as the 1
                        ;Last byte of the string
8068: JSR   $8E03      ;Skip any spaces before first letter of digit
                        ;(this routine uses the actual address of the
                        ;String that is stored in locations $24, $25)
806B: LDX   #1         ;Set the RAM Bank of where the string is stored
806D: JSR   $8D22      ;Calculate the VALue of the string and convert
                        ;That value to Floating Point format
8070: PLA                          ;Get back the LSB of the address of
                        ;The string's descriptor
8071: LDY   #0         ;Index to the last byte of the string
8073: STA   ($72),Y     ;Save the original value back into
                        ;The string descriptor
8075: RTS                          ;And exit

```

```

*****
*                                                     *
*                   BASIC DEC FUNCTION                 *
*                                                     *
* Function syntax:  DEC(HEX)                          *
*                                                     *
* NOTE:  HEX = up to a four-digit hexadecimal string *
*                                                     *
* This numeric function converts a hexadecimal number *
* into the equivalent decimal number.  Decimal numbers *

```

```

* are mostly used in BASIC programming, whereas      *
* hexadecimal numbers are used in machine language   *
* programming.                                       *
*                                                     *
*****

```

```

8076: JSR   $866E      ;Get the string parameters
8079: STA   $26        ;Save the length of the string
807B: LDY   #0         ;Set the index to the first byte of string
807D: STY   $27        ;Clear the digit counter
807F: STY   $72        ;And clear two areas to hold the
8081: STY   $73        ;Final converted number
8083: CPY   $26        ;Compare the length of the string
8085: BEQ   $80BB      ;And exit if it is equal to zero
8087: JSR   $03B7      ;Get a byte of the string
808A: INY                   ;Increment the index to the next byte
808B: CMP   #$20        ;Is the character of the string a space?
808D: BEQ   $8083      ;If it is, then branch to get the next char
808F: INC   $27        ;Increment the digit counter by one
8091: LDX   $27        ;If there are more than 4 digits
8093: CPX   #5         ;In the string
8095: BEQ   $80C2      ;Then generate an 'ILLEGAL QUANTITY' error
8097: CMP   #'0'       ;Is the character a zero?
8099: BCC   $80C2      ;If the character is less than zero,
                        ;Then 'ILLEGAL QUANTITY' error
809B: CMP   #':'       ;If the character is a digit
809D: BCC   $80A9      ;Between zero and nine, then skip the letter
                        ;Compare routine
809F: CMP   #'a'       ;If the character is less
80A1: BCC   $80C2      ;Than an 'A' or greater
80A3: CMP   #'g'       ;Than a 'G'
80A5: BCS   $80C2      ;Generate an 'ILLEGAL QUANTITY' error
80A7: SBC   #$07       ;If the character is a letter
80A9: SBC   #$2F       ;Then subtract 55
                        ;If the character is a digit
                        ;Then subtract 48
80AB: ASL                   ;Multiply the result
80AC: ASL                   ;By 16
80AD: ASL
80AE: ASL
80AF: LDX   #$04       ;Set the shift counter for 4
80B1: ASL                   ;Multiply the value by 2
80B2: ROL   $72        ;And shift any resulting carry
80B4: ROL   $73        ;Into locations $72, $73
80B6: DEX                   ;Continue until four complete shifts
80B7: BNE   $80B1      ;Have been done
80B9: BEQ   $8083      ;Continue until all characters are done

```

```

80BB: LDY   $72       ;Get the LSB
80BD: LDA   $73       ;And the MSB of the final value
80BF: JMP   $84C9     ;Convert the value to Floating Point format
                        ;And exit
80C2: JMP   $7D28     ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
*           BASIC PEEK FUNCTION
*
* Function syntax: PEEK(X)
*
* NOTE: X = a memory location value between
*         0 - 65535
*
* This function returns a value between 0 and 255
* that represents the value stored in the address that
* is being PEEKed. The bank from which the value is
* PEEKed is specified by the BANK command.
*
*****

```

```

80C5: LDA   $17       ;Save $16, $17
80C7: PHA
80C8: LDA   $16       ;For recall after
80CA: PHA
80CB: JSR   $77DA     ;Ensure the value type is numeric
                        ;If not, then generate a 'TYPE MISMATCH' error
80CE: JSR   $8815     ;Convert the value PEEK address to integer
                        ;format and store it into locations $16, $17
80D1: LDX   $03D5     ;Get the BANK # for the peek
80D4: LDY   #0        ;Zero the index pointer
80D6: LDA   #$16      ;Get the pointer to the PEEK address
80D8: JSR   $FF74     ;Get the PEEK value from memory
80DB: TAY
80DC: PLA
80DD: STA   $16       ;To their
80DF: PLA
80E0: STA   $17       ;Values
80E2: JMP   $84D4     ;Change the PEEK value that is stored in the
                        ;Y register to Floating Point format

```

```

*****
*
*          BASIC POKE COMMAND
*
* Command syntax: POKE X, Y
*
* NOTE: X = the memory address to be POKEd
*       Y = the value to be POKEd to the memory
*       address specified by X
*
* This function will place the value Y into the
* address specified by X. The bank to which the
* value specified by Y is to be POKEd is set by the
* BANK command.
*
*****

80E5: JSR   $8803      ;Get the address to POKE to from locations $16,
                    ;$17 and put the POKE value into the X register
80E8: TXA
50E9: LDY   #0        ;Zero the index pointer
80EB: LDX   #$16      ;Get the pointer to the POKE address
80ED: STX   $12B9     ;Save it
80F0: LDX   $03D5     ;Get the BANK number to POKE to
80F3: JMP   $FF77     ;And store value in the memory address specified

*****
*
*          BASIC ERR$ FUNCTION
*
* Function syntax: ERR$(X)
*
* NOTE: If X = 'ER', then the ERR$ function will
*       print the error that occurred most recently.
*
*       If X = a number from 1 to 41, the ERR$
*       function will return the system error
*       message that is represented in the operating
*       system by that number.
*
*****

80F6: JSR   $87F7     ;Get the error number that is in parentheses
                    ;Into the X register
80F9: DEX
                    ;Subtract one from the error number
                    ;To obtain an index value
80FA: TXA
                    ;Save the value to the accumulator

```

```

80FB:  CMP    #41      ;If the error number is larger than 40
80FD:  BCS    $8136    ;Generate an 'ILLEGAL QUANTITY' error
80FF:  JSR    $4A82    ;Find the correct error message and store
                        ;The address of that error message in $26, $27
8102:  LDY    #$FF     ;Set index value to $FF so the INC instruction
                        ;At $8107 will set it to zero
8104:  LDX    #0       ;Set the length of the ERROR message to zero
8106:  INX                    ;Increment the length counter by one
8107:  INY                    ;And increment index to next byte of error text
8108:  LDA    ($26),Y  ;Get a character from the error message
810A:  BMI    $8112    ;Branch if it's the end of the text
810C:  CMP    #$20     ;If the byte is less than a space, skip it and
810E:  BCC    $8107    ;Get the next character
8110:  BCS    $8106    ;If the character is greater than or equal to a
                        ;Space, then increment the length counter
                        ;And the index value
8112:  TXA                    ;Save the length of message to the accumulator
8113:  JSR    $8690    ;Allocate room for the string in RAM BANK 1
                        ;Using accumulator (length) as a length counter
8116:  LDX    #0       ;
8118:  LDY    #$FF     ;Set index value to $FF so the INC instruction
                        ;At $8107 will set it to zero
811A:  STA    $FF04    ;Enable RAM BANK 15 with RAM BANK 1
811D:  INY                    ;Move to the next byte of string
811E:  LDA    ($26),Y  ;Get a byte of the string
8120:  CMP    #$20     ;Is it a space?
8122:  BCC    $811D    ;Skip the byte if it is less than a space
8124:  JSR    $8139    ;Swap the X and Y registers
8127:  PHA                    ;Save the byte of the string onto the stack
8128:  AND    #%01111111 ;Drop bit 7 to make the character unshifted
812A:  STA    ($64),Y  ;And save the character into string memory
812C:  JSR    $8139    ;Restore the X and Y registers
812F:  INX                    ;Increment the length counter by one
8130:  PLA                    ;Get back the original character from the stack
8131:  BPL    $811D    ;Continue until a shifted character is found
8133:  JMP    $85D1    ;Pull the address of the calling routine off of
                        ;The stack and delete the string descriptor
8136:  JMP    $7D28    ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
*                               SWAPXY
*
*   This routine is used to swap the contents of the
*   X and Y registers.
*
*****

```

```

8139: PHA
813A: TXA
813B: PHA
813C: TYA
813D: TAX
813E: PLA
813F: TAY
8140: PLA
8141: RTS

```

```

*****
*
*                               BASIC HEX$ FUNCTION
*
* Function syntax:  HEX$(X)
*
* NOTE:  X = a decimal value between 0 - 65535
*
* This function will return a four-digit hexadecimal
* value for the decimal value specified by X.  The
* decimal value must be greater than or equal to zero
* and less than or equal to 65535 or an 'ILLEGAL
* QUANTITY' error will result.
*
*****

```

```

8142: JSR   $77DA      ;Insures the data type is numeric and if not
                       ;Then a 'TYPE MISMATCH' error is generated
8145: LDA   $16        ;Save the values stored
8147: PHA           ;In locations $16, $17
8148: LDA   $17        ;Onto the stack
814A: PHA           ;For later use
814B: JSR   $8815      ;Convert the value in parentheses to integer
                       ;Format and place it in locations $16, $17.
814E: LDA   #$04       ;Create a string four characters in length
8150: JSR   $8690      ;Which will be the result of the HEX$ function
8153: LDY   #0         ;Zero the index register
8155: LDA   $17        ;Get MSB of value to be converted to a string
8157: STA   $FF04      ;Enable BANK 15 but with RAM BANK1 enabled
815A: JSR   $816B      ;Convert MSB of the value into two HEX digits
815D: LDA   $16        ;Convert the LSB of the value into
815F: JSR   $816B      ;Two HEX digits
8162: PLA           ;Restore the
8163: STA   $17        ;Two values
8165: PLA           ;That were previously
8166: STA   $16        ;In $16, $17

```

```
8168: JMP   $85D1      ;Save the string descriptor onto the string
                        ;Descriptor stack
```

```
*****
*
*                               HEXASC                               *
*
*       HEX to ASCII conversion routine                             *
*
* This routine is used to convert the value that is                *
* in the the accumulator to a two digit ASCII value                *
* which is then stored into the the string address                 *
* which is pointed to by locations $64, $65 plus the               *
* offset in the Y register.                                         *
*
*
*****
```

```
816B: PHA                ;Save the value to be converted onto the stack
816C: LSR                ;Divide by 16 to drop the ones digit
816D: LSR
816E: LSR
816F: LSR
8170: JSR   $8174        ;Convert the 16's digit into ASCII format and
                        ;Store the result into the address specified
8173: PLA                ;Get the original value off the stack
8174: AND   #$0F         ;Drop the 16's digit
8176: CMP   #10         ;Compare the value to ten
8178: BCC   $817C        ;And if less then ten, then branch to convert to
                        ;ASCII and save it into the address specified
817A: ADC   #$06         ;If the value was ten or greater, then add six
                        ;To make it into a sixteens digit
817C: ADC   #$30         ;Convert the value to an ASCII value
817E: STA   ($64),Y     ;And store it into the address specified
8180: INY
8181: RTS                ;Exit the routine
```

```
*****
*
*                               BASIC RGR FUNCTION                       *
*
* Function syntax:  RGR(0)                                           *
*
* This function will return the value of the current                 *
* graphic mode.  The values and their meanings are as               *
* follows:                                                           *
*
```

*	VALUE	MEANING	*
*	-----	-----	*
*	0	40 column text mode	*
*	1	Standard bitmap mode	*
*	2	Split screen bitmap mode	*
*	3	Multicolor bitmap mode	*
*	4	Split screen multicolor bitmap mode	*
*	5	80 column text mode	*
*			*
*****			

```

8182: JSR   $77DA      ;Check to ensure that the value is numeric and
                        ;If not then generate a 'TYPE MISMATCH' error
8185: JSR   $818C      ;Find out what graphics mode the computer is
                        ;In and place that value into the accumulator
8188: TAY
8189: JMP   $84D4      ;Then change it into a Floating Point number
818C: LDA   $D8        ;Get the flag for text / graphics mode
818E: CLC
818F: ROL
8190: ROL
8191: ROL
8192: ADC   #0         ;Place the carry in
8194: BIT   $D7        ;Test flag for 80/40 column mode ($80=80 Column)
8196: BPL   $819A      ;Branch if we are in the 40 column mode
8198: ADC   #5         ;Add 5 to indicate 80 column mode
819A: RTS
                        ;Exit
    
```

```

*****
*
*           BASIC RCLR FUNCTION
*
* Function syntax:  RCLR(X)
*
* NOTE:  X = a value of 0 - 6 that represents the
*         color source to read the current color
*         value from
*
* This function will return the current color value
* (1 - 16) of the color source specified by X.
* The possible values of X and their meanings are as
* follows:
*
*   VALUE      MEANING
*   -----
*   0          40 background color
*   1          Bitmap foreground
    
```



```

*      2      Multicolor 1 color      *
*      3      Multicolor 2 color      *
*      4      40 column border color  *
*      5      40 or 80 column character color *
*      6      80 column background color *
*
*****

```

```

819B: JSR  $87F7      ;Get the color source in the X register
819E: JSR  $A845      ;Switch to BANK 15
81A1: DEX                      ;Subtract one if the value was zero and then get
81A2: BMI  $81B9      ;The background color
81A4: DEX                      ;If the value was 1, get the HI-RES foreground
81A5: BMI  $81C1      ;Color (PIXEL COLOR)
81A7: DEX                      ;If the value was 2, then get the Multicolor 1
81A8: BMI  $81C6      ;Text or HI-RES color
81AA: DEX                      ;If the value was 3, then get the Multicolor 2
81AB: BMI  $81CB      ;Text or HI- RES color
81AD: DEX                      ;If the value was 4, then get the 40 column
81AE: BMI  $81D0      ;Border color
81B0: DEX                      ;If the value was 5, get the character color
81B1: BMI  $81D6      ;In 40 or 80 column text
81B3: DEX                      ;If the value was 6, then get the 80 column
81B4: BMI  $81DE      ;Background color
81B6: JMP  $7D28      ;If the value was greater then six
                        ;Then 'ILLEGAL QUANTITY ERROR'
81B9: LDA  $D021      ;Get the background color value into accumulator
81BC: AND  #*01111111 ;Drop bit 7
81BE: JMP  $81EC      ;Convert the background color value into
                        ;Floating Point format
81C1: LDA  $86        ;Get the bitmap foreground color and
81C3: JMP  $81EC      ;Convert it to a Floating Point format
81C6: LDA  $84        ;Get the multicolor 1 color
81C8: JMP  $81EC      ;Then convert it to Floating Point format
81CB: LDA  $85        ;Get the multicolor 2 color
81CD: JMP  $81EC      ;Then convert it to Floating Point format
81D0: LDA  $D020      ;Get the border color
81D3: JMP  $81EC      ;Then convert it to Floating Point format
81D6: LDA  $F1        ;Get the character color
81D8: BIT  $D7        ;See if you are in the 40 or 80 column screen
81DA: BPL  $81EC      ;If you are in 40 columns, branch to convert the
                        ;Character color into Floating Point format
81DC: BMI  $81E6      ;If you are in 80 columns, branch to convert the
                        ;Foreground color into Floating Point format
81DE: LDA  #26        ;Read the foreground/background color
81E0: STA  $D600      ;From the 8563
81E3: LDA  $D601      ;80 column chip

```

```

81E6: AND  #\$0F      ;Drop bit's 4-7 (FOREGROUND COLOR)
81E8: TAX                ;Move the accumulator to the X register
                        ;As an index pointer
81E9: LDA  \$81F3,X     ;Get the corresponding color value
81EC: AND  #\$0F      ;Drop bit's 4-7 (FOREGROUND COLOR)
81EE: TAY                ;Place the color value in the Y register
81EF: INY                ;Add one to the color value
81F0: JMP  \$84D4       ;Convert the value in the Y register into
                        ;Floating point format and exit

```

```

*****
*
* This table is used to convert the RGBI values from
* the 80 column chip to the proper BASIC color code.
*
* Example: The RGBI value to produce the color RED is
*          8. Using that value as an index to the
*          data table, the corresponding BASIC color
*          code would be 3, which is derived by getting*
*          the value from the table and adding one.
*
*****

```

```

81F3: .BYTE $00,$0C,$06
81F6: .BYTE $0E,$05,$0D
81F9: .BYTE $0B,$03,$02
81FC: .BYTE $0A,$08,$04
81FF: .BYTE $09,$07,$0F
8202: .BYTE $01

```

```

*****
*
*          BASIC JOY FUNCTION
*
* Function syntax: JOY (joyport #)
*
* This function will return the current state of the
* specified joystick port.
*
* The values returned by this function are:
*
*   0 = Center      1 = North      2 = Northeast
*   3 = East        4 = Southeast  5 = South
*   6 = Southwest   7 = West       8 = Northwest
*
* If the fire button is pressed, the corresponding
* value above will have 128 added to it.
*

```

```

* Example: A value of 129 indicates the fire button *
* is being pressed while the joystick is *
* held in the North position. *
* *
*****

```

```

8203: JSR   $87F7      ;Convert the ASCII joyport number to
                    ;A HEX value and store it into the X register
8206: DEX
8207: CPX   #2        ;If the value is less than 1 or greater than 2
8209: BCS   $823F     ;Branch to generate an 'ILLEGAL QUANTITY' error
820B: TXA
820C: EOR   #%00000001 ;Invert bit 0
820E: TAX
820F: PHP
8210: JSR   $A845     ;Switch to BANK 15
8213: SEI
8214: LDA   $DC00     ;Get the current value of CIA1 Data Port A
8217: PHA
8218: LDY   #$FF      ;Initialize CIA1
821A: STY   $DC00
821D: LDA   $DC00,X   ;Wait until the values
8220: CMP   $DC00,X   ;Have
8223: BNE   $821D     ;Stabilized
8225: TAX
8226: PLA
8227: STA   $DC00     ;Of CIA1
822A: TXA
822B: PLP
822C: AND   #%00001111 ;Get back the Status Register
                    ;Drop bits 7-4 to get the Joystick values
822E: TAY
822F: LDA   $823D,Y   ;Save the value to the Y register as an index
                    ;Get the corresponding direction value from the
                    ;Table
8232: TAY
                    ;Save it into the Y register
8233: TXA
                    ;Check to see if the fire button
8234: AND   #%00010000 ;Is being pressed
8236: BNE   $823C     ;If not, then exit
8238: TYA
                    ;Add 128 to Joystick direction value
8239: ORA   #$80
                    ;To indicate that the fire button
823B: TAY
                    ;Is pressed
823C: JMP   $84D4     ;Convert the final value to Floating Point
                    ;Format and exit
823F: JMP   $7D28     ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
* This table is used by the JOY (X) function above to
* generate the corresponding HEX joystick direction
* values.
*
*****

8242: .BYTE $04,$02,$03 ;SE, NE, E
8245: .BYTE $00,$06,$08 ;Center, SW, NW
824A: .BYTE $07,$00,$05 ;W, Center, S
824D: .BYTE $01,$00 ;N, center

*****
*
* BASIC POT FUNCTION
*
* Function syntax: POT(X)
*
* NOTE: If X = 1, then POT returns the position value
* of paddle number one.
*
* If X = 2, then POT returns the position value
* of paddle number two.
*
* If X = 3, then POT returns the position value
* of paddle number three.
*
* If X = 4, then POT returns the position value
* of paddle number four.
*
* This function returns the position value of the
* paddle number specified by X.
*
*****

824D: JSR $7956 ;Check for the closing parenthesis ')', if it is
;Not found, then generate a 'SYNTAX' error
8250: JSR $87F7 ;Convert the Paddle number to a hex value and
;Store it into the X register
8253: DEX ;Subtract one from the Paddle number
8254: CPX #$04 ;If the Paddle number is less than 1
8256: BCS $82AB ;Or Greater than 4, then generate
;An 'ILLEGAL QUANTITY' error
8258: JSR $A845 ;Switch to BANK 15 to switch the I/O in
825B: LDY #0 ;Clear the index to the Paddle register
825D: TXA ;Transfer the Paddle number minus

```

```

;One to the accumulator
825E: LDX  #$40      ;Value for Paddle set A (Port #1)
8260: LSR           ;Divide the Paddle number by 2 to calculate
;The correct register number
8261: BCC  $8264     ;If POT number is equal to 1 or 3, then branch
8263: INY
8264: LSR
8265: BCC  $8269     ;If POT number is equal to 1 or 2, then branch
8267: LDX  #$80      ;Value for Paddle set B (Port #2)
8269: STX  $12B1     ;Save i ($40 = Paddle set A,$80 = Paddle set B)
826C: PHP           ;Save the Status register
826D: SEI           ;Stop all the background operations
826E: LDA  $DC00     ;Save the current value of CIA1
8271: PHA           ;Onto the stack
8272: STX  $DC00     ;Select the proper Paddle register to read
8275: LDX  #0        ;Delay loop
8277: INX           ;For the stabilization
8278: BNE  $8277     ;Of the values
827A: LDA  $D419,Y   ;Wait until the values
827D: CMP  $D419,Y   ;Have stabilized
8280: BNE  $827A
8282: STA  $12B2     ;Save the current Paddle value
8285: LDX  #0
8287: BIT  $12B1     ;Check which paddle set
828A: BMI  $828D     ;If it is set B, then branch
828C: INX           ;Increment the index by one
828D: LDA  #$04      ;Bit number of Fire button
828F: DEY           ;Pot register number
8290: BMI  $8293     ;If it is set A, then branch
8292: ASL           ;Multiply the value by 2, if it is set B
8293: LDY  #$FF      ;Initialize
8295: STY  $DC00     ;CIA1
8298: INY           ;To zero
8299: AND  $DC00,X   ;Mask to check the Fire button
829C: BNE  $829F     ;If the Fire button is pressed, then branch
829E: INY           ;The value will be 1 if button is not pressed
829F: PLA           ;Get the original value of CIA1 from the stack
82A0: STA  $DC00     ;Store it to CIA1
82A3: TYA           ;Save the Y register to the accumulator
82A4: LDY  $12B2     ;Load the paddle register value into Y register
82A7: PLP           ;Restore the flags
82A8: JMP  $84C9     ;Convert value to Floating Point format and exit
82AB: JMP  $7D28     ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
*                               *
*          BASIC PEN FUNCTION   *
*                               *
* Function syntax:  PEN (X)     *
*                               *
* NOTE:  If X = 0, then PEN returns the value of the *
*        X coordinate of the light pen position.    *
*                               *
*        If X = 1, then PEN returns the value of the *
*        Y coordinate of the light pen position.    *
*                               *
*        If X = 2, then PEN returns the value of the *
*        X coordinate of the 80 column display.     *
*                               *
*        If X = 3, then PEN returns the value of the *
*        Y coordinate of the 80 column display.     *
*                               *
*        If X = 4, then PEN returns the value of the *
*        light pen trigger.                               *
*                               *
* This function returns the current X & Y coordinates *
* of the Light Pen.                                     *
*                               *
*****

```

```

82AE: JSR   $7956      ;Check for a closing parenthesis ')'
                        ;And if it is not found, then generate a
                        ;'SYNTAX' error

82B1: JSR   $87F7      ;Get the mode value into the X register
82B4: CPX   #5         ;If the value is larger than 4,
82B6: BCS   $82AB      ;Then generate an 'ILLEGAL QUANTITY' error
82B8: CPX   #$02       ;If the mode value is larger than 1, then
82BA: BCS   $82D6      ;Branch to get the values for 80 column screen
82BC: LDY   $11E9,X    ;Get the current Light Pen X or Y coordinate
82BF: STY   $12B1      ;Save it
82C2: LDA   #0         ;Reset the current X or Y coordinate
82C4: STA   $11E9,X    ;To zero
82C7: CPX   #0         ;If the Y coordinate was selected,
82C9: BNE   $82D0      ;Then branch
82CB: ASL   $12B1      ;If the X coordinate was selected, then
82CE: ADC   #0         ;Multiply the coordinate by 2 and add
                        ;The carry ,if any, to the accumulator

82D0: LDY   $12B1      ;Get the coordinate
82D3: JMP   $84C9      ;And convert Y, A (LSB,MSB) to Floating Point
                        ;Format and exit

```

```

82D6: JSR   $A845      ;Switch to BANK 15
82D9: CPX   #$04      ;Branch to read the
82DB: BEQ   $82ED      ;Current Light Pen trigger value

*****
*
*   This routine is used to read the Light Pen
*   values for the 80 column screen.
*
*   NOTE: The values returned are the row and column
*   coordinates instead of the pixel coordinates.
*
*****

82DD: LDY   #$11      ;Select Register 17 of the VDC for the
                        ;Light Pen X coordinate
82DF: CPX   #$02      ;Check if the X coordinate was selected
82E1: BEQ   $82E4      ;If so, then branch to read the X coordinate
82E3: DEY                       ;If not, then select Register 16 of the VDC for
                        ;The Light Pen Y coordinate
82E4: STY   $D600      ;Select the Register
82E7: LDY   $D601      ;Read the current value from the Register
82EA: JMP   $84D4      ;Convert value to Floating Point format and exit

*****
*
*   This routine is used to read the trigger value
*   of the light pen.
*
*****

82ED: LDA   $D600      ;Get the value of the trigger
82F0: LDY   #0         ;Value for trigger not pressed
82F2: AND   #%01000000 ;Check if trigger is pressed
82F4: BEQ   $82F7      ;If not, then exit with a zero
82F6: INY                       ;If trigger is pressed,
82F7: JMP   $84D4      ;Then exit with a 1 to denote that the
                        ;Trigger is pressed

```

```

*****
*
*           BASIC POINTER FUNCTION
*
* Command syntax:  POINTER(var.)
*
* NOTE:  var. = the name of the variable of which the
*         the address is to be found
*
* This function returns the decimal representation of
* the address of the variable specified by 'var.'.
*
* The address that is returned by the POINTER command
* is the address of the variable's descriptor in
* RAM BANK 1.
*
*****

```

```

82FA: JSR   $0380      ;Get the first character after the command
82FD: JSR   $7959      ;Check for an opening parenthesis '(' and if it
                        ;Is not found, then generate a 'SYNTAX' error
8300: JSR   $7B3C      ;Check if the first character in parentheses is
8303: BCC   $831B      ;A letter. If not then generate a 'SYNTAX' error
8305: JSR   $7AAF      ;Search for the variable and return with
                        ;The A,Y registers containing the LSB,MSB of the
                        ;Address to the variable's descriptor in BANK 1
8308: TAX
8309: TYA
                        ;Temporarily save the LSB into the X register
830A: PHA
                        ;Save the MSB of the address
                        ;Onto the stack
830B: JSR   $7956      ;Get the last char. and if it is not a closing
                        ;Parenthesis ')', then generate a 'SYNTAX' error
830E: TXA
                        ;Restore the LSB into the accumulator
830F: TAY
                        ;And move it into the Y register
8310: PLA
                        ;Get the MSB of the address of the descriptor
8311: CMP   #$03
                        ;Check if the variable was TI, TI$, DS, or DS$
8313: BNE   $8318      ;If not, convert the address of the descriptor
                        ;Into Floating Point format
8315: LDA   #0
                        ;Return a zero value
8317: TAY
                        ;If TI or TI$ was selected
8318: JMP   $84C9      ;Convert Y,A to Floating Point Format and exit
831B: JMP   $796C      ;Generate a 'SYNTAX ERROR'

```



```

*****
*
*          BASIC RSPRITE FUNCTION
*
* Command syntax:  RSPRITE(spr#,char)
*
* NOTE:  spr# =  sprite number
*        char =  characteristic value
*
* This function will return information about the
* characteristic specified by 'char' of the sprite
* whose number is specified by 'spr#'.  The value of
* the sprite number is between 1 - 8 and the
* characteristic value is between 0 - 5.  The meaning
* of the characteristic value is as follows:
*
* CHARACTERISTIC      MEANING OF VALUE
*   VALUE            RETURNED BY RSPRITE
*   -----
*       0             Sprite enabled (1), disabled (0)
*       1             Sprite color (1 - 16)
*       2             Displayed in front of (0)
*                    or behind (1) the other objects
*                    displayed on the screen
*       3             Expand in the X direction (y=1, n=0)
*       4             Expand in the Y direction (y=1, n=0)
*       5             Multicolor sprite          (y=1, n=0)
*
* NOTE:  This function has a flaw in that, even though
* there are only eight sprites available on the
* 128, a value from 1 to 16 may be specified
* for the 'spr#' parameter.
*
*****

```

```

831E: JSR  $87F7      ;Convert the sprite number to an integer and
                    ;Store the value in the X register
8321: DEX
                    ;Subtract one from the sprite number
8322: CPX  #$10
                    ;If the sprite number is larger
8324: BCS  $8358
                    ;Than 16, then 'ILLEGAL QUANTITY' error
8326: TXA
                    ;Save the sprite number
8327: PHA
                    ;Temporarily to the stack
8328: JSR  $795C
                    ;Check for a comma after the sprite number
832B: JSR  $87F4
                    ;Get the second parameter into the X register
832E: JSR  $7956
                    ;Check for a closing parenthesis ')'
8331: CPX  #$06
                    ;If the second parameter is larger
8333: BCS  $8358
                    ;Than 5, then 'ILLEGAL QUANTITY' error

```

```

8335: PLA                ;Get the sprite number off of the stack
8336: TAY                ;And transfer it to the Y register
8337: JSR   $A845        ;Switch to BANK 15
833A: LDA   $D027,Y     ;Get the color value of the specified sprite
833D: AND   #$0F         ;Drop the high nybble
833F: CLC                ;Of the color value and add one to that value
8340: ADC   #1           ;To get the BASIC color code value
8342: CPX   #1           ;If the second parameter was
8344: BEQ   $8354        ;A one, then exit
8346: LDA   $835B,X     ;Get the offset to the specified sprite
8349: TAX                ;Characteristic and save it as the index
834A: LDA   $6CB3,Y     ;Get appropriate mask value for the specified
834D: AND   $D000,X     ;Sprite and drop the unwanted bits
8350: BEQ   $8354        ;If the value is zero, then exit
8352: LDA   #1           ;If the value is greater than zero, then make
8354: TAY                ;The value equal to one
8355: JMP   $84D4        ;Convert the value to Floating Point Format
                        ;And exit
8358: JMP   $7D28        ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
*                               TABLE FOR RSPRITE
*
* This table contains the LSB of the address for each
* of the sprite characteristics. The MSB of the
* address is $D0.
*
*****

```

```

835B: .BYTE $15,$27,$1B
835E: .BYTE $1D,$17,$1C

```

```

*****
*
*                               BASIC RSPCOLOR FUNCTION
*
* Command syntax: RSPCOLOR (X)
*
* NOTE: If X equals one, then RSPCOLOR returns the
*       sprite multicolor 1 value
*
*       If X equals two, then RSPCOLOR returns the
*       sprite multicolor 2 value
*
*****

```

```

* This function returns a value of 1 to 16 that      *
* represents the multicolor 1 or multicolor 2 value *
* as specified by the value of X.                  *
*                                                    *
*****
    
```

```

8361: JSR  $7956      ;Check for an opening parenthesis '('
8364: JSR  $87F7      ;Get the parameter into the X register
8367: DEX                    ;Subtract one from the value
8368: CPX  #$02      ;If the parameter is less than 1 or greater
836A: BCS  $8379      ;Than 2, then 'ILLEGAL QUANTITY' error
836C: JSR  $A845      ;Switch to BANK 15
836F: LDA  $D025,X    ;Get the current multicolor value
8372: AND  #$0F      ;And drop the high nybble of the color value
8374: TAY                    ;Add one to the color value
8375: INY                    ;To make the BASIC color code
8376: JMP  $84D4      ;Convert the color value to Floating Point
                        ;Format and exit
8379: JMP  $7D28      ;Generate an 'ILLEGAL QUANTITY' error
    
```

```

*****
*                                                    *
*                BASIC BUMP FUNCTION                *
*                                                    *
* Function syntax:  BUMP (X)                        *
*                                                    *
* NOTE:  If X equals one, then the value returned by *
*        BUMP represents which sprites, if any, have *
*        collided with each other.                  *
*                                                    *
*        If X equals two, then the value returned by *
*        BUMP represents which sprites, if any, have *
*        collided with an object on the screen.     *
*                                                    *
* This function returns sprite collision information  *
* as specified by the value of X.                  *
*                                                    *
*****
    
```

```

837C: JSR  $7956      ;Check for an opening parenthesis '('
837F: JSR  $87F7      ;Get the parameter into the X register
8382: DEX                    ;Subtract one from the value
8383: CPX  #$02      ;If the parameter is less than 1 or
8385: BCS  $8394      ;Greater than 2, then 'ILLEGAL QUANTITY' error
8387: SEI                    ;Disable the background operations
8388: LDY  $11E7,X    ;Get the current sprite collision value
838B: LDA  #0          ;Clear the register
    
```

```

838D: STA  $11E7,X    ;After reading it
8390: CLI                      ;Restart the background operations
8391: JMP  $84D4      ;Convert the collision value into Floating Point
                        ;Format and exit
8394: JMP  $7D28      ;Generate an 'ILLEGAL QUANTITY' error
    
```

```

*****
*
*              BASIC RSPPOS FUNCTION
*
* Function syntax:  RSPPOS (spr#,P)
*
* NOTE:  If P equals zero, then RSPPOS returns the
*        current X coordinate of the sprite specified
*        by the value of 'spr#'.
*
*        If P equals one, then RSPPOS returns the
*        current Y coordinate of the sprite specified
*        by the value of 'spr#'.
*
*        If P equals two, then RSPPOS returns the
*        speed value (0-15) of the sprite specified
*        by the value of 'spr#'.
*
* This function will return speed and position
* information of the sprite specified by the value of
* 'spr#'.
*
* NOTE:  This function has the same flaw that the
*        RSPRITE function does, in that even though
*        there are only eight sprites available with
*        the 128, a value from 1 to 16 may be
*        specified for 'spr#'.
*
*****
    
```

```

8397: JSR  $87F7      ;Get the sprite number into the X register
839A: DEX                      ;Subtract one from the sprite value
839B: CPX  #$10        ;If the sprite number is less than 1 or
839D: BCS  $83DE      ;Greater than 16, then 'ILLEGAL QUANTITY' error
839F: TXA                      ;Temporarily save the sprite number
83A0: PHA                      ;Onto the stack
83A1: JSR  $795C      ;Check for a comma
83A4: JSR  $87F4      ;Get the second parameter into the X register
83A7: JSR  $7956      ;Check for a closing parenthesis ')'
83AA: CPX  #$03        ;If the second parameter is greater
83AC: BCS  $83DE      ;Than 2, then 'ILLEGAL QUANTITY' error
    
```

```

83AE: PLA                ;Get the sprite number back
83AF: TAY                ;Off of the stack
83B0: CPX    #$02        ;If the second parameter is not equal to 2,
83B2: BNE    $83BD       ;Then branch to get the sprite coordinates
83B4: LDX    $6DD9,Y     ;Get the index to the specified sprite
83B7: LDY    $117E,X     ;Get the speed value of the specified sprite
83BA: JMP    $84D4       ;Convert it to Floating Point Format and exit
83BD: SEI                ;Stop all background operations
83BE: LDA    $6CB3,Y     ;Get the mask for the specified sprite
83C1: AND    $11E6       ;See if the X coordinate is larger than 255
83C4: BEQ    $83C8       ;And branch if it is not
83C6: LDA    #1          ;If it is larger than 255, then save a one
83C8: PHA                ;As the MSB of the X coordinate value
83C9: TYA
83CA: ASL                ;Multiply the sprite number by 2
83CB: TAY                ;And save it
83CC: TXA                ;As an index to the sprite coordinate
83CD: LSR                ;Divide the second parameter by 2
83CE: BCC    $83D5       ;And branch if we are need the X coordinate
83D0: INY                ;Increment the index to the Y coordinate
83D1: PLA                ;Remove the X coordinate MSB from the stack
83D2: LDA    #0          ;And replace it
83D4: PHA                ;With a zero
83D5: LDA    $11D6,Y     ;Get the LSB of the sprite's coordinate
83D8: CLI                ;And restart all of the background operations
83D9: TAY                ;Save the coordinate's LSB to the Y register
83DA: PLA                ;And save the MSB to the accumulator
83DB: JMP    $84C9       ;Convert the value to Floating Point and exit
83DE: JMP    $7D28       ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
*          BASIC XOR FUNCTION
*
* Function syntax:  XOR (x1,x2)
*
* NOTE:  x1, x2 = an unsigned value between 0 - 65535
*
* This function returns a value that is equal to the
* value of 'x1' eXclusive ORed with the value of 'x2'.
*
*****

```

```

83E1: LDA    $16          ;Save these
83E3: PHA                ;Two values
83E4: LDA    $17          ;Temporarily
83E6: PHA                ;Onto the stack

```

```

83E7: JSR   $77DA      ;Check on numeral
83EA: JSR   $8815      ;Get the first value into the accumulator and
                        ;The Y register
83ED: PHA
83EE: TYA      ;Two values
83EF: PHA      ;Onto the stack
83F0: JSR   $880F      ;Check for a comma and get the second value
83F3: JSR   $7956      ;Check for a closing parenthesis ')'
83F6: PLA      ;Get the LSB of the first value and ex-
83F7: EOR   $16       ;clusive OR it with the LSB of the second value
83F9: TAY      ;Save the result to the Y register
83FA: PLA      ;Get the MSB of the first value and exclusive
83FB: EOR   $17       ;OR it with the MSB of the second value
83FD: JSR   $84C9      ;Convert the result to Floating Point Format
8400: PLA      ;Restore locations $16, $17
8401: STA   $17       ;To their
8403: PLA      ;Original
8404: STA   $16       ;Values
8406: RTS      ;And exit

```

```

*****
*
*          BASIC RWINDOW FUNCTION          *
*
*  Function syntax:  RWINDOW (X)          *
*
*  NOTE:  If X equals zero, then RWINDOW returns a
*         value that represents the number of lines in
*         the current window.
*
*         If X equals one, then RWINDOW returns a
*         value that represents the number of rows in
*         the current window.
*
*         If X equals two, then RWINDOW will return a
*         value that of 40 if you are in the 40 column
*         mode or a value of 80 if you are in the 80
*         column mode.
*
*  This function returns dimension information about
*  the current window.
*
*****

```

```

8407: JSR   $7956      ;Check for a closing parenthesis ')'
840A: JSR   $87F7      ;Get the parameter into the X register
840D: CPX   #$02      ;Branch if the parameter is equal to 2

```

```

840F: BEQ  $8425      ;To get the current screen output format
8411: BCS  $8431      ;If the parameter is greater than two,
                        ;'ILLEGAL QUANTITY' error
8413: CPX  #0         ;If the parameter is equal to one,
8415: BNE  $841E      ;Then branch to get the number of columns
                        ;In the current window
8417: LDA  $E4        ;Subtract the upper limit of the window
8419: SEC                        ;From the lower limit of the window
841A: SBC  $E5        ;To get the number of rows minus one
                        ;Of the current window
841C: BCS  $842D      ;And exit
841E: LDA  $E7        ;Subtract the right margin of the window
8420: SEC                        ;From the left margin of the window
8421: SBC  $E6        ;To get the number of columns minus one
                        ;Of the current window
8423: BCS  $842D      ;And exit
8425: LDA  #$28        ;Value of 40 for the 40 column mode
8427: BIT  $D7        ;Are we in the 40 or 80 column mode
8429: BPL  $842D      ;Exit with 40 for the 40 column mode
842B: LDA  #$50        ;Value of 80 for the 80 column mode
842D: TAY                        ;Move the value to the Y register for conversion
842E: JMP  $84D4      ;Convert the value to Floating Point Format
8431: JMP  $7D28      ;Generate an 'ILLEGAL QUANTITY' error
    
```

```

*****
*
*
*           BASIC RND FUNCTION
*
* Function syntax:  RND (X)
*
* NOTE:  If X is positive, the next RND value is
*        generated by multiplying the seed value by
*        the multiplication constant for RND (RMULC),
*        adding the addition constant for RND (RADDCC),
*        and then scrambling the resulting bytes.
*
*        If X is negative, the value of X is scrambled
*        and the resulting value then becomes the new
*        seed. This allows duplication of the random
*        numbers that are generated.
*
*        If X is equal to zero, the four bytes of the
*        Floating Point accumulator are loaded with the
*        LSB and MSB of CIA1 TIMER A and the seconds &
*        tenths of a second clock registers of CIA1.
*
*****
    
```

---

```
8434: JSR   $8C57      ;Get the sign of the Floating Point accumulator
                        ;(FAC1)
8437: BMI   $846A      ;If it's negative, then branch to use the
                        ;Parameter as a seed value
8439: BNE   $8455      ;If it is greater then 0, then branch to use the
                        ;Old seed value
843B: JSR   $A845      ;Switch to BANK 15
843E: LDA   $DC06      ;Get CIA #1 timer B LSB
8441: STA   $64        ;Save it
8443: LDA   $DC07      ;Get CIA #1 timer B MSB
8446: STA   $66        ;Save it
8448: LDA   $DC04      ;Get CIA #1 timer A LSB
844B: STA   $65        ;Save it
844D: LDA   $DC05      ;Get CIA #1 timer A MSB
8450: STA   $67        ;Save it
8452: JMP   $847A      ;Round it off and assign it to a variable
8455: LDA   #$1B       ;Set the accumulator and Y register to point
                        ;To RNDX ($121B)
8457: LDY   #$12       ;Which holds the current seed value
8459: JSR   $8BD4      ;And store the seed value in the FAC1
845C: LDA   #$90       ;Set the accumulator and Y register to $8490
845E: LDY   #$84       ;Which holds the constant for the RND factor
8460: JSR   $8A08      ;Multiply the value by FAC1
                        ;(FAC1 = FAC1 * CONSTANT)
8463: LDA   #$95       ;Set the accumulator and Y register to $8495
8465: LDY   #$84       ;Which holds the second constant for RND
8467: JSR   $8A12      ;Add the value to FAC1 (FAC1 = FAC1 + CONSTANT)
846A: LDX   $67        ;Reverse FAC1, by moving M4
846C: LDA   $64        ;To M1
846E: STA   $67        ;
8470: STX   $64        ;M1 to M4
8472: LDX   $65        ;
8474: LDA   $66        ;M3 to M2
8476: STA   $65        ;
8478: STX   $66        ;And M2 to M3
847A: LDA   #0        ;Make the sign of the Floating Point accumulator
847C: STA   $68        ;(FAC1) positive
847E: LDA   $63        ;Use the current exponent of FAC1
8480: STA   $71        ;As the rounding value
8482: LDA   #$80       ;Make the exponent of FAC1
8484: STA   $63        ;Equal to -1
8486: JSR   $88B6      ;Round off FAC1
8489: LDX   #$1B       ;Set up the X and Y registers
848B: LDY   #$12       ;To point to location $121B
848D: JMP   $8C00      ;Store the value of FAC1 to $121B and exit
```



```

*****
*
*                               RMULC
*
*           Random MULTiplication Constant
*
* The constant value in this table is stored in
* Floating Point format and is used as a multiplier to
* the seed value for obtaining the next RND value.
*
*****

```

```

      EX  M1  M2  M3  M4
      --  --  --  --  --
8490: .BYTE $98,$35,$44,$7A,$00 ;11879546

```

```

*****
*
*                               RADD C
*
*           Random ADDition Constant
*
* The constant value in this table is stored in
* Floating Point format and is added to the seed
* value in the process of obtaining a RND value.
*
*****

```

```

      EX  M1  M2  M3  M4
      --  --  --  --  --
8495: .BYTE $68,$28,$B1,$46,$00 ;3.92767774E-4

```

```

*****
*
*                               N32768
*
*           Negative value of 32768
*
* This Floating Point number is used for range
* checking during the process of converting a Floating
* Point number to a signed integer. The minimum
* allowable value is -32768.
*
*****

```

```

      EX  M1  M2  M3  M4
      --  --  --  --  --

```

```
849A:  .BYTE $90,$80,$00,$00,$00 ; -32768
```

```

*****
*
*                               FPNINT
*
* Convert a Floating Point Number into a signed
* INTeGer.
*
* The Floating Point value to be converted must be
* in FAC1. Upon exiting this routine, the accumulator
* will hold the LSB and the Y register the MSB of the
* resulting integer.
*
*****

```

```

849F:  JSR  $84B4      ;Change the Floating Point Accumulator value
                        ;Into an integer value
84A2:  LDA  $66        ;Get the LSB value of the integer
84A4:  LDY  $67        ;Get the MSB value of the integer
84A6:  RTS

```

```

*****
*
*                               INTIDX
*
* Change a Floating Point Number to a positive INTeGer.*
*
* The Floating Point value to be converted must be
* in FAC1. Upon exiting this routine, the accumulator
* will hold the LSB and the Y register the MSB of the
* resulting integer.
*
* The integer must be positive and it cannot be
* greater than a value of 32768. If the integer is
* either negative, greater than 32768, or both, then
* an 'ILLEGAL QUANTITY' error will result.
*
*****

```

```

84A7:  JSR  $0380      ;Get the next character from the BASIC text
84AA:  JSR  $77EF      ;Evaluate the expression
84AD:  JSR  $77DA      ;And make sure the expression is numeric
84B0:  LDA  $68        ;Check the sign of FAC1
84B2:  BMI  $84C1      ;And branch if the value in FAC1 is negative

```

```

*****
*
*                               AYINT
*
* Convert the Floating Point value in FAC1 to a two
* byte integer whose value is stored in $66, $67.
*
*****

```

```

84B4: LDA  $63      ;Check if the value in FAC1
84B6: CMP  #$90      ;Is greater than 32768
84B8: BCC  $84C6     ;If the value is less than 32768, then
                        ;Convert FAC1 to an integer and exit
84BA: LDA  #$9A      ;If the value is equal to or greater than 32768
84BC: LDY  #$84      ;Check to see if it is a value of -32768
84BE: JSR  $8C87     ;Compare the Floating Point Accumulator (FAC1)
                        ;To the CONSTANT value of -32768
84C1: BEQ  $84C6     ;Exit if the value is in the range of -32768 to
                        ;32768
84C3: JMP  $7D28     ;If the value is out of this range then
                        ;Generate an 'ILLEGAL QUANTITY' error
84C6: JMP  $8CC7     ;Convert FAC1 to an integer and exit

```

```

*****
*
*                               GIVAYF1
*
* GIVE the Accumulator and the Y register to be
* converted to Floating point format.
*
*****

```

```

84C9: JSR  $84E5     ;Set up the parameters for the conversion from
                        ;Fixed to Floating Point
84CC: SEC                      ;Set the carry to generate a positive value
84CD: JMP  $8C75     ;Convert the value in locations $64, $65
                        ;To Floating Point format

```

```

*****
*
*                               BASIC POS FUNCTION
*
* Function syntax:  POS (X)
*
* NOTE:  X = a value that must be specified but is
*         ignored by the POS function.
*
* This function returns a value that indicates where
* the cursor is within the screen window.
*****

```

```

84D0: SEC                ;Set the carry flag for the GET cursor position
84D1: JSR  $928D         ;Get the cursor position into X and Y registers
84D4: LDA  #0           ;Convert the position of the cursor
84D6: JMP  $793C         ;From integer format to Floating Point format

```

```

*****
*
*                               ERRDIR
*
*                               ERRor if in the DIREct mode
*
* This routine checks to see if we are in the DIRECT
* mode and if so, then the routine issues an error.
* This routine is called by other routines that do not
* allow certain commands or functions to be executed
* in the DIRECT mode.
*
*****

```

```

84D9: BIT  $7F          ;Check for the DIRECT mode
84DB: BMI  $84EF         ;If we are in the PROGRAM mode, then exit
84DD: LDX  #21          ;'ILLEGAL DIRECT' error
84DF: .BYTE $2C         ;Mask to fall through to
84E0: LDX  #27          ;'UNDEF'D FUNCTION' error
84E2: JMP  $4D3C         ;Jump to the error message routine

```

```

*****
*
*                               GIVAYF2
*
* This routine sets up parameters in order to convert
* a Fixed Point integer value (-32768 to 32768) to
* Floating Point format.
*****

```

```

* Enter this routine with the accumulator and      *
* Y register containing the MSB and LSB of the value *
* to be converted.                                *
*                                                  *
*****

```

```

84E5: LDX #0 ;Set VALTYP
84E7: STX $0F ;To numeric
84E9: STA $64 ;Save the MSB
84EB: STY $65 ;And the LSB of the value to convert
84ED: LDX #$90 ;Set the exponent to maximum allowable of 32768
84EF: RTS ;Exit

```

```

*****
*
* ERRPRG
*
* Error if in the PRoGram mode
*
* This routine checks to see if we are in the PROGRAM
* mode and if so, then the routine issues an error.
* This routine is called by other routines that do not
* allow certain commands or functions to be executed
* while in the PROGRAM mode.
*
*****

```

```

84F0: BIT $7F ;Check to see if we are in the PROGRAM mode and
;If so,
84F2: BMI $84F5 ;Branch to generate a 'DIRECT MODE ONLY' error
84F4: RTS ;Exit
84F5: LDX #34 ;'DIRECT MODE ONLY' error number
84F7: JMP $4D3C ;Jump to the error message routine

```

```

*****
*
*                               BASIC DEF FN COMMAND
*
* Command syntax:  DEF FN nm (var.) = exp.
*
* NOTE:  nm = the name of the function preceded by
*         the letters 'FN' and followed by any
*         alphanumeric name which begins with a letter.
*         (example: FNA)
*
*         var. = a dummy numeric variable
*
*         exp. = the formula to be defined
*
* The function can be executed by replacing 'var.'
* with any number as shown below.
*
* EXAMPLE:  10 DEF FNA(Y) = 2 * Y
*           20 PRINT FNA(8)
*
* In this example, every occurrence of the variable 'Y'
* is replaced by the number '8' as defined in line 20.
* Therefore, the result of line 20 would be 16.
*
*****

```

```

84FA: JSR   $8528      ;Check on the proper syntax and variable type
84FD: JSR   $84D9      ;Insure that we are in the PROGRAM mode
8500: JSR   $7959      ;Check for an opening parenthesis '('
8503: LDA   #$80       ;Set the flag to prevent integer
8505: STA   $12        ;And string variables
8507: JSR   $7AAF      ;Get the variable within the parentheses
850A: JSR   $77DA      ;Ensure that it is a numeric variable
850D: JSR   $7956      ;Check for a closing parenthesis ')'
8510: LDA   #$B2       ;Check for the equal sign '='
8512: JSR   $795E      ;And get the next character
8515: PHA                ;Save the first character after the equal sign
8516: LDA   $4A        ;Save the address of the
8518: PHA                ;Variable in parentheses
8519: LDA   $49        ;Onto the
851B: PHA                ;Stack

```

```

851C: LDA   $3E           ;Save TXTPTR which points to the first
851E: PHA                   ;Character of the argument
851F: LDA   $3D           ;Onto the
8521: PHA                   ;Stack
8522: JSR   $528F         ;Search for the next statement or an end of line
                               ;Flag
8525: JMP   $85A0         ;Restore the original value of
                               ;The variable in parentheses

```

```

*****
*
*                               GETFNM
*
*                               GET the Function NaMe
*
* This routine checks the FN and DEF syntax and if
* an error is encountered, then a 'SYNTAX' error
* message is generated.
*
*****

```

```

8528: LDA   #$A5           ;Check for the token for 'FN'
852A: JSR   $795E         ;And get the Function Name
852D: ORA   #$80           ;Set bit 7 of the Function Name
852F: STA   $12           ;And save it
8531: JSR   $7AB6         ;Search for the Function Name and ensure that it
                               ;Is not an integer or a string
8534: STA   $50           ;Save the LSB
8536: STY   $51           ;And the MSB of the address of the Function Name
8538: JMP   $77DA         ;Ensure that Function Name is a numeric variable

```

```

*****
*
*                               FNDOER
*
*                               BASIC FN FUNCTION
*
* Function syntax:  FN xx(x)
*
* This function returns the value that is developed
* by the function that was defined by the DEF FNxx
* function.
*
*****

```

```

853B: JSR  $8528      ;Check on the 'FN' syntax and get Function Name
853E: LDA  $51        ;Save the address of the
8540: PHA                    ;Descriptor for the Function Name
8541: LDA  $50        ;Onto the
8543: PHA                    ;Stack
8544: JSR  $7950      ;Evaluate the expression within parentheses and
                        ;Store the result in FAC1
8547: JSR  $77DA      ;Check to see if the expression is numeric and
                        ;If it is not, generate a 'TYPE MISMATCH' error
854A: PLA                    ;Restore the address of the descriptor for the
854B: STA  $50        ;Function Name and store it to locations $50,
                        ;$51
854D: PLA                    ;Note: This address is pointing to LSB of the
854E: STA  $51        ;Expression in the DEFine FuNction's descriptor
    
```

```

*****
*
* This routine gets the address of the variable
* descriptor.
*
* NOTE: This address points to the LSB of the
* exponent in the variable descriptor (byte 2).
*
*****
    
```

```

8550: LDY  #2          ;Skip over the expression's address
8552: JSR  $42CE      ;And get the LSB of the variable
8555: STA  $49        ;Descriptor and save it to location $49
8557: TAX                    ;Then save it in the X register (useless code)
8558: INY                    ;Move the Index register over one to
8559: JSR  $42CE      ;Get MSB of the variable descriptor's address
855C: BEQ  $84E0      ;And if it is equal to zero then generate
                        ;An 'UNDF FUNCTION' error
855E: STA  $4A        ;Save the MSB of the address of the variable
                        ;Descriptor's address
8560: INY                    ;Increment the index pointer so that the index
                        ;Register will point to MANTISSA 4 (byte 7) in
                        ;The variable descriptor
    
```

```

*****
*
* This routine saves the value of the variable that
* was specified in the original definition just in
* case a different variable is being used this time.
*
*****
    
```



```

8561: LDA   #$49       ;Set up an indirect address
8563: JSR   $03AB      ;And get a byte from the variable's descriptor
8566: PHA                   ;And save it onto the stack
8567: DEY                   ;(save bytes 2 thru 7)
8568: BPL   $8561      ;Continue until the value is completely saved
                        ;Onto the stack
    
```

```

*****
*
* This routine moves the value of the variable that
* was specified this time into the variable's
* descriptor that was originally specified. The
* original value has been preserved on the stack.
*
*****
    
```

```

856A: LDY   $4A       ;Transfer the value of the expression
856C: STA   $FF04      ;In parentheses to the variable descriptor
856F: JSR   $8C00      ;That was originally specified
8572: LDA   $3E       ;Save TXTPTR onto
8574: PHA                   ;The stack. Note: This address points to the
8575: LDA   $3D       ;First character after
8577: PHA                   ;The closing parenthesis
8578: JSR   $42CE      ;Get the address of the expression
857B: STA   $3D       ;In the DEF statement (bytes 2 & 3 in DEF
                        ;Descriptor)
857D: INY                   ;And save it as
857E: JSR   $42CE      ;The TXTPTR
8581: STA   $3E
8583: LDA   $4A       ;Save the descriptor address of the variable
8585: PHA                   ;In the parentheses of the DEF statement
8586: LDA   $49       ;Onto
8588: PHA                   ;The stack
8589: JSR   $77D7      ;Obtain the numeric value of the expression
                        ;That was specified in the DEF statement
                        ;And place the result into FAC1
858C: PLA                   ;Restore the descriptor's address
858D: STA   $50       ;Of the variable in parentheses of DEF statement
858F: PLA                   ;Into locations
8590: STA   $51       ;$50, $51
    
```

```

*****
*
* This section of code checks to ensure that the
* first character after the expression is an End of
* Line flag or and End of Statement flag (chain flag)
* If it is not either of these, then a 'SYNTAX' error
*
    
```

```

* will result. However, the error is not indicated *
* as being in the DEF statement, but, rather as being *
* in the line the FNxx(x) is used in. This is because *
* the line number where the DEF statement occurs is *
* not moved into OLDLIN at locations $3B, $3C. *
* *
*****

```

```

8592: JSR $0386 ;If the character after the expression
8595: BEQ $859A ;Is not a colon or an End of Statement, then
8597: JMP $796C ;Generate a 'SYNTAX' error
859A: PLA ;Else restore the TXTPTR back to pointing
859B: STA $3D ;To the first character
859D: PLA ;After the
859E: STA $3E ;Closing parenthesis
85A0: LDY #0 ;Restore the original
85A2: STA $FF04 ;Value of the variable
85A5: PLA ;In the parentheses that was used
85A6: STA ($50),Y ;In the DEF statement
85A8: INY
85A9: CPY #$05
85AB: BNE $85A5
85AD: RTS ;And exit with the function value in FAC1

```

```

*****
*
* BASIC STR$ FUNCTION *
*
* Function syntax: STR$ (X) *
*
* NOTE: X = the numeric value to be converted to a *
* string *
*
* This function takes the numeric value specified by *
* the argument X and converts it to its string *
* equivalent preceded by a space if the value is *
* positive or by a minus sign if the value is *
* negative. *
*
*****

```

```

85AE: JSR $77DA ;Make sure the VALTYP is numeric
85B1: LDY #0 ;Place the string in the buffer starting at $FF
85B3: JSR $8E44 ;Convert FAC1 to an ASCII string
85B6: PLA ;Pull the return address of
85B7: PLA ;The routine that called this one off the stack
85B8: LDA #$FF ;The string starts at $FF

```

```
85BA: LDY #0 ;Index the pointer
85BC: JMP $869A ;Set up the string and allocate space in memory
```

```
*****
*
*          BASIC CHR$ FUNCTION          *
*
* Function syntax: CHR$ (X)            *
*
* NOTE: X = CBM ASCII code number      *
*
* This function will return the character that
* corresponds to the numeric value of the argument
* 'X'. In order to accomplish this, this routine
* converts the ASCII value in the parentheses to a
* HEX numeric value. The routine then allocates a
* space in the string storage area and places the HEX
* value in that space.
*
*****
```

```
85BF: JSR $87F7 ;Convert the ASCII value to a numeric value
85C2: TXA ;Place the numeric value into the accumulator
85C3: PHA ;Save it onto the stack for recall later
85C4: LDA #1 ;Set string length of the string to one char.
85C6: JSR $8690 ;Allocate the room for the string in RAM BANK 1
;And create the descriptor in locations $63, $65
85C9: PLA ;Get the numeric value back off of the stack
85CA: LDY #0 ;Zero the index register
85CC: STA $FF04 ;Enable BANK 15 with RAM BANK 1 enabled
85CF: STA ($64),Y ;Save the numeric value to the allocated area
85D1: PLA ;Pull the address off the stack of
85D2: PLA ;The return address of the BASIC interpreter
85D3: JMP $86E3 ;Transfer the string's descriptor from $63 - $65
;Into the currently available temporary string
;Descriptor pointed to by location $18. Then
;Save the pointer that is in $18 to location $66
;To Point to the temporary string descriptor
;Which holds the string descriptor
```

```
*****
*
*          BASIC LEFT$ FUNCTION          *
*
* Function syntax: LEFT$ (string, first para.) *
*
```

```

* This function returns a string that consists of the *
* number of leftmost characters of the 'string' that *
* is determined by the 'first para.'. If the 'first *
* para.' is greater than the length of the 'string', *
* then the entire string is returned by the LEFT$ *
* function. *
* *
*****

```

```

85D6: JSR   $864D   ;Get the address of the string descriptor off of
                ;The stack and the first parameter
85D9: PHA                ;Save the first parameter onto the stack
85DA: JSR   $42D8   ;Get the length of the string from
85DD: STA   $79     ;String descriptor and save it into location $79
85DF: PLA                ;Get the length of the 'substring' requested by
                ;The first parameter and
85E0: CMP   $79     ;Compare to length of string in parentheses
85E2: TYA                ;Zero the accumulator (The zero in Y register
                ;Was placed there by PERAM at $864D.)
85E3: BCC   $85EA   ;If the first parameter is less than the length
                ;Of the string, then branch
85E5: JSR   $42D8   ;If the first parameter is => the length of the
                ;String, then get the length of the
                ;String from the string descriptor and use it
85E8: TAX                ;Move the length of the string into X register
85E9: TYA                ;Move the Index value into the accumulator

```

```

*****
*
* If the first parameter is less than the length of *
* the string, the accumulator will contain a zero *
* and the X register will contain the first parameter. *
* However, if the 1st parameter is equal to or greater *
* than the length of the string, the accumulator will *
* contain a zero and the X register will contain the *
* the length of the string. *
*
*****

```

```

85EA: PHA                ;Place the index value ($00) onto the stack
85EB: TXA                ;Move the first parameter into the accumulator
85EC: PHA                ;Place the first parameter onto the stack
85ED: JSR   $8690   ;Allocate the room for the 'substring' in RAM
                ;BANK 1 and create the descriptor for the
                ;'Substring' locations $63 - $65 (string length,
                ;LSB/MSB of the address of the string)
85F0: LDA   $52     ;Get the address of the string's descriptor

```

```

85F2: LDY   $53           ;That was placed there by PERAM at $864D
85F4: JSR   $8785        ;Deallocate the temporary string if it is on the
                        ;Temporary string stack, else place the string's
                        ;Address in locations $24, $25
85F7: PLA                       ;Get the first parameter off of the stack
85F8: TAY                       ;And place it into the Y register
85F9: PLA                       ;Get the index value off of the stack
85FA: CLC                       ;And add it to the
85FB: ADC   $24              ;Address of the string in the temporary work
85FD: STA   $24              ;Area to index over to the starting position in
85FF: BCC   $8603           ;The string in which the
8601: INC   $25              ;New 'substring' is to be created from
8603: TYA                       ;Move first parameter back into the accumulator
8604: JSR   $8763          ;Move the 'substring' into the area in RAM BANK
                        ;1 that was allocated by the JSR to $8690 above
                        ;Which placed address into locations $35, $36.
8607: JMP   $86E3          ;Transfer substring's descriptor from locations
                        ;$63 - $65 into currently available temporary
                        ;String descriptor pointed to by location $18.
                        ;Then save the pointer that is in location $18
                        ;To location $66 to point to temporary string
                        ;Descriptor which holds substring's descriptor

```

```

*****
*
*                               BASIC RIGHT$ FUNCTION
*
* Function syntax:  RIGHT$ (string, first para.)
*
* This function returns a string that consists of the
* number of rightmost characters of the 'string' that
* is determined by the 'first para.'.  If the 'first
* para.' is greater than the length of the 'string',
* then the entire string is returned by the RIGHT$
* function.
*
*****

```

```

860A: JSR   $864D          ;Get the address of the string's descriptor and
                        ;The first parameter off of the stack and
                        ;Zero the index register
860D: PHA                       ;Save the first parameter onto the stack
860E: JSR   $42D8          ;Get the length of the string from
8611: STA   $79             ;The string's descriptor and save it
8613: PLA                       ;Get the first parameter off of the stack
8614: CLC                       ;And subtract the length of the string from

```

```

8615: SBC   $79           ;The first parameter. Then add #$FF if the
                        ;First parameter is greater than the
8617: EOR   #$FF         ;Length of the string, clear the carry flag
                        ;If it is greater, and set the carry flag if it
                        ;Is not
8619: JMP   $85E3        ;Jump to an entry point of LEFT$. If the carry
                        ;Is cleared, the accumulator will hold the index
                        ;Value,
                        ;X register will hold the first parameter, and
                        ;Y register will hold a zero ( X and Y registers
                        ;Were set by the PERAM routine). If the carry is
                        ;Set, accumulator is substituted with the length
                        ;Of the string and moved into the X register
                        ;And the Y register will still contain
                        ;The zero placed there by the PREAM routine.

```

```

*****
*
*           BASIC MID$ FUNCTION
*
* Function syntax: MID$(string, 1st para., 2nd para.)
*
* This function returns a substring that consists of
* the characters of the 'string' starting at the
* character specified by '1st para.' and whose length
* is specified by '2nd para.'.
*
*****

```

```

861C: LDA   #$FF         ;Set second parameter to a default value of $FF
861E: STA   $67          ;Which will cause the 'substring' to be created
                        ;From the first parameter (moving right) to the
                        ;End of the string
8620: JSR   $0386        ;Get the char. just after the first parameter
8623: CMP   #' ) '       ;Is it the closing parenthesis?
8625: BEQ   $862D        ;If so, use the default second parameter and
                        ;Branch past the check for a comma and the
                        ;Second parameter
8627: JSR   $795C        ;If it is not the closing parenthesis, it MUST
                        ;Be a comma and if so, skip the comma and move
                        ;TXTPTR
                        ;To the next ASCII character. If the comma is
                        ;Not found, then generate a 'SYNTAX' error
862A: JSR   $87F4        ;Convert the ASCII numeric character to a HEX
                        ;Character value in location $67 and move TXTPTR
                        ;To point to the next char. If this value is
                        ;Negative, generate an 'ILLEGAL QUANTITY' error

```

```

862D: JSR   $864D      ;Get the address of the string's descriptor and
                        ;The first parameter off of the stack and then
                        ;Zero the Y register
8630: BEQ   $8685      ;If the first parameter is a zero, then generate
                        ;An 'ILLEGAL QUANTITY' error
8632: DEX                       ;Subtract one form the first parameter
8633: TXA                       ;And move it into the accumulator
8634: PHA                       ;Then save the first parameter minus one on to
                        ;The stack for use by the LEFT$ routine
8635: LDX   #0           ;Set the length of the 'substring' to 0 in case
                        ;The first parameter is greater than the length
                        ;Of the string. (the 'substring' will equal
                        ;A null string)
8637: PHA                       ;Save the first parameter minus one onto the
                        ;Stack again for recall by this routine later
8638: JSR   $42D8      ;Get the length of the string from the
863B: STA   $79         ;String's descriptor and save it to location $79
863D: PLA                       ;Get first parameter minus one off of the stack
863E: CLC                       ;And subtract the length of the string
863F: SBC   $79         ;From the first parameter and if
8641: BCS   $85EB      ;The first parameter is greater than the length
                        ;Of string, create 'substring' as a null string
8643: EOR   #$FF        ;Add #$FF to the result to create the new second
                        ;Parameter (Note: If the first parameter and the
                        ;Length of the string are the same, a zero is
                        ;Returned here)
8645: CMP   $67         ;Compare the result to the second parameter
8647: BCC   $85EC      ;And if the result is less than, then use that
                        ;Value as the second parameter
8649: LDA   $67         ;Else, use the default value of $FF as
864B: BCS   $85EC      ;The second parameter and jump to the middle of
                        ;The LEFT$ function routine in order
                        ;To create the 'substring'

```

```

*****
*
*
*           PERAM
*
*           PERform Address Manipulations
*
*
* This routine pulls the string's address and the
* first parameter off of the stack for the LEFT$,
* RIGHT$, and MID$ functions. These values were
* placed onto the stack by the routine at $4C16.
*

```

```

* They are placed on the stack in the following order: *
*
* The address of the string *
* The first parameter that follows the string *
* The return address to the BASIC Interpreter *
* The return address of the string function *
*
*****

```

```

864D: JSR    $7956    ;Check for closing parenthesis and if it is not
                    ;Found, then generate a 'SYNTAX' error
8650: PLA                    ;Get return address of string function routine
                    ;Off of the stack
8651: TAY                    ;And place the LSB of address into Y register
8652: PLA                    ;Get the MSB of the address and
8653: STA    $57        ;Save it to location $57 for recall after the
                    ;Execution of this subroutine
8655: PLA                    ;Pull the return address of the JSR to $0056
                    ;Which was
8656: PLA                    ;Performed in routine at $4C16, discard them
8657: PLA                    ;Get the first parameter off of the stack
8658: TAX                    ;And place it into the X register
8659: PLA                    ;Get the address of the string descriptor that
865A: STA    $52        ;Was in parentheses
865C: PLA                    ;And save it
865D: STA    $53        ;Into locations $52, $53
865F: LDA    $57        ;Get the address of string function routine that
8661: PHA                    ;Called this subroutine
8662: TYA                    ;And push it onto the stack so
8663: PHA                    ;That this routine will return there after the
                    ;RTS below is executed
8664: LDY    #0          ;Zero the Y register
8666: TXA                    ;Place the first parameter into the accumulator
8667: RTS                    ;And exit the routine

```

```

*****
*
* BASIC LEN FUNCTION *
*
* Function syntax: LEN (string) *
*
* This function returns a number that represents the *
* length of the string. If the string contains spaces *
* or non-printable characters, these are also *
* included in the LENgth count. *
*
*****

```



```
8668: JSR   $866E      ;Get the length of the string and place that
                    ;Value into the accumulator
866B: JMP   $84D4      ;Change the value to Floating Point format
```

```
*****
*
*           Get/Set the parameters of a string
*
* Set the VALTYP flag at location $0F to zero to
* indicate a numeric value.
*
* The registers represent the values as follows:
*
*           A = The length of the string
*           X = Is equal to zero
*           Y = The length of the string
*
*****
```

```
866E: JSR   $877E      ;Get the length of the string
                    ;And place it in accumulator
8671: LDX   #0         ;Flag for numeric
8673: STX   $0F        ;Set the flag
8675: TAY           ;Move the length of string into the Y register
8676: RTS           ;Exit the routine
```

```
*****
*
*           BASIC ASC FUNCTION
*
* Function syntax:  ASC (string)
*
* This function returns the ASCII code of the first
* character that is contained in the string.
*
*****
```

```
8677: JSR   $866E      ;Get the string's location (pointer)
867A: BEQ   $8682      ;If length is 0, then 'ILLEGAL QUANTITY' error
867C: LDY   #0         ;Zero the index pointer
867E: JSR   $03B7      ;Get the first character from the string
8681: TAY           ;Place the ASCII code in the Y register
8682: JMP   $84D4      ;Change the value to Floating Point format
8685: JMP   $7D28      ;Generate an 'ILLEGAL QUANTITY' error
```

```

*****
*
*          CALCULATE THE STRING VECTOR          *
*
*****

8688: LDX  $66      ;Get the address of the
868A: LDY  $67      ;String and store
868C: STX  $52      ;The address to
868E: STY  $53      ;A temporary work area
8690: JSR  $9299    ;Create room in RAM BANK 1 for the string
                        ;Whose length is in the accumulator. Add 2 to
                        ;The length for the descriptor pointer
8693: STX  $64      ;Save the address of the next available
8695: STY  $65      ;String descriptor in RAM BANK 1
8697: STA  $63      ;Save the length of the string
8699: RTS                    ;Exit

*****
*
*          STRLIT          *
*
* Set up and allocate space in memory for the string. *
*
*****

869A: LDX  #'"'      ;Set the beginning and the
869C: STX  $09      ;Ending delimiter
869E: STX  $0A      ;To a quote
86A0: STA  $70      ;Store the beginning address of the
86A2: STY  $71      ;String into locations $70, $71
86A4: STA  $64      ;And also in
86A6: STY  $65      ;Locations $64, $65
86A8: LDY  #$FF     ;Set the index pointer to $FF so that the next
                        ;Instruction will increment the value to $00
86AA: INY                    ;Add one to the index pointer
86AB: JSR  $42F1    ;Get the first character from the string
86AE: BEQ  $86BC    ;If it is the End of the Line Flag, then branch
86B0: CMP  $09      ;Is it the first delimiter?
86B2: BEQ  $86B8    ;If yes, then branch
86B4: CMP  $0A      ;If it is not the second delimiter, then branch
86B6: BNE  $86AA    ;to continue obtaining the length of the string
86B8: CMP  #'"'     ;If the delimiter is a quote,
86BA: BEQ  $86BD    ;Then branch
86BC: CLC                    ;Clear the carry flag for addition
86BD: STY  $63      ;Save the index value as length of the string
86BF: TYA                    ;Move the length of the string into accumulator

```

```

86C0: ADC    $70      ;And add it to the address (LSB) of the string
86C2: STA    $72      ;And save it as current char. being processed
86C4: LDX    $71      ;Get the MSB of the address of the string
86C6: BCC    $86C9    ;And if there is no overflow from the previous
                        ;Addition, then branch
86C8: INX                    ;If an overflow did occur, add one to the MSB
86C9: STX    $73      ;Save the value back as the MSB of the current
                        ;Character being processed
86CB: TYA                    ;Move the index value back into the accumulator
    
```

```

*****
*
* This routine will move the string into the address *
* stored in locations $70,$71 and will store the length*
* of the string into the accumulator. The routine then *
* moves the string from RAM BANK 0 into RAM BANK 1. *
*
*****
    
```

```

86CC: JSR    $8688    ;Allocate the space for the string in RAM BANK 1
86CF: TAY                    ;Move the length of string into the accumulator
86D0: BEQ    $86E3    ;If length of the string is equal to zero, then
                        ;Branch to move the length and address of the
                        ;String into the temporary string stack
86D2: PHA                    ;Save the length of the string onto the stack
86D3: DEY                    ;Subtract one to use the length of the string
                        ;As a true index value
86D4: JSR    $42F1    ;Get a character from the string in RAM BANK 0
86D7: STA    $FF04    ;Enable BANK 15 with RAM BANK 1 enabled
86DA: STA    ($37),Y  ;Place the character into the area in RAM BANK 1
                        ;That was allocated by the routine at $86CC
86DC: TYA                    ;Move the index pointer back into accumulator
86DD: BNE    $86D3    ;If there are more characters to move, branch
86DF: PLA                    ;Get the true length of string off of the stack
86E0: JSR    $8771    ;Update FRESPEC to indicate last used string area
    
```

```

*****
*
* This routine will move the string descriptor from *
* locations $63 - $65 into the temporary string stack *
* pointed to by TMPPT ($18). If the string stack is *
* full, then location $18 will have a value of $24 *
* in it and a 'FORMULA TOO COMPLEX' error will result. *
*
*****
    
```

```

86E3: LDX  $18      ;Get pointer to the next available string stack
86E5: CPX  #$24      ;Is the stack full?
86E7: BNE  $86EE     ;If not, then branch
86E9: LDX  #25       ;If the string stack is full, then generate
86EB: JMP  $4D3C     ;A 'FORMULA TOO COMPLEX' error
86EE: LDA  $63       ;Get the length of the string
86F0: STA  $00,X     ;And place it into the temporary string stack
86F2: LDA  $64       ;Get LSB of the string's address in RAM BANK 1
86F4: STA  $01,X     ;And place it into the temporary string stack
86F6: LDA  $65       ;Get MSB of the string's address in RAM BANK 1
86F8: STA  $02,X     ;And place it into the temporary string stack
86FA: LDY  #0
86FC: STX  $66       ;Save the address of the string stack
86FE: STY  $67       ;Which was indicated by TEMPPT ($18)
8700: STY  $71       ;Into locations $66, $67, and $71
8702: DEY          ;Subtract one from the zero to obtain the flag
                        ;For a string
8703: STY  $0F       ;Set the flag for 'string'
8705: STX  $19       ;Save the string stack address we just used, as
                        ;The last string stack that was used
8707: INX
8708: INX          ;Add three
                        ;To TEMPPT to
8709: INX          ;Indicate which temporary descriptor
870A: STX  $18       ;Stack is available next
870C: RTS          ;Exit the routine

```

```

*****
*
*
*           CAT
*
* This routine is used when the text of one string is
* to be added to the text of a second string. For
* example: C$ = (A$ + B$).
*
*****

```

```

870D: LDA  $67       ;Save the address of the
870F: PHA          ;First string's variable
8710: LDA  $66       ;Descriptor onto the stack
8712: PHA          ;For recall later
8713: JSR  $78D7     ;Get address of the second string's descriptor
                        ;And place the address into locations $66, $67
8716: JSR  $77DD     ;Insure that the second string is indeed a
                        ;String and not a numeric value. If it is not a
                        ;String, then generate a 'TYPE MISMATCH' error
8719: PLA          ;Obtain the address of the first
871A: STA  $70       ;String's descriptor off of the stack

```

```

871C: PLA                ;And save it
871D: STA    $71         ;Into locations $70, $71
871F: LDY    #0          ;Get the length of the first string from
8721: JSR    $42F6       ;Nominal byte zero of the variable descriptor
8724: STA    $79         ;And save it
8726: JSR    $42E7       ;Get length of the second string from nominal
                        ;Byte zero of variable descriptor. The address
                        ;Of the descriptor is in locations $66, $67
8729: CLC                ;Clear the carry for addition
872A: ADC    $79         ;Add the length of the first string to the
                        ;Length of the second string
872C: BCC    $8731       ;If the length of both strings combined is
                        ;Less than 255, then branch
872E: JMP    $A5ED       ;If the length value is greater than 255, then
                        ;Generate a 'STRING TOO LONG' error
8731: JSR    $8688       ;Allocate a space in the string storage area for
                        ;The new string with the above length result
8734: JSR    $874E       ;Move the first string into the new space
8737: LDA    $52         ;Get the address of the descriptor for
8739: LDY    $53         ;The second string
873B: JSR    $8785       ;And de-allocate the second string
873E: JSR    $8763       ;Move the second string into the new space
8741: LDA    $70         ;Get the address of the first
8743: LDY    $71         ;String's descriptor
8745: JSR    $8785       ;And de-allocate the first string
8748: JSR    $86E3       ;Move the concatenated string descriptor from
                        ;Locations $63-$65 to the temporary string
                        ;Stack. If there is no room left on the stack,
                        ;Then generate a 'FORMULA TOO COMPLEX' error
874B: JMP    $7809       ;Continue evaluating the expression

```

```

*****
*
*                               MOVINS
*
* This routine sets up the address of the string to be
* moved. On entry into this routine, locations $70, $71
* contain the address of the string's descriptor. This
* routine will then fall through to the next routine
* with the length of the string in the accumulator, the
* the LSB of the address of the string in the X regis-
* ter, and the MSB of that address in the Y register.
*
*****

```

```

874E: LDY    #0          ;Index pointer
8750: JSR    $42F6       ;Get the string length

```

```

8753: PHA                ;Save it onto the stack
8754: INY                ;Increment the pointer
8755: JSR    $42F6       ;Get the string address LSB
8758: TAX                ;Save it in the X register
8759: INY                ;Increment the pointer
875A: JSR    $42F6       ;Get the string address MSB
875D: TAY                ;Save it in the Y register
875E: PLA                ;Get the string length back

```

```

*****
*
* This routine will move the string whose address is
* stored in the X and Y registers and whose length is
* in the accumulator to the address pointed to by
* locations $37, $38. This address is usually the
* last allocated string area.
*
*****

```

```

875F: STX    $24         ;Save the string's address LSB
8761: STY    $25         ;Save the string's address MSB
8763: TAY                ;Move the string's length into the Y register
                        ;To be used as an index value
8764: BEQ    $8771       ;If the length is equal to zero, then branch
8766: PHA                ;Save length onto the stack
8767: DEY                ;Subtract one from the length
8768: JSR    $03B7       ;Get a byte of the string that is in RAM BANK 1
876B: STA    ($37),Y     ;And place it into the new address that is also
                        ;In RAM BANK 1
876D: TYA                ;Move the length of the old string
                        ;Back into the accumulator
876E: BNE    $8767       ;If the entire string has not been moved, then
                        ;Branch to continue moving the string
8770: PLA                ;Get length of the original string off of stack
8771: CLC                ;Clear the carry for addition
8772: ADC    $37         ;Add the length of the original string to
8774: STA    $37         ;Update the end of the allocated area
8776: BCC    $877A       ;No overflow, then branch
8778: INC    $38         ;Add one to the MSB due to overflow
877A: RTS                ;Exit the routine

```

```

*****
*
* FRESTR
*
* This routine will call FRETMS ($87E0) to clear a
* temporary string from the temporary string stack.
*
*****

```

```

* If the string is deleted, the Y register will then *
* be set to a value of zero and this routine will *
* set various pointers to reflect the deletion. *
* *
*****

```

```

877B: JSR   $77EF      ;Evaluate the expression
877E: JSR   $77DD      ;Insure the VALTYP is a string and if it is not,
                        ;Then generate a 'TYPE MISMATCH' error
8781: LDA   $66        ;Move the address of the
8783: LDY   $67        ;String descriptor
8785: STA   $24        ;Into a temporary
8787: STY   $25        ;Storage area
8789: JSR   $87E0      ;If the string is on the temporary string stack
                        ;Delete it and if it is deleted, then set the
                        ;Y register to zero
878C: BNE   $87CA      ;If the string descriptor was on the
                        ;String stack, then branch
878E: JSR   $54F6      ;Check if the string is in the proper memory
                        ;Area and clear the carry if it is
8791: BCC   $87CA      ;If the string was 'IN RANGE', then branch to
                        ;Move the index to the second character
                        ;After the string
8793: DEY
8794: LDA   #$FF       ;Value to allow garbage collection to erase
                        ;The string
8796: STA   $FF04      ;Enable BANK 15 with RAM BANK 1
8799: STA   ($24),Y    ;Store the delimiter after the string
879B: DEY             ;Move the index to the first character
                        ;After the string
879C: TXA
879D: STA   ($24),Y    ;String directly after the string
879F: PHA
87A0: EOR   #$FF       ;Invert the length so
87A2: SEC
                        ;The following will subtract
87A3: ADC   $24        ;The length of the de-allocated string
87A5: LDY   $25        ;From the string's address
87A7: BCS   $87AA      ;Handle the underflow by
87A9: DEY             ;Decrementing the MSB of the string's address
87AA: STA   $24        ;And save the address of the string
87AC: STY   $25        ;Back to memory
87AE: TAX
                        ;Save LSB of the string's address to X register
87AF: PLA
                        ;Get the length of the string off of the stack
87B0: CPY   $36
87B2: BNE   $87F0
87B4: CPX   $35
87B6: BNE   $87F0

```

```

87B8: PHA                ;Save length of the string back onto the stack
87B9: SEC                ;Add the length of the
87BA: ADC    $35         ;String to the
87BC: STA    $35         ;String address to move the FRETOP pointer
                        ;To point to the second character after the
                        ;String to de-allocate the string in RAM BANK 1
87BE: BCC    $87C2       ;If no overflow occurred, then branch
87C0: INC    $36         ;If an overflow did occur, then add one to the
                        ;MSB of the FRETOP pointer
87C2: INC    $35         ;Add one to the FRETOP pointer
87C4: BNE    $87C8       ;If no overflow occurred, then branch
87C6: INC    $36         ;Add one to the MSB of the FRETOP pointer
87C8: PLA                ;Get length of the string back off of the stack
87C9: RTS                ;And exit the routine

```

```

*****
*
*   This routine moves the index pointer to point to the *
*   second character after the string. The routine exits *
*   with the X and Y registers holding the address of the*
*   string and the accumulator holding the string length.*
*
*****

```

```

87CA: LDY    #0          ;Zero the index pointer
87CC: JSR    $03B7       ;Get the length of the string
87CF: PHA                ;And save it onto the stack
87D0: INY                ;Move the index pointer to the LSB of
87D1: JSR    $03B7       ;The string's address and
87D4: TAX                ;Save it into the X register
87D5: INY                ;Move the index pointer to the MSB of
87D6: JSR    $03B7       ;The string's address and
87D9: TAY                ;Save it into the Y register
87DA: STX    $24         ;Place the address of the string
87DC: STY    $25         ;Into locations $24, $25
87DE: PLA                ;Get the length of the string off of the stack
87DF: RTS                ;And exit the routine with the X and Y registers
                        ;Holding address of string and the accumulator
                        ;Holding the string length

```

```

*****
*
*
*   FRETMS
*
*   This routine compares the current string with the *
*   strings that are on the string descriptor stack and *
*   if a match is found, that string is removed from the *

```



```

* stack. Upon entry to this routine the following      *
* exist:                                                *
*                                                       *
*           A = The LSB of which descriptor            *
*                is being used                        *
*                                                       *
*           Y = The MSB of which descriptor            *
*                is being used                        *
*                                                       *
*           X = Is not used or affected                *
*                                                       *
*****

```

```

87E0: CPY   $1A           ;If the MSB of the string descriptor's address
87E2: BNE   $87F0         ;Does not equal, then it cannot be in the
                        ;Temporary string stack
87E4: CMP   $19           ;Is it the temporary descriptor used last?
87E6: BNE   $87F0         ;No, then branch to exit
87E8: STA   $18           ;Save the pointer to the LSB of the 3 BYTE
                        ;String descriptor
87EA: SBC   #3            ;Delete the string from the stack
87EC: STA   $19           ;By decrementing the LSB pointer of
                        ;Last string stack address
87EE: LDY   #0            ;Flag for STRING DELETED
87F0: RTS                    ;Exit

```

```

*****
*                                                       *
*           GETBYTC                                         *
*                                                       *
* This routine converts an ASCII character into an      *
* integer value. It also makes sure the value is       *
* between 0 - 255 and if it is not, then the routine   *
* generates an 'ILLEGAL QUANTITY' error.              *
*                                                       *
* On entry to this routine, you MUST have the TXTPNTR  *
* set up to point to the character before the ASCII    *
* character to be converted.                            *
*                                                       *
* Upon exiting this routine, the X register will have  *
* the value of 0 - 255 in it.                          *
*                                                       *
*****

```

```

87F1: JSR   $0380         ;Get the next character

```

```

*****
*
*                               CONVASCN
*
* This routine will decrement TXTPTR and then convert
* the ASCII character it points at to a numeric value.
*
* Upon exiting this routine, the X register will have
* the value of 0 - 255 in it.
*
*****

```

```
87F4: JSR  $77D7      ;Evaluate expression and check for numeric
```

```

*****
*
*                               ASCI HEX
*
* This routine converts an ASCII value to a HEX value.
*
*****

```

```

87F7: JSR  $84AD      ;Turn the ASCII value into integer format
87FA: LDX  $66        ;Get the sign of the value
87FC: BNE  $882B      ;If the sign is negative, then branch to
                        ;Generate an 'ILLEGAL QUANTITY' error
87FE: LDX  $67        ;Return the numeric value into the X register
8800: JMP  $0386      ;Update TXTPTR

```

```

*****
*
*                               GETADR
*
* This routine is used to get address parameters by
* first checking to see if the value in the Floating
* Point accumulator is a positive number that is less
* than the value 65536. If the value is indeed less
* than 65536, then this routine calls the routine that
* converts the value from Floating Point format into a
* two byte unsigned integer. This routine is used by
* such routines as the PEEK and POKE routines.
*
*****

```

```

8803: JSR  $77D7      ;Evaluate the expression
8806: JSR  $8815      ;Get the 16 bit address into locations $16, $17
8809: JSR  $795C      ;Search for ', '. If it is not found, then

```



```

*****
*
*                               FSUB1
*
*                               FAC2 = MEMORY - FAC1
*
* This routine subtracts FAC1 from the number stored
* in RAM BANK 1 whose address is in the Y register and
* the accumulator (LSB,MSB). This is accomplished by
* moving the number stored in memory into FAC2 and then
* falling through to the routine below (FSUBT).
*
*****

```

```

882E: JSR  $8AB4      ;Move the Floating Point number stored in RAM
                    ;BANK 1 into FAC2

```

```

*****
*
*                               FSUBT
*
*                               BASIC MINUS (-) FUNCTION
*
* Function syntax: first exp. - second exp.
*
* This routine performs subtraction by changing the
* sign of the FAC1 and adding the value to FAC2.
*
*****

```

```

8831: LDA  $68        ;Invert the sign of FAC1
8833: EOR  #$FF       ;And save it back
8835: STA  $68        ;Into the FAC1 sign flag
8837: EOR  $6F       ;Compare the sign of FAC1 to FAC2
                    ;($00 = the same, $FF = different)
8839: STA  $70        ;Save the result
883B: LDA  $63        ;Get the exponent of FAC1
883D: JMP  $8848     ;And add FAC1 to FAC2
8840: JSR  $8979     ;Shift result right
8843: BCC  $8882     ;Check if a borrow was done and if so, result
                    ;sign (-)

```

```

*****
*
*                               FADD
*
*                               FAC1 = FAC1 + FAC2
*
*****

```

```

* This routine adds FAC1 to a number stored in memory *
* whose address is in the Y register and the *
* accumulator (LSB,MSB) by moving the number that is *
* stored in memory into FAC2 and then falling through *
* to the routine below (FADDT). *
* *
*****

```

```

8845: JSR $8AB4 ;Add floating point value, that is pointed to
;By the accumulator and the Y register to FAC1

```

```

*****
* *
* FADDT *
* *
* BASIC PLUS (+) FUNCTION *
* *
* Function syntax: first exp. + second exp. *
* *
* This routine adds FAC1 to FAC2 and then stores the *
* result into FAC1. *
* *
*****

```

```

8848: BNE $884D ;Move FAC2 (ARG) to
884A: JMP $8C28 ;FAC1. If FAC1 = zero,
884D: LDX $71 ;Save the
884F: STX $58 ;Rounding flag
8851: LDX #$6A ;Set up an index to FAC2 (ARG)
8853: LDA $6A ;Get the exponent of FAC2
8855: TAY ;Save it in the Y register
8856: BNE $8859 ;Exit if the value in FAC2
8858: RTS ;Is zero
8859: SEC ;Subtract the exponent of FAC1 from the
885A: SBC $63 ;Exponent of FAC2 to calculate the difference
885C: BEQ $8882 ;If there is no difference, then branch to act
;On fractions
885E: BCC $8872 ;Branch if FAC1 is larger than FAC2
8860: STY $63 ;Save the exponent of FAC2 into FAC1. If FAC2 is
;Greater than FAC1,
8862: LDY $6F ;Save the sign byte of FAC2 as
8864: STY $68 ;The sign byte of FAC1
8866: EOR #$FF ;Compute the two's
8868: ADC #0 ;Complement of the difference
886A: LDY #0 ;Between FAC1 and FAC2
886C: STY $58 ;Clear temporary storage area for rounding flag
886E: LDX #$63 ;Set up an index to FAC1

```

```

8870: BNE    $8876    ;Branch to skip the clearing of rounding flag
8872: LDY    #0       ;Clear the
8874: STY    $71     ;Rounding flag
8876: CMP    #$F9     ;If the difference between FAC1 and FAC2 is
                        ;Greater than 8,
8878: BMI    $8840    ;Then branch to do the preshifts
887A: TAY                    ;Save the number of preshifts in the Y register
887B: LDA    $71     ;Get the rounding flag
887D: LSR    $1,X    ;
887F: JSR    $8990    ;Do the preshifts
8882: BIT    $70     ;If the signs of FAC1 and FAC2 are the same,
8884: BPL    $88DD    ;Then add the fractions

```

```

*****
*
*                               FADD4
*
* This routine makes the result negative if a borrow
* was done.
*
*****

```

```

8886: LDY    #$63     ;Make the Y register to point to
8888: CPX    #$6A     ;The larger, FAC1 or FAC2
888A: BEQ    $888E
888C: LDY    #$6A
888E: SEC
888F: EOR    #$FF     ;Compute the borrow
8891: ADC    $58     ;From the rounding flag
8893: STA    $71     ;And save it

```

```

*****
*
*      FAC1 = FAC1 - FAC2    or    FAC1 = FAC2 - FAC1
*
* Upon entry into this routine the X register points
* to the Floating Point accumulator that holds the
* smaller value and the Y register points to the
* Floating Point accumulator that holds the larger
* value. This determines whether FAC2 will be
* subtracted from FAC1 or vice versa.
*
*****

```

```

8895: LDA    $4,Y    ;Compute the difference between
8898: SBC    $4,X    ;The two Floating Point accumulators
889A: STA    $67    ;M4

```

```

889C: LDA    $3,Y
889F: SBC    $3,X
88A1: STA    $66      ;M3
88A3: LDA    $2,Y
88A6: SBC    $2,X
88A8: STA    $65      ;M2
88AA: LDA    $1,Y
88AD: SBC    $1,X
88AF: STA    $64      ;M1
88B1: BCS    $88B6    ;Branch if there is no borrow
88B3: JSR    $8926    ;Negate FAC1 if there was a borrow
88B6: LDY    #0       ;Clear the Y register
88B8: TYA                    ;And the accumulator
88B9: CLC
88BA: LDX    $64      ;If M1 is equal to zero, then
88BC: BNE    $8908    ;Shift the fraction over
88BE: LDX    $65      ;One byte
88C0: STX    $64
88C2: LDX    $66
88C4: STX    $65
88C6: LDX    $67
88C8: STX    $66
88CA: LDX    $71
88CC: STX    $67
88CE: STY    $71
88D0: ADC    #8       ;Update the exponent by 8
88D2: CMP    #32      ;And continue until
88D4: BNE    $88BA    ;4 bytes are shifted
88D6: LDA    #0       ;Make FAC1 equal to zero
88D8: STA    $63
88DA: STA    $68      ;Clear the sign byte of FAC1
88DC: RTS            ;And exit the routine

*****
*
*          FAC1 = FAC1 + FAC2
*
* This routine adds FAC1 to FAC2 and places the result
* into FAC1.
*
*****

88DD: ADC    $58      ;Compute the initial fraction
88DF: STA    $71      ;And save as the rounding flag
88E1: LDA    $67      ;Add M4 of FAC1
88E3: ADC    $6E      ;To M4 of FAC2
88E5: STA    $67      ;Save in FAC1

```

```

88E7: LDA $66 ;Add M3 of FAC1
88E9: ADC $6D ;To M3 of FAC2
88EB: STA $66 ;Save in FAC1
88ED: LDA $65 ;Add M2 of FAC1
88EF: ADC $6C ;To M2 of FAC2
88F1: STA $65 ;Save in FAC1
88F3: LDA $64 ;Add M1 of FAC1
88F5: ADC $6B ;To M1 of FAC2
88F7: STA $64 ;Save in FAC1
88F9: JMP $8915 ;Normalize FAC1, if needed, and exit

```

```

*****
*
*
*
*
*
*
*
*
*
*
*
*****

```

NORMAL

Normalize the Floating Point number.

```

88FC: ADC #1 ;Increase the exponent by 1
88FE: ASL $71 ;Shift the fraction left
8900: ROL $67 ;One bit
8902: ROL $66
8904: ROL $65
8906: ROL $64
8908: BPL $88FC ;Continue until an overflow from M1 occurs
890A: SEC ;Subtract the correction
890B: SBC $63 ;From the exponent
890D: BCS $88D6 ;If there was an underflow, then make FAC1
;Equal to zero and exit
890F: EOR #$FF ;Invert the value
8911: ADC #1 ;Increase the value by 1
8913: STA $63 ;And save it as the new exponent
8915: BCC $8925 ;If no overflow occurred, then exit
8917: INC $63 ;If there was overflow, increment exponent by 1
8919: BEQ $895D ;If the exponent overflows, then generate an
; 'OVERFLOW' error
891B: ROR $64 ;If there was not an overflow, then after
; Incrementing the exponent,
891D: ROR $65 ;Shift the fraction right 1 bit
891F: ROR $66
8921: ROR $67
8923: ROR $71
8925: RTS ;And exit

```



```

*****
*
*                               NEGFAC
*
* This routine takes the value in FAC1, generates the
* two's complement of that value, and places the value
* back into FAC1.
*
*****

```

```

8926: LDA  $68      ;Generate the complement of
8928: EOR  #$FF     ;The sign of FAC1
892A: STA  $68      ;And save it

```

```

*****
*
*                               Invert the mantissa of FAC1
*
*****

```

```

892C: LDA  $64      ;Invert M1
892E: EOR  #$FF
8930: STA  $64
8932: LDA  $65      ;Invert M2
8934: EOR  #$FF
8936: STA  $65
8938: LDA  $66      ;Invert M3
893A: EOR  #$FF
893C: STA  $66
893E: LDA  $67      ;Invert M4
8940: EOR  #$FF
8942: STA  $67
8944: LDA  $71      ;Invert the rounding byte
8946: EOR  #$FF
8948: STA  $71
894A: INC  $71      ;Add 1 to the rounding byte
894C: BNE  $895C    ;And exit if mantissas do not need rounding
894E: INC  $67      ;Add 1 to M4
8950: BNE  $895C    ;Exit if there is no carry
8952: INC  $66      ;Add 1 to M3
8954: BNE  $895C    ;Exit if there is no carry
8956: INC  $65      ;Add 1 to M2
8958: BNE  $895C    ;Exit if there is no carry
895A: INC  $64      ;Add 1 to M1
895C: RTS          ;Exit the routine
895D: LDX  #15      ;Generate an
895F: JMP  $4D3C    ;'OVERFLOW' error

```

```

*****
*
*                               MULTSHF
*
*   This routine shifts the result of a multiplication
*   to the right.
*
*****

8962: LDX  #$27           ;Index to the address of the result
8964: LDY  $4,X          ;Shift the fraction
8966: STY  $71           ;Right one byte
8968: LDY  $3,X
896A: STY  $4,X
896C: LDY  $2,X
896E: STY  $3,X
8970: LDY  $1,X
8972: STY  $2,X
8974: LDY  $03DF        ;Overflow marker for FAC1
8977: STY  $1,X
8979: ADC  #8           ;Increment the exponent by 8
897B: BMI  $8964        ;And continue
897D: BEQ  $8964
897F: SBC  #8           ;Create a shift counter
8981: TAY
8982: LDA  $71          ;Check the rounding flag
8984: BCS  $899A        ;If the exponent is zero, then exit
8986: ASL  $1,X         ;Otherwise shift
8988: BCC  $898C        ;The fraction right one bit
898A: INC  $1,X
898C: ROR  $1,X
898E: ROR  $1,X
8990: ROR  $2,X
8992: ROR  $3,X
8994: ROR  $4,X
8996: ROR
8997: INY
8998: BNE  $8986        ;Put the last bit into the accumulator
8999: CLC
899B: RTS
8998: BNE  $8986        ;Continue until the shift
899A: CLC
899B: RTS
8998: BNE  $8986        ;Counter equals zero
899A: CLC
899B: RTS
8998: BNE  $8986        ;And exit.

*****
*
*                               FONE
*
*                               Constant For ONE (1)
*
*   The five bytes from $899C to $89A0 represent the
*
*****

```

```
* value of one in Floating Point format. This value is*
* used by the Floating Point routines and also as a de-*
* fault value for the STEP value for the FOR statement.*
*
*****
```

```
EX M1 M2 M3 M4
-- -- -- -- --
```

```
899C: .BYTE $81,$00,$00,$00,$00 ;1
```

```
*****
*
* LOGCN2
*
* This table of Floating Point numbers are the
* constants that are used by the LOG function.
*
*****
```

```
EX M1 M2 M3 M4
-- -- -- -- --
```

```
89A1: .BYTE $03 ;3 = Polynomial degree then 4
89A2: .BYTE $7F,$5E,$56,$CB,$79 ; .434255942 / coefficients
89A7: .BYTE $80,$13,$9B,$0B,$64 ; .576584541
89AC: .BYTE $80,$76,$38,$93,$16 ; .961800759
89B1: .BYTE $82,$38,$AA,$3B,$20 ;2.88539007
89B6: .BYTE $80,$35,$04,$F3,$31 ; .707106781 = 1/SQR(2)
89BB: .BYTE $81,$35,$04,$F3,$34 ;1.41421356 = SQR(2)
89C0: .BYTE $80,$80,$00,$00,$00 ;-0.5
89C5: .BYTE $80,$31,$72,$17,$F8 ; .693147181 = LOG(2)
```

```
*****
*
* BASIC LOG FUNCTION
*
* Function syntax: LOG (X)
*
* This function returns the natural log of the number
* specified by 'X'.
*
* EXAMPLE: PRINT LOG (2)
* .693147181
*
*****
```

```
89CA: JSR $8C57 ;Get the sign of FAC1 into the accumulator
89CD: BEQ $89D1 ;If FAC1 equals 0, then 'ILLEGAL QUANTITY' error
```

```

89CF: BPL $89D4 ;If FAC1 equals one, then the value is positive
;So branch
89D1: JMP $7D28 ;Generate an 'ILLEGAL QUANTITY' error
89D4: LDA $63 ;Get the exponent of FAC1
89D6: SBC #$7F ;Keep the sign of the exponent on the stack
89D8: PHA ;Save the bit 7 value onto the stack
89D9: LDA #$80 ;Make the FAC1 exponent sign positive
89DB: STA $63 ;Save it
89DD: LDA #$B6 ;Pointer to
89DF: LDY #$89 ;CONSTANT 1/SQR(2)
89E1: JSR $8A12 ;FAC1 = FAC1 + CONSTANT 1/SQR(2)
89E4: LDA #$BB ;Pointer to constant SQR(2)
89E6: LDY #$89
89E8: JSR $8A1E ;FAC1 = SQR(2)/FAC1
89EB: LDA #$9C ;Pointer to constant 1
89ED: LDY #$89
89EF: JSR $8A18 ;FAC2 = 1 - FAC1
89F2: LDA #$A1 ;Pointer to the polynomial
89F4: LDY #$89 ;Coefficients
89F6: JSR $9086 ;Calculate the polynomial
89F9: LDA #$C0 ;Pointer to
89FB: LDY #$89 ;Constant -0.5
89FD: JSR $8A12 ;FAC1 = (-.5 + FAC1)
8A00: PLA ;Get bit 7 of the exponent back
8A01: JSR $8DB0 ;FAC1 = FAC1 + BIT 7 of the FAC1 mantissa
8A04: LDA #$C5 ;Pointer to
8A06: LDY #$89 ;Constant LOG(2)

```

```

*****
*
*                               FMULT
*
*                               FAC2 = FAC1 * MEMORY
*
* This routine moves the value in MEMORY into FAC2,
* multiplies FAC1 by FAC2, then places the result
* into FAC2.
*
*****

```

```

8A08: JSR $8A89 ;FAC2 = CONSTANT
8A0B: JMP $8A27 ;BASIC MULTIPLY

```

```

*****
*
*          FAC1 = - 0.5
*
*****

8A0E: LDA   #$76       ;Pointer to
8A10: LDY   #$8F       ;Constant 0.5 (1/2)
8A12: JSR   $8A89     ;FAC2 = 0.5
8A15: JMP   $8848     ;Add FAC1 to FAC2

*****
*
*          FSUB0
*
*          FAC2 = MEMORY - FAC1
*
* This routine subtracts FAC1 from the number stored
* in RAM BANK 0 whose address is in the Y register
* and the accumulator (LSB,MSB). This is accomplished
* by moving the number that is stored in memory into
* FAC2, then subtracting FAC1 from FAC2, and then
* placing the result into FAC2.
*
*****

8A18: JSR   $8A89     ;Move the Floating Point number stored in RAM
                        ;BANK 0 into FAC2
8A1B: JMP   $8831     ;BASIC SUBTRACTION FUNCTION

*****
*
*          FDIV
*
*          FAC2 = MEMORY / FAC1
*
* This routine divides a number that is stored in
* memory by FAC1. This is accomplished by moving the
* number that is stored in memory into FAC2,
* then performing the division, and finally placing
* the result into FAC2.
*
*****

8A1E: JSR   $8A89     ;Move constant whose address is pointed to by
                        ;The accumulator and Y register into FAC2 (ARG)
8A21: JMP   $8B4C     ;Perform the division

```

```
8A24: JSR   $8AB4       ;Move the value to FAC2
```

```
*****
*
*                               FMULT                               *
*
*                               BASIC MULTIPLY (*) FUNCTION        *
*
* Function syntax:  first exp. * second exp.                      *
*
* This function returns the result of multiplying                  *
* the 'first expression' by the 'second expression'.              *
*
*****
```

```
8A27: BNE   $8A2C       ;If not equal to zero, then branch
8A29: JMP   $8A88       ;RTS
8A2C: JSR   $8AEC       ;Add the exponents of FAC1 and FAC2
8A2F: LDA   #0          ;Zero the temporary storage area
8A31: STA   $28
8A33: STA   $29
8A35: STA   $2A
8A37: STA   $2B
8A39: LDA   $71         ;Get the overflow flag and multiply it
8A3B: JSR   $8A55       ;By FAC plus the value in the storage area
8A3E: LDA   $67         ;Do the same for M4
8A40: JSR   $8A55
8A43: LDA   $66         ;And M3
8A45: JSR   $8A55
8A48: LDA   $65         ;And M2
8A4A: JSR   $8A55
8A4D: LDA   $64         ;And M1
8A4F: JSR   $8A5B
8A52: JMP   $8BC1       ;Move the result into FAC1 and exit
```

```
*****
*
*                               MLTPLY                               *
*
* This subroutine adds a mantissa byte of FAC2 to FAC1 *
* the number of times indicated by the accumulator.          *
*
*****
```

```
8A55: BNE   $8A5B       ;If there is a number to multiply, then branch
8A57: SEC
8A58: JMP   $8962       ;Shift the value right
```



```

8A93: DEY                ;Y = Y - 1
8A94: LDA ($24),Y       ;Get M3 from memory
8A96: STA $6D           ;Save it into FAC2
8A98: DEY
8A99: LDA ($24),Y       ;Get M2 from memory
8A9B: STA $6C           ;Save it into FAC2
8A9D: DEY
8A9E: LDA ($24),Y       ;Get M1 from memory
8AA0: STA $6F           ;Save as the sign of FAC2
8AA2: EOR $68           ;Compare the sign of FAC2 into FAC1
8AA4: STA $70           ;And save the result
8AA6: LDA $6F           ;Set bit 7 of M1
8AA8: ORA #$80          ;And store it back into FAC2
8AAA: STA $6B
8AAC: DEY
8AAD: LDA ($24),Y       ;Get the exponent of the value in memory
8AAF: STA $6A           ;Save it into FAC2
8AB1: LDA $63           ;Get the exponenet of FAC1 into the accumulator
8AB3: RTS                ;And exit

```

```

*****
*
* This routine reads a Floating Point number from RAM
* BANK 1 into FAC2. Upon entry to this routine the
* accumulator and the Y register hold the LSB and MSB
* of the address that points to the number to be moved.
*
*****

```

```

8AB4: STA $24           ;Save the Address of where in RAM BANK 1 the
8AB6: STY $25           ;Floating point value is stored
8AB8: LDA $FF00         ;Save the current memory configuration
8ABB: PHA              ;Onto the stack
8ABC: LDY #4           ;Index for 5 bytes to read
8ABE: JSR $03B7         ;Get M4 of the value in memory
8AC1: STA $6E           ;Save it into FAC2
8AC3: DEY
8AC4: JSR $03B7         ;Get M3 of the value in memory
8AC7: STA $6D           ;Save it into FAC2
8AC9: DEY
8ACA: JSR $03B7         ;Get M2 of the value in memory
8ACD: STA $6C           ;Save it into FAC2
8ACF: DEY
8AD0: JSR $03B7         ;Get M1 of the value in memory
8AD3: STA $6F           ;Save it into FAC2
8AD5: EOR $68           ;Compare the sign of FAC2 to FAC1
8AD7: STA $70           ;And save the result

```





```

8B09: LDA   $68           ;Get the sign of FAC1
8B0B: EOR   #$FF         ;Reverse it (+ = -) or (- = +)
8B0D: BMI   $8B14        ;If it is '-', then 'OVERFLOW' error
8B0F: PLA                   ;Pull the return address
8B10: PLA                   ;Off the stack
8B11: JMP   $88D6        ;Make the sign and exponent of FAC1 = to zero
8B14: JMP   $895D        ;Generate an 'OVERFLOW' error

```

```

*****
*
*                               MULT10
*
*                               FAC1 = FAC1 * 10
*
* This routine multiplies FAC1 by 10 to aid in
* converting a Floating Point number to a series of
* ASCII numerals.
*
*****

```

```

8B17: JSR   $8C38        ;Round off and place the Floating Point number
                               ;From FAC1 to FAC2
8B1A: TAX                   ;Save the sign of the exponent into X register
8B1B: BEQ   $8B2D        ;If FAC1 equals zero, then exit
8B1D: CLC                   ;Clear the carry flag for addition
8B1E: ADC   #2            ;Exponent = Exponent +2 (EXPONENT = EXPONENT * 2)
8B20: BCS   $8B14        ;'OVERFLOW' error
8B22: LDX   #0            ;Make the result
8B24: STX   $70           ;Of the sign zero
8B26: JSR   $8855
8B29: INC   $63           ;Increase the exponent by one
8B2B: BEQ   $8B14        ;'OVERFLOW' error
8B2D: RTS                   ;Exit

```

```

*****
*
*                               Constant for Floating Point
*
*****

```

```

          EX  M1  M2  M3  M4
          --  --  --  --  --
8B2E: .BYTE $84,$20,$00,$00,$00 ; 10
8B33: LDX   #20           ;'DIVISION BY ZERO' error
8B35: JMP   $4D3C        ;Jump to the error message routine

```

```
*****
*
*                               DIV10
*
*                               FAC1 = FAC1 / 10
*
*   This routine the value in FAC1 by 10.
*
*****
```

```
8B38: JSR  $8C38      ;Move FAC1 to FAC2 (ARG)
8B3B: LDA  #$2E      ;Pointer to the Floating Point
8B3D: LDY  #$8B      ;Constant of 10
8B3F: LDX  #0        ;Clear the sign comparison flag
8B41: STX  $70
8B43: JSR  $8BD4      ;Move the constant of 10 to FAC1
8B46: JMP  $8B4C      ;And divide FAC1 by 10
8B49: JSR  $8AB4      ;Get Floating Point value from the address in
                    ;Pointed to by the accumulator and Y register
                    ;And it into FAC2 (ARG) (BANK 1)
```

```
*****
*
*                               BASIC DIVISION (/) FUNCTION
*
*   Function syntax:  first exp. / second exp.
*
*   This function will return the result of the 'first
*   expression' divided by the 'second expression'.
*
*****
```

```
8B4C: BEQ  $8B33      ;If FAC1 equals zero, then
                    ;Generate a 'DIVISION BY ZERO' error
8B4E: JSR  $8C47      ;Round off FAC1
8B51: LDA  #0        ;Calculate the
8B53: SEC                      ;Two's complement of the exponent of FAC1
8B54: SBC  $63
8B56: STA  $63
8B58: JSR  $8AEC      ;Add the exponents of FAC1 and FAC2 together
8B5B: INC  $63        ;Increment the exponent by one
8B5D: BEQ  $8B14      ;If there was an overflow, then
                    ;Generate an 'OVERFLOW' error
8B5F: LDX  #$FC      ;Index to the storage area for FAC
8B61: LDA  #1        ;Set up for eight shifts
8B63: LDY  $6B      ;Compare M1 of FAC2
8B65: CPY  $64      ;To M1 of FAC1
```

---

```

8B67: BNE $8B79 ;Branch if they are not the same
8B69: LDY $6C ;Compare M2 of FAC2
8B6B: CPY $65 ;To M2 of FAC1
8B6D: BNE $8B79 ;Branch if they are not the same
8B6F: LDY $6D ;Compare M3 of FAC2
8B71: CPY $66 ;To M3 of FAC1
8B73: BNE $8B79 ;Branch if they are not the same
8B75: LDY $6E ;Compare M4 of FAC2
8B77: CPY $67 ;To M4 of FAC1
8B79: PHP ;Save the result of the comparison
8B7A: ROL ;See if we have done eight shifts yet
8B7B: BCC $8B86 ;And Branch if not
8B7D: INX ;Increment to the next byte in storage
8B7E: STA $2B,X ;Save the result
8B80: BEQ $8BB4 ;If the byte just saved was M4, then branch to
;Do the overflow byte

8B82: BPL $8BB8 ;If the overflow byte is done, then exit
8B84: LDA #1 ;Set up for eight shifts
8B86: PLP ;Restore the result of the comparison
8B87: BCS $8B97 ;If the dividend is larger, branch to subtract
8B89: ASL $6E ;Shift the dividend
8B8B: ROL $6D ;Left one bit
8B8D: ROL $6C
8B8F: ROL $6B
8B91: BCS $8B79 ;Branch with comparison equal to one
8B93: BMI $8B63 ;Branch to compare FAC2 to FAC1
8B95: BPL $8B79 ;Branch with comparison equal to zero
8B97: TAY ;If the dividend is larger, then
8B98: LDA $6E ;Subtract M4 of FAC1
8B9A: SBC $67 ;From M4 of FAC2
8B9C: STA $6E
8B9E: LDA $6D ;Subtract M3 of FAC1
8BA0: SBC $66 ;From M3 of FAC2
8BA2: STA $6D
8BA4: LDA $6C ;Subtract M2 of FAC1
8BA6: SBC $65 ;From M2 of FAC2
8BA8: STA $6C
8BAA: LDA $6B ;Subtract M1 of FAC1
8BAC: SBC $64 ;From M1 of FAC2
8BAE: STA $6B
8BB0: TYA
8BB1: JMP $8B89 ;Repeat until complete
8BB4: LDA #$40 ;Perform two more iterations
8BB6: BNE $8B86 ;For the overflow byte
8BB8: ASL ;Shift the result of
8BB9: ASL ;The overflow byte
8BBA: ASL ;6 bytes to the left

```

```

8BBB: ASL
8BBC: ASL
8BBD: ASL
8BBE: STA    $71      ;Save the result
8BC0: PLP                ;Get back the result of the last comparison

```

```

*****
*
*           Move the result into FAC1
*
*****

```

```

8BC1: LDA    $28      ;Transfer
8BC3: STA    $64      ;The resulting floating point
8BC5: LDA    $29      ;Value from $28 - $2B
8BC7: STA    $65      ;To FAC1
8BC9: LDA    $2A
8BCB: STA    $66
8BCD: LDA    $2B
8BCF: STA    $67
8BD1: JMP    $88B6    ;Normalize FAC1 and exit

```

```

*****
*
* This routine transfers the constant whose address is *
* stored in the accumulator and the Y register to FAC1.*
*
*****

```

```

8BD4: STA    $24      ;Save the LSB
8BD6: STY    $25      ;And the MSB of the address of the Floating
                        ;Point value
8BD8: LDY    #4
8BDA: LDA    ($24),Y  ;Read M4 from memory
8BDC: STA    $67      ;Save it into FAC1
8BDE: DEY
8BDF: LDA    ($24),Y  ;Read M3 from memory
8BE1: STA    $66      ;Save it into FAC1
8BE3: DEY
8BE4: LDA    ($24),Y  ;Read M2 from memory
8BE6: STA    $65      ;Save it into FAC1
8BE8: DEY
8BE9: LDA    ($24),Y  ;Read M1 from memory
8BEB: STA    $68      ;Save as the sign of FAC1
8BED: ORA    #$80     ;Set bit 7 of M1
8BEF: STA    $64      ;And save it into FAC1
8BF1: DEY

```

```

8BF2: LDA    ($24),Y    ;Read the exponent from memory
8BF4: STA    $63        ;Save it into FAC1
8BF6: STY    $71        ;Clear the rounding flag
8BF8: RTS                ;Exit

```

```

*****
*
*   This routine moves FAC1 to either temporary buffer
*   number three or four.
*
*****

```

```

8BF9: LDX    #$5E        ;$05E, Temporary buffer #3
8BFB: .BYTE  $2C        ;MASK
8BFC: LDX    #$59        ;$059, Temporary buffer #4
8BFE: LDY    #0         ;High byte of the address of the buffer
8C00: JSR    $8C47      ;Round off FAC1
8C03: STX    $24        ;Store the target
8C05: STY    $25        ;Address in locations $24, $25
8C07: LDY    #4         ;Number of bytes minus one to move
8C09: LDA    $67        ;Transfer M4
8C0B: STA    ($24),Y    ;To target address
8C0D: DEY
8C0E: LDA    $66        ;Transfer M3
8C10: STA    ($24),Y    ;To target address
8C12: DEY
8C13: LDA    $65        ;Transfer M2
8C15: STA    ($24),Y    ;To target address
8C17: DEY
8C18: LDA    $68        ;Compute the sign bit
8C1A: ORA    #$7F       ;Of M4
8C1C: AND    $64        ;And store at
8C1E: STA    ($24),Y    ;The target address
8C20: DEY
8C21: LDA    $63        ;Transfer exponent
8C23: STA    ($24),Y    ;To target address
8C25: STY    $71
8C27: RTS                ;And exit.

```

```

*****
*
*           MOVE FAC2 to FAC1
*
*****

```

```

8C28: LDA    $6F        ;Transfer the sign of ARG (FAC2)
8C2A: STA    $68        ;To FAC1

```

```

8C2C: LDX #5 ;Transfer the 5 byte floating point value
8C2E: LDA $69,X ;From ARG (FAC2)
8C30: STA $62,X ;To FAC1
8C32: DEX
8C33: BNE $8C2E ;Clear the sign comparison
8C35: STX $71 ;Flag
8C37: RTS ;Exit
    
```

```

*****
*
*             MOVE FAC1 to FAC2
*
*****
    
```

```

8C38: JSR $8C47 ;Round off FAC1
8C3B: LDX #6 ;Transfer 5 bytes
8C3D: LDA $62,X ;From FAC1
8C3F: STA $69,X ;To ARG (FAC2)
8C41: DEX
8C42: BNE $8C3D ;Clear the sign comparison
8C44: STX $71 ;Flag
8C46: RTS ;Exit
    
```

```

*****
*
*             ROUND OFF FAC1
*
*****
    
```

```

8C47: LDA $63 ;If the value in FAC1
8C49: BEQ $8C46 ;Is zero, then exit
8 4B: ASL $71 ;If the flag is not set
8C4D: BCC $8C46 ;For rounding, then exit
8C4F: JSR $894E ;Round off FAC1
8C52: BNE $8C46 ;If the value does not read normalizing, exit
8C54: JMP $8917 ;Normalize FAC1 and exit
    
```

```

*****
*
*             GET THE SIGN OF FAC1
*
* This routine gets the sign of FAC1 and then sets
* the accumulator equal to the following values:
*
*             $00 - if FAC1 equals zero
*
*             $01 - if FAC1 is positive
*
*****
    
```

```

*           $FF - if FAC1 is negative           *
*                                                                 *
*****
8C57: LDA   $63           ;If the value in FAC1 is zero,
8C59: BEQ   $8C64        ;Then exit
8C5B: LDA   $68           ;If the value in FAC1
8C5D: ROL                   ;Is negative,
8C5E: LDA   #$FF         ;Then exit with a value of $FF in accumulator
8C60: BCS   $8C64        ;If the value in FAC1
8C62: LDA   #1           ;Is positive, then exit with
8C64: RTS                   ;The accumulator containing the value of $01

```

```

*****
*                                                                 *
*           BASIC SGN FUNCTION           *
*                                                                 *
* Function syntax:  SGN (X)           *
*                                                                 *
* This function returns a value which represents the *
* sign of the number specified by 'X'. The value *
* returned will be '+1' for values of 'X' greater than *
* zero, '0' if the value of 'X' is zero, and '-1' if *
* the value of 'X' is less than zero. *
*                                                                 *
*****

```

```

8C65: JSR   $8C57        ;Get the sign of FAC1 ($01 = Positive, $FF =
                        ;negative)
8C68: STA   $64          ;Save the sign of FAC1
8C6A: LDA   #$00        ;Clear M2
8C6C: STA   $65          ;Of FAC1
8C6E: LDX   #$88        ;For exponent of 7 + 129 = 136 ($88)
8C70: LDA   $64          ;Invert the sign of FAC1
8C72: EOR   #$FF        ;And shift the result
8C74: ROL                   ;Into the carry bit
8C75: LDA   #0          ;Clear the remainder of FAC1
8C77: STA   $67          ;Clear mantissa
8C79: STA   $66          ;LSB high/low bytes
8C7B: STX   $63          ;Save the exponent of 7
8C7D: STA   $71          ;Clear the rounding byte
8C7F: STA   $68          ;And the sign byte of FAC1
8C81: JMP   $88B1       ;Convert the SGN value to Floating Point
                        ;Format and exit
                        ;(1 = positive, 0 = value zero, -1 = negative)

```



```

*****
*
*          BASIC ABS FUNCTION
*
* Function syntax:  ABS (X)
*
* This function returns the absolute (positive value)
* of the number or expression specified for 'X'.
*
*****

```

```

8C84: LSR   $68           ;Make the sign bit positive for FAC1
8C86: RTS                   ;Exit routine

```

```

*****
*
*          FCOMP
*
*          COMPare memory with FAC1
*
* This routine compares the Floating Point value in
* memory whose address is contained in the accumulator
* and Y register to the Floating Point value in FAC1.
* Upon return from this routine, the accumulator will
* hold a zero if the two values were identical and a
* non-zero value if there was no match.
*
*****

```

```

8C87: STA   $26           ;Save the LSB and
8C89: STY   $27           ;The MSB of the address in memory of
                               ;The Floating Point value
8C8B: LDY   #0            ;Zero the index pointer
8C8D: LDA   ($26),Y       ;Get the exponent of the Floating Point value
8C8F: INY                   ;Increment the index pointer to MANTISSA 1 (M1)
8C90: TAX                   ;Place the exponent in the X register
8C91: BEQ   $8C57         ;If the Floating Point value in memory
                               ;Is equal to zero, then exit
8C93: LDA   ($26),Y       ;Get M1 of the value in memory
8C95: EOR   $68           ;Compare the signs of FAC1 and MEMORY
8C97: BMI   $8C5B         ;If they are different, then exit
8C99: CPX   $63           ;Compare the exponents of the two values
8C9B: BNE   $8CBE         ;And branch if they are different
8C9D: LDA   ($26),Y       ;Get M1 of the value in memory
8C9F: ORA   #$80          ;Set bit seven
8CA1: CMP   $64           ;And compare it to M1 of FAC1
8CA3: BNE   $8CBE         ;Branch if the two values are different

```

```

8CA5:  INY                ;Increment the index pointer to MANTISSA 2 (M2)
8CA6:  LDA  ($26),Y      ;Get M2 of the value in memory
8CA8:  CMP  $65          ;Compare it to M2 of FAC1
8CAA:  BNE  $8CBE        ;Branch if the two values are different
8CAC:  INY                ;Increment the index pointer to MANTISSA 3 (M3)
8CAD:  LDA  ($26),Y      ;Get M3 of the value in memory
8CAF:  CMP  $66          ;Compare it to M3 of FAC1
8CB1:  BNE  $8CBE        ;Branch if the two values are different
8CB3:  INY                ;Increment the index pointer to MANTISSA 4 (M4)
8CB4:  LDA  #$7F         ;If the value is to be
8CB6:  CMP  $71          ;Rounded, then compensate by setting carry flag
8CB8:  LDA  ($26),Y      ;Get M4 of the value in memory
8CBA:  SBC  $67          ;Compare it to M4 of FAC1
8CBC:  BEQ  $8CE8        ;And exit if the two values are the same
8CBE:  LDA  $68          ;Get the sign of FAC1
8CC0:  BCC  $8CC4        ;If M4 in memory is larger than M4 in FAC1, exit
8CC2:  EOR  #$FF         ;Result smaller, then invert the sign
8CC4:  JMP  $8C5D        ;Exit

```

```

*****
*
*                               QINT                               *
*
*          Convert FAC1 into an INTeGer within FAC1                *
*
* This routine converts the value that is in FAC1 into *
* a four byte signed integer in locations $64 - $67 *
* with the most significant byte first. *
*
*****

```

```

8CC7:  LDA  $63          ;If the value in FAC1 is zero,
8CC9:  BEQ  $8D18        ;Then exit
8CCB:  SEC                ;Subtract 160
8CCC:  SBC  #$A0         ;From the exponent of FAC1 to check for a whole
8CCE:  BIT  $68          ;Number, if the exponent of FAC1 is positive,
8CD0:  BPL  $8CDC        ;Then branch
8CD2:  TAX                ;If the result is negative, save accumulator
8CD3:  LDA  #$FF         ;Overflow marker for FAC1
8CD5:  STA  $03DF        ;Overflow marker for FAC1
8CD8:  JSR  $892C        ;Negate FAC1
8CDB:  TXA                ;Restore the accumulator
8CDC:  LDX  #$63         ;Set the index to the FAC1 address
8CDE:  CMP  #$F9         ;Branch if eight or more places
8CE0:  BPL  $8CE9        ;To be shifted
8CE2:  JSR  $8979        ;Perform the shift
8CE5:  STY  $03DF        ;And reset the OVERFLOW marker

```

```

8CE8: RTS                ;Exit the routine
8CE9: TAY                ;Save the accumulator
8CEA: LDA $68           ;Save bit seven
8CEC: AND #$80          ;Of the sign byte of FAC1
8CEE: LSR $64           ;Shift M1 right one bit
8CF0: ORA $64           ;Condition bit seven of M1 to indicate the sign
8CF2: STA $64           ;Bit seven set = negative sign
8CF4: JSR $8990         ;Normalize M2, M3, and M4
8CF7: STY $03DF        ;Reset the OVERFLOW marker
8CFA: RTS                ;And exit the routine

```

```

*****
*
*                               BASIC INT FUNCTION
*
* Function syntax: INT (X)
*
* This function returns the integer form of the number
* specified for 'X'. It accomplishes this by taking
* the value contained within the parentheses and
* removing the fractional portion of the number.
*
* Enter this routine with the value to be converted
* in FAC1. The routine will return the integer form
* to FAC1.
*
*****

```

```

8CFB: LDA $63           ;Get the exponent of FAC1 plus 129
8CFD: CMP #$A0          ;If the exponent is greater than 30, then exit
8CFF: BCS $8D21         ;Because, there cannot be any fractional values
                        ;Above the exponent of 30
8D01: JSR $AA68         ;Change FAC1 to a four byte signed integer
8D04: STY $71           ;Clear the OVERFLOW flag
8D06: LDA $68           ;Get the sign of FAC1
8D08: STY $68           ;Make the sign of FAC1 positive
8D0A: EOR #$80          ;Invert the sign
8D0C: ROL               ;And put the value into the carry flag
8D0D: LDA #$A0          ;Make the exponent equal to 30 + 129
8D0F: STA $63           ;Save the exponent
8D11: LDA $67           ;Move M4 into a
8D13: STA $9            ;Temporary storage area
8D15: JMP $88B1         ;Make FAC1 left binding.
8D18: STA $64           ;Fill the
8D1A: STA $65           ;MANTISSA of
8D1C: STA $66           ;FAC1 with
8D1E: STA $67           ;Zeros

```

```

8D20: TAY          ;Zero the Y register
8D21: RTS          ;Exit the routine

```

```

*****
*
*                      ASCFPN
*
*   Change an ASCII number to a Floating Point Number
*
*   This routine converts an ASCII number to its
*   Floating Point equivalent in FAC1. For ASCII values
*   stored in BANK 0, point TXTPTR to the first
*   character of the ASCII string and put the first
*   character in the accumulator. Also, set location
*   $03DA to zero for BANK 0. Set $03DA to a value
*   greater than zero for BANK 1 and put the address of
*   the ASCII string into locations $24, $25.
*
*****

```

```

8D22: STX  $03DA
8D25: LDY  #0          ;Clear FAC1
8D27: LDX  #$A        ;And FAC2 with zeros
8D29: STY  $5F,X
8D2B: DEX
8D2C: BPL  $8D29
8D2E: BCC  $8D3F      ;Branch if the first character is numeric
8D30: CMP  #'-'      ;If not, then check for a minus sign '-'
8D32: BNE  $8D38      ;If it is not a minus sign, then branch
8D34: STX  $69        ;If it is a minus sign, then set flag negative
8D36: BEQ  $8D3C      ;Branch to get the next character
8D38: CMP  #'+'      ;Is the first character a plus sign '+'?
8D3A: BNE  $8D41      ;No, then branch
8D3C: JSR  $8DF5      ;Get the next character from the string
8D3F: BCC  $8D9C      ;Branch if the character is a digit 0 - 9
8D41: CMP  #'.'      ;Is the character a decimal point '.'?
8D43: BEQ  $8D73      ;If it is, then branch
8D45: CMP  #'E'      ;Is the character an 'E' for exponentiation
8D47: BNE  $8D79      ;No, then branch
8D49: JSR  $8DF5      ;Get the next char from the string after the 'E'
8D4C: BCC  $8D65      ;If it is a digit 0 - 9, then branch
8D4E: CMP  #'-'      ;Is it the token for minus?
8D50: BEQ  $8D60      ;If so, then branch
8D52: CMP  #'-'      ;Is it the minus sign?
8D54: BEQ  $8D60      ;If so, then branch
8D56: CMP  #'+'      ;Is it the token for plus?
8D58: BEQ  $8D62      ;If so, then branch

```

---

```

8D5A:  CMP    #'+'      ;Is it the plus sign?
8D5C:  BEQ    $8D62     ;If so, then branch
8D5E:  BNE    $8D67     ;If it is not the plus sign, then combine the
                        ;Exponent and the fraction
8D60:  ROR    $62      ;Set the flag for a negative exponent
8D62:  JSR    $8DF5     ;Get the next character of the string
8D65:  BCC    $8DC3     ;Branch if it is a digit 0 - 9
8D67:  BIT    $62      ;If the sign of the exponent
8D69:  BPL    $8D79     ;Is positive, then branch
8D6B:  LDA    #0       ;Calculate the two's complement
8D6D:  SEC                    ;Of the exponent
8D6E:  SBC    $60      ;And save it
8D70:  JMP    $8D7B     ;Combine the exponent and the fraction
8D73:  ROR    $61      ;Set the flag for decimal point
8D75:  BIT    $61      ;If a decimal point was already found,
8D77:  BVC    $8D3C     ;Then branch - this is the end of the number
8D79:  LDA    $60      ;Get the two's complement of the exponent
8D7B:  SEC                    ;Subtract the number of digits
8D7C:  SBC    $5F      ;After the decimal point from the exponent
8D7E:  STA    $60      ;And save the result
8D80:  BEQ    $8D94     ;If there are no digits after dec. point, exit
8D82:  BPL    $8D8D     ;If value is positive, then multiply FAC by 10
8D84:  JSR    $8B38     ;If value is negative, then divide the FAC by 10
8D87:  INC    $60      ;Increase the exponent by one
8D89:  BNE    $8D84     ;Continue until the decimal point has been
                        ;Completely shifted
8D8B:  BEQ    $8D94     ;Exit
8D8D:  JSR    $8B17     ;Multiply the FAC by 10
8D90:  DEC    $60      ;Subtract one from the exponent
8D92:  BNE    $8D8D     ;Continue until the decimal point has been
                        ;Completely shifted
8D94:  LDA    $69      ;If the value is negative,
8D96:  BMI    $8D99     ;Then branch to set the negative flag for FAC1
8D98:  RTS                    ;Else, exit
8D99:  JMP    $8FFA     ;Adjust the negative flag in FAC1
8D9C:  PHA                    ;Save character from the string onto the stack
8D9D:  BIT    $61      ;Check if there was a dec. point before number
8D9F:  BPL    $8DA3     ;Increment the decimal place counter by one
8DA1:  INC    $5F      ;Increment the decimal place counter by one
8DA3:  JSR    $8B17     ;Multiply FAC1 by 10
8DA6:  PLA                    ;Get character from string back off the stack
8DA7:  SEC                    ;Convert the ASCII digit
8DA8:  SBC    #$30     ;To its HEX equivalent
8DAA:  JSR    $8DB0     ;Add the value in the accumulator to FAC1
8DAD:  JMP    $8D3C     ;Continue with the next character

```

```

*****
*
*                               FINLOG                               *
*
*                               FAC1 = FAC1 + Accumulator           *
*
* This routine takes the numerical value that is in                 *
* the accumulator and adds that value to the Floating               *
* Point value that is in FAC1.                                     *
*
*****

8DB0: PHA                ;Save the value to be added onto the stack
8DB1: JSR   $8C38        ;Transfer FAC1 to FAC2
8DB4: PLA                ;Get the value to be added off the stack
8DB5: JSR   $8C68        ;Put the value in the accumulator into FAC1
8DB8: LDA   $6F          ;Get the sign from FAC2
8DBA: EOR   $68          ;Compare the sign of FAC1 to the sign of FAC2
8DBC: STA   $70          ;And save the result
8DBE: LDX   $63          ;Get the exponent of FAC1 into the X register
8DC0: JMP   $8848        ;Add FAC1 to FAC2
8DC3: LDA   $60          ;Get the exponent of the Floating Point number
8DC5: CMP   #$0A        ;If the value of the exponent is less
8DC7: BCC   $8DD2        ;Than 10, then branch
8DC9: LDA   #$64        ;If the sign of the exponent
8DCB: BIT   $62          ;Is negative,
8DCD: BMI   $8DF0        ;Then branch
8DCF: JMP   $895D        ;If the sign of the exponent is positive,
                        ;Then and error will result
8DD2: ASL                ;Multiply the exponent by 2
8DD3: ASL                ;By 4
8DD4: CLC
8DD5: ADC   $60          ;By 5
8DD7: ASL                ;By 10
8DD8: CLC
8DD9: LDY   #0          ;Zero the index pointer
8DDB: STA   $79          ;Save the exponent
8DDD: LDA   $03DA       ;Get the bank number to get the character from
8DE0: BNE   $8DE8        ;If the bank is not zero, then get the character
                        ;From BANK 1
8DE2: JSR   $03C9        ;Get the character which is pointed to by TXTPTR
8DE5: JMP   $8DEB        ;Add it to the exponent
8DE8: JSR   $03B7        ;Get the char. which is pointed to by locations
                        ;$24, $25 from BANK 1
8DEB: ADC   $79          ;Add it to the exponent
8DED: SEC                ;Subtract $30
8DEE: SBC   #$30        ;To create the HEX equivalent

```

```

8DF0: STA    $60      ;And save it
8DF2: JMP    $8D62    ;Continue to the next character
8DF5: LDA    $03DA    ;Get the bank number to get the character from
8DF8: BNE    $8DFD    ;If the bank number is not zero, then branch to
                        ;Get the character from BANK 1
8DFA: JMP    $0380    ;If the bank number is zero, then get the next
                        ;Character from the CHRGET routine
    
```

```

*****
*
*   If this routine is entered at this point, you can
*   get a character from BANK 1 and process it the same
*   way that CHRGET does.
*
*****
    
```

```

8DFD: INC    $24      ;Add one to the pointers LSB
8DFF: BNE    $8E03    ;No carry, then branch
8E01: INC    $25      ;If there is a carry, add one to the MSB
8E03: LDY    #0       ;Zero the index pointer
8E05: JSR    $03B7    ;Get the next character
8E08: CMP    #' :'    ;Is it => than the statement delimiter
8E0A: BCS    $8E16    ;If it is equal to or greater than
                        ;The delimiter ':', then exit
8E0C: CMP    #$20     ;Is it a space?
8E0E: BEQ    $8DFD    ;Skip the space and get the next character
8E10: SEC                        ;Set the carry for subtraction
8E11: SBC    #'0'     ;Convert from ASCII to numeric digit
8E13: SEC                        ;Set carry for subtraction
8E14: SBC    #$D0     ;If the value is less than the ASCII value for
                        ;Zero, then set the carry flag
8E16: RTS                        ;The carry flag is cleared for a numeric
                        ;Character or the delimiter, exit the routine
    
```

```

*****
*
*                               N0999
*
*   Constants for Floating Point Numbers
*
*   The Floating Point numbers listed below are used
*   by the different routines when converting a string
*   to a Floating Point number.
*
*****
    
```

```

                EX  M1  M2  M3  M4
                --  --  --  --  --
8E17:  .BYTE $9B,$3E,$BC,$1F,$FD ;   99,999,999.9
8E1C:  .BYTE $9E,$6E,$6B,$27,$FD ;  999,999,999
8E21:  .BYTE $9E,$6E,$6B,$28,$00 ;1,000,000,000 1E9
    
```

```

*****
*
*                               INPRT
*
*       PRinT 'IN' followed by the line number
*
* This routine prints the text 'IN' followed by the
* current BASIC line number pointed to by locations
* $3B, $3C.
*
*****
    
```

```

8E26: JSR   $9281      ;Print 'IN'
8E29: TXT   ' in '
8E2D: .BYTE $00        ;End of text flag
8E2E: LDA   $3C        ;Get the MSB of the line number we are on
8E30: LDX   $3B        ;Get the LSB of the line number we are on
                        ;And fall through to print the line number
    
```

```

*****
*
*                               LINPRT
*
*       Output the ASCII characters of a number
*
* This routine is used to output an ASCII number
* whose value is held in the accumulator (MSB) and the
* X register (MSB). This process is accomplished by
* first converting the value of the ASCII number to
* its Floating Point equivalent and then to its ASCII
* form. The series of ASCII characters that
* represent the ASCII number are then stored starting
* at location $0100.
*
*****
    
```

```

8E32: STA   $64        ;Save the MSB and
8E34: STX   $65        ;The LSB of the value
8E36: LDX   #$90       ;Set the maximum exponent allowed (15)
8E38: SEC
8E39: JSR   $8C75      ;Convert the integer value to
    
```



```

;Floating Point format in FAC1
8E3C: JSR   $8E44   ;Convert FAC1 to an ASCII string at $0100
8E3F: JMP   $55E2   ;Print the ASCII string

```

```

*****
*
*                               FOUT
*
*   Convert the contents of FAC1 into an ASCII string
*
*   This routine takes the Floating Point value in FAC1
*   and converts it to an ASCII string.  It then stores
*   the ASCII string starting at address $0100.  The
*   ASCII string is delimited at the end by a zero.
*
*****

```

```

8E42: LDY   #1       ;Index the first byte of the buffer
8E44: LDA   #32      ;Value for a space
8E46: BIT   $68      ;If the value
8E48: BPL   $8E4C    ;In FAC1 is negative,
8E4A: LDA   #'-'     ;Then store a minus sign '-'
8E4C: STA   $00FF,Y  ;As the first character of the number
8E4F: STA   $68      ;If the value is positive,
8E51: STY   $72      ;Then store a space
8E53: INY                ;Move the index to the next byte in the buffer
8E54: LDA   #'0'     ;Value for an ASCII '0'
8E56: LDX   $63      ;Get the exponent of FAC1
8E58: BNE   $8E5D    ;And branch if it is not equal to zero
8E5A: JMP   $8F69    ;If FAC1 equals zero, then store an ASCII zero
;In the buffer and exit
8E5D: LDA   #0       ;Zero the accumulator
8E5F: CPX   #$80     ;Compare the exponent to an exponent of '-1'
8E61: BEQ   $8E65    ;If it is a '-1', then branch
8E63: BCS   $8E6E    ;If the exponent is => than zero
;Then branch
8E65: LDA   #$21     ;Pointer to the constant
8E67: LDY   #$8E     ;'1E09'
8E69: JSR   $8A08    ;Multiply FAC1 by '1E09'
8E6C: LDA   #$F7     ;Set the exponent to '-09'
8E6E: STA   $5F      ;And save it
8E70: LDA   #$1C     ;Pointer to constant
8E72: LDY   #$8E     ;999,999,999
8E74: JSR   $8C87    ;Compare FAC1 to the constant
8E77: BEQ   $8E97    ;If they are equal, then branch
8E79: BPL   $8E8D    ;Branch if FAC1 is less than 999,999,999

```

```

*****
*
* At this point in this routine, if the two
* aforementioned branches were false, then the value
* in FAC1 is greater than 99,999,999.
*
*****

8E7B: LDA  #$17      ;Pointer to constant
8E7D: LDY  #$8E      ;99,999,999.9
8E7F: JSR  $8C87     ;Compare FAC1 to 99,999,999.9
8E82: BEQ  $8E86     ;Branch if they are equal
8E84: BPL  $8E94     ;Branch if FAC1 is less than 99,999,999.9

*****
*
* At this point in this routine, if the two
* aforementioned branches were false, then the value
* in FAC1 is greater than 99,999,999.9.
*
*****

8E86: JSR  $8B17     ;FAC1 = FAC1 * 10
8E89: DEC  $5F       ;Decrement the exponent by one
8E8B: BNE  $8E7B     ;Continue until FAC1 = FAC1 > X.9
8E8D: JSR  $8B38     ;FAC1 = FAC1 / 10
8E90: INC  $5F       ;Increment the exponent by one
8E92: BNE  $8E70     ;Continue until FAC1 is greater than 999,999,999
                        ;Or until nine shifts have been completed
8E94: JSR  $8A0E     ;FAC1 = FAC1 + 0.5
8E97: JSR  $8CC7     ;Convert FAC1 to a four byte signed integer
8E9A: LDX  #1        ;Index pointer to position of the decimal point
8E9C: LDA  $5F       ;Get the exponent
8E9E: CLC          ;Clear the carry flag for addition
8E9F: ADC  #$0A      ;Add 10 to the exponent
8EA1: BMI  $8EAC     ;If smaller than 10, then branch
8EA3: CMP  #$0B      ;If greater than 10,
8EA5: BCS  $8EAD     ;Then branch
8EA7: ADC  #$FF      ;Invert the exponent
8EA9: TAX          ;Save it in the X register
8EAA: LDA  #2        ;Set up to end with a zero for the exponent
8EAC: SEC          ;
8EAD: SBC  #2        ;Subtract 2 from the exponent
8EAF: STA  $60       ;Save the exponent
8EB1: STX  $5F       ;And the place of the decimal point
8EB3: TXA          ;Restore the accumulator
8EB4: BEQ  $8EB8     ;If the value is equal to zero, then branch

```

```

8EB6: BPL    $8ECB    ;If the exponent is positive, then branch
8EB8: LDY    $72     ;Get the index to the next byte in the buffer
8EBA: LDA    #$2E    ;Value for a decimal point
8EBC: INY    ;Move to the next free byte in the buffer
8EBD: STA    $00FF,Y ;Save the decimal point in the ASCII number
8EC0: TXA    ;Restore the accumulator
8EC1: BEQ    $8EC9    ;If the value is equal to zero, then branch
8EC3: LDA    #$30    ;Store the ASCII value
8EC5: INY    ;For a zero into the next
8EC6: STA    $00FF,Y ;Available space in the buffer
8EC9: STY    $72     ;Save the index
8ECB: LDY    #0      ;Clear the index
8ECD: LDX    #$80    ;Set up the initial digit
8ECF: LDA    $67     ;Get MANTISSA 4 (M4) of FAC1
8ED1: CLC    ;
8ED2: ADC    $8F7E,Y ;Add M4 of the table entry to it
8ED5: STA    $67     ;Save it back to FAC1
8ED7: LDA    $66     ;Do the same
8ED9: ADC    $8F7D,Y ;For M3
8EDC: STA    $66     ;
8EDE: LDA    $65     ;And M2
8EE0: ADC    $8F7C,Y ;
8EE3: STA    $65     ;
8EE5: LDA    $64     ;And M1
8EE7: ADC    $8F7B,Y ;
8EEA: STA    $64     ;
8EEC: INX    ;Add one to the digit
8EED: BCS    $8EF3    ;If there was not any overflow
8EEF: BPL    $8ECF    ;And the result is positive, then repeat
8EF1: BMI    $8EF5    ;If result is negative without overflow, branch
8EF3: BMI    $8ECF    ;If the result is negative and there was an
                        ;Overflow then repeat
8EF5: TXA    ;Save the digit into the accumulator
8EF6: BCC    $8EFC    ;If there is no overflow, then branch
8EF8: EOR    #$FF    ;Invert the digit
8EFA: ADC    #$A     ;Add 10 to the digit
8EFC: ADC    #$2F    ;And add 47 to the digit
8EFE: INY    ;Move the index
8EFF: INY    ;To the next table entry
8F00: INY    ;
8F01: INY    ;
8F02: STY    $49     ;Save the table index
8F04: LDY    $72     ;Get the index to the string buffer at $0100
8F06: INY    ;Increment it by one
8F07: TAX    ;Move the digit to the X register
8F08: AND    #$7F    ;Drop bit seven of the digit
8FOA: STA    $00FF,Y ;And save it in the buffer

```

---

```

8F0D: DEC    $5F      ;Subtract one from the position of decimal point
8F0F: BNE    $8F17    ;Branch if we have not reached the decimal point
8F11: LDA    #$2E     ;Get the value from the decimal point
8F13: INY                    ;Move to the next place in the buffer
8F14: STA    $00FF,Y  ;Save decimal point to the number in the buffer
8F17: STY    $72      ;Save the index to the buffer
8F19: LDY    $49      ;Get the table index
8F1B: TXA                    ;Move the digit to the accumulator
8F1C: EOR    #$FF     ;Return it to its original form
8F1E: AND    #$80     ;Preserve bit seven
8F20: TAX                    ;And save the result to the X register
8F21: CPY    #$24     ;If we are at the ninth table entry,
8F23: BEQ    $8F29    ;Then branch
8F25: CPY    #$3C     ;If we are at the fifteenth table entry,
8F27: BNE    $8ECF    ;Then branch
8F29: LDY    $72      ;Get the buffer index
8F2B: LDA    $00FF,Y  ;Get the last character that was put into buffer
8F2E: DEY                    ;Decrement the index by one
8F2F: CMP    #$30     ;Is this character a zero?
8F31: BEQ    $8F2B    ;If so, then move to the next character
8F33: CMP    #$2E     ;If it is not a zero, is it a decimal point?
8F35: BEQ    $8F38    ;If so, then remove it
8F37: INY                    ;Move to the next character in the buffer
8F38: LDA    #$2B     ;Get the value for a plus sign '+'
8F3A: LDX    $60      ;Get the exponent of the value
8F3C: BEQ    $8F6C    ;If it is equal to zero, then place delimiter
                        ;In the buffer and exit
8F3E: BPL    $8F48    ;If it is positive, then place the plus sign
                        ;'+' into the buffer
8F40: LDA    #0       ;Generate the two's
8F42: SEC                    ;Complement of
8F43: SBC    $60      ;The exponent
8F45: TAX                    ;Save the complemented exponent into X register
8F46: LDA    #'-'     ;Get the value for a minus sign '-'
8F48: STA    $0101,Y  ;Save the ASCII sign into the buffer
8F4B: LDA    #'E'     ;Get the value for 'E'
8F4D: STA    $0100,Y  ;Store it into the buffer
8F50: TXA                    ;Move the complemented exponent to accumulator
8F51: LDX    #$2F     ;Get the ASCII base digit of '-1'
8F53: SEC                    ;
8F54: INX                    ;Add one to the ASCII base digit
8F55: SBC    #$0A     ;Subtract 10 from the exponent
8F57: BCS    $8F54    ;Continue until an underflow occurs
8F59: ADC    #$3A     ;Convert the second exponent digit to ASCII
8F5B: STA    $0103,Y  ;Save it into the buffer
8F5E: TXA                    ;Move exponent's first digit into accumulator
8F5F: STA    $0102,Y  ;Store it into the buffer

```

```

8F62: LDA  #0           ;Get the delimiter value of zero
8F64: STA  $0104,Y      ;And store it into the buffer
8F67: BEQ  $8F71        ;Exit
8F69: STA  $00FF,Y      ;Store the accumulator into the buffer
8F6C: LDA  #0           ;Place the delimiter value of zero
8F6E: STA  $0100,Y      ;After the string in the buffer
8F71: LDA  #0           ;Exit with the address of the ASCII string
8F73: LDY  #1           ;In the accumulator and Y register (LSB/MSB)
8F75: RTS                ;Exit

```

```

*****
*                                                                           *
*                               FHALF                                     *
*                                                                           *
*           Constant for the SQR and for rounding numbers             *
*                                                                           *
*   This five byte Floating Point number is the value of             *
*   1/2 and is used for the SQR function and for                      *
*   rounding numbers.                                                 *
*                                                                           *
*****

```

```

      EX  M1  M2  M3  M4
      --  --  --  --  --

```

```

8F76: .BYTE $80,$00,$00,$00,$00 ;          0.5

```

```

*****
*                                                                           *
*                               FOUTBL                                   *
*                                                                           *
*           Table of divisors for conversion to decimal.             *
*                                                                           *
*   This table of powers of ten is used when converting             *
*   a number to decimal format.                                       *
*                                                                           *
*****

```

```

8F7B: .BYTE $FA,$0A,$1F,$00      ; -100,000,000
8F7F: .BYTE $00,$98,$96,$80      ; + 10,000,000
8F83: .BYTE $FF,$F0,$BD,$C0      ; - 1,000,000
8F87: .BYTE $00,$01,$86,$A0      ; +   100,000
8F8B: .BYTE $FF,$FF,$D8,$F0      ; -   10,000
8F8F: .BYTE $00,$00,$03,$EB      ; +     1,000
8F93: .BYTE $FF,$FF,$FF,$9C      ; -      100
8F97: .BYTE $00,$00,$00,$0A      ; +       10
8F9B: .BYTE $FF,$FF,$FF,$FF      ; -        1

```

```

*****
*
*                               FDCEND
*
*           Constants for TI$ conversion
*
*   This table contains the Floating Point number values
*   necessary when converting TI$ to ASCII.
*
*****

```

```

8F9F: .BYTE $FF,$DF,$0A,$80 ; 2,160,000
8FA3: .BYTE $00,$03,$4B,$C0 ; 216,000
8FA7: .BYTE $FF,$FF,$73,$60 ; 36,000
8FAB: .BYTE $00,$00,$0E,$10 ; 3,600
8FAF: .BYTE $FF,$FF,$FD,$A8 ; 600
8FB3: .BYTE $00,$00,$00,$3C ; 60

```

```

*****
*
*                               BASIC SQR FUNCTION
*
*   Function syntax: SQR (X)
*
*   This function returns the square root of the number
*   or expression specified for 'X'.
*
*****

```

```

8FB7: JSR   $8C38 ;Move FAC1 to FAC2
8FBA: LDA   #$76 ;Pointer to the constant
8FBC: LDY   #$8F ;(0.5)
8FBE: JSR   $8BD4 ;Transfer the constant into FAC1

```

```

*****
*
*                               BASIC EXP FUNCTION
*
*   Function syntax: EXP (X)
*
*   This function returns a value which is equal to
*   the value of 2.7182813 to the power specified by
*   the value of 'X'.
*
*****

```

```

8FC1: BEQ    $9033
8FC3: LDA    $6A      ;If FAC2 does not equal zero,
8FC5: BNE    $8FCA    ;Then branch
8FC7: JMP    $88D8    ;If FAC2 is equal to zero, then exit
8FCA: LDX    #$50     ;Transfer the contents
8FCC: LDY    #0       ;Of FAC1
8FCE: JSR    $8C00    ;To location $0050
8FD1: LDA    $6F      ;Check the sign of FAC2
8FD3: BPL    $8FE4    ;And branch if it is positive
8FD5: JSR    $8CFB    ;Convert FAC1 to integer format
8FD8: LDA    #$50     ;Set the accumulator and Y register to point to
8FDA: LDY    #0       ;The constant
8FDC: JSR    $8C87    ;Compare FAC1 with the constant
8FDF: BNE    $8FE4    ;If they are not equal, then branch
8FE1: TYA
8FE2: LDY    $09      ;Get the sign change flag
8FE4: JSR    $8C2A    ;Move FAC2 to FAC1
8FE7: TYA            ;Save the sign change
8FE8: PHA            ;flag onto the stack
8FE9: JSR    $89CA    ;Perform the BASIC log function
8FEC: LDA    #$50     ;Set the accumulator and Y register to point to
8FEE: LDY    #0       ;The constant
8FF0: JSR    $8A24    ;Move the constant from BANK 1 to FAC2
8FF3: JSR    $9033    ;Perform the BASIC EXP function
8FF6: PLA            ;Get the sign change flag back
8FF7: LSR            ;Shift it into the carry flag
8FF8: BCC    $9004    ;If the carry is clear then exit
8FFA: LDA    $63      ;Else check the exponent
8FFC: BEQ    $9004    ;If it is equal to zero, then exit
8FFE: LDA    $68
9000: EOR    #$FF     ;Else invert the exponent
9002: STA    $68      ;And save it as the sign byte
9004: RTS            ;Exit the routine

```

```

*****
*
*           Constants for EXP
*
*****

```

	EX	M1	M2	M3	M4	
	--	--	--	--	--	
9005:	.BYTE	\$81,\$38,\$AA,\$3B,\$29				; 1.44269504 = 1/LOG(2)
900A:	.BYTE	\$07				; 7 = POLYNOMIAL DEGREE
900B:	.BYTE	\$71,\$34,\$58,\$3E,\$56				; 2.14987637E-5
9010:	.BYTE	\$74,\$16,\$7E,\$B3,\$1B				; 1.4352314E-4
9015:	.BYTE	\$77,\$2F,\$EE,\$E3,\$85				; 1.34226348E-3

```

901A: .BYTE $7A,$1D,$84,$1C,$2A ; 9.61401701E-3
901F: .BYTE $7C,$63,$59,$58,$9A ; .055505129
9024: .BYTE $7E,$75,$FD,$E7,$C6 ; .240226385
9029: .BYTE $80,$31,$72,$18,$10 ; .693147186
902E: .BYTE $81,$00,$00,$00,$00 ; 1

```

```

*****
*
*          BASIC EXP FUNCTION
*
* Function syntax:  EXP (X)
*
* NOTE:  X = the power that e (2.7182813) is to be
*         raised to
*
* This function will return a value that is the result
* of the value e (2.7182813) raised to the power
* specified by 'X'.
*
*****

```

```

9033: LDA    #$05      ;Pointer to constant
9035: LDY    #$90      ;1/LOG(2)
9037: JSR    $8A08     ;Multiply FAC1 BY 1/LOG(2)
903A: LDA    $71       ;Get the overflow byte
903C: ADC    #$50      ;Add 80 to it
903E: BCC    $9043     ;If there was no overflow, then branch
9040: JSR    $8C4F     ;Round FAC1
9043: STA    $58       ;Save the overflow value temporarily
9045: JSR    $8C3B     ;Move FAC1 to FAC2
9048: LDA    $63       ;Get the exponent of FAC1
904A: CMP    #$88      ;If the exponent of FAC1
904C: BCC    $9051     ;Is less than 8, then branch
904E: JSR    $8B09     ;If it is equal to or greater than 8, then
                       ;Generate an overflow error
9051: JSR    $8CFB     ;Convert the value in FAC1 to integer format
9054: LDA    $09       ;Get the integer format value
9056: CLC          ;Clear the carry for addition
9057: ADC    #$81      ;Add 129 to it to create an exponent
9059: BEQ    $904E     ;If the value was equal to or greater than 127,
                       ;Then generate an overflow error
905B: SEC          ;Set the carry for subtraction
905C: SBC    #$01      ;Subtract one to restore the value
905E: PHA          ;Save it onto the stack
905F: LDX    #5        ;Transfer the five bytes
9061: LDA    $6A,X     ;Of FAC2
9063: LDY    $63,X     ;To FAC1

```



```

9065: STA $63,X
9067: STY $6A,X
9069: DEX
906A: BPL $9061
906C: LDA $58 ;Restore the
906E: STA $71 ;Overflow byte
9070: JSR $8831 ;Subtract FAC2 from FAC1
9073: JSR $8FFA ;Invert the sign of FAC1 if FAC1 > 0
9076: LDA #$0A ;Get the LSB of the number of the
;Polynomial coefficient and the
9078: LDY #$90 ;MSB of the address of the polynomial degree
907A: JSR $909C ;Calculate the polynomial of the value in FAC1
907D: LDA #0 ;Clear the sign comparison flag
907F: STA $70 ;For FAC1 and FAC2
9081: PLA ;Get the original exponent back off the stack
9082: JSR $8AEE ;Add it to the present exponent of FAC1
9085: RTS ;And exit the routine

```

```

*****
*
* POLYNOMIAL CALCULATION
*
* Formula: Y = A1 * X + A2 * X^3 + A3 * X^5
*
*****

```

```

9086: STA $72 ;Save the LSB and the MSB of the address
9088: STY $73 ;Of the polynomial degree
908A: JSR $8BFC ;Move the value to calculate the polynomial from
;FAC1 to location $59
908D: LDA #$59 ;Get the pointer to the address of the value to
;Calculate the polynomial of
908F: JSR $8A24 ;Square it (value ^2)
9092: JSR $90A0 ;Calculate the polynomial of the value in FAC1
9095: LDA #$59 ;Get the address of the value
9097: LDY #0 ;To calculate the polynomial of
9099: JMP $8A24 ;Multiply FAC1 by that value

```

```

*****
*
* POLYNOMIAL CALCULATION
*
* Formula: Y = A0 + A1 * X + A2 * X^2 + A3 * X^3
*
*****

```

```

909C: STA $72 ;Save the LSB and the
909E: STY $73 ;And the MSB of the address of the polynomial
90A0: JSR $8BF9 ;Move the value to calculate the polynomial of
;From FAC1 to location $5E
90A3: LDA ($72),Y ;Get the polynomial degree
90A5: STA $69 ;Save it
90A7: LDY $72 ;Get the LSB of the address
90A9: INY ;Increment it to the first coefficient
90AA: TYA ;Move the LSB to the accumulator
90AB: BNE $90AF ;If there was an overflow, then branch
90AD: INC $73 ;Add one to the MSB of the address
90AF: STA $72 ;Save the LSB
90B1: LDY $73 ;Get the MSB of the coefficient address
90B3: JSR $8A08 ;Multiply FAC1 by the coefficient
90B6: LDA $72 ;Get the LSB and the
90B8: LDY $73 ;MSB of the address to the coefficient
90BA: CLC ;Clear the carry for addition
90BB: ADC #$05 ;Add 5 to move to the next coefficient
90BD: BCC $90C0 ;If there was an overflow, then branch
90BF: INY ;Increment the MSB by one
90C0: STA $72 ;Save the address
90C2: STY $73 ;Back
90C4: JSR $8A12 ;Round FAC1 by adding 0.5 to it
90C7: LDA #$5E ;Get the address of the value
90C9: LDY #0 ;To calculate the polynomial of
90CB: DEC $69 ;Subtract one from the polynomial degree
90CD: BNE $90B3 ;Continue until the degree equals zero
90CF: RTS ;Exit the routine

```

```

*****
*
* IOERR
*
* I/O ERROR message handler
*
* This routine handles the interface from KERNAL to
* BASIC whenever an ERROR occurs during a KERNAL I/O
* operation. On entry to this routine, the
* accumulator contains the error number. If the error
* number is 0, then the error number is changed to
* 30 to indicate a 'BREAK' error.
*
*****

```

```

90D0: TAX ;Save the error number into the X register
90D1: BNE $90D5 ;If the error number is greater than 0, branch
90D3: LDX #30 ;'BREAK' error

```

90D5: JMP \$4D3C ;Jump to the error message routine

```
*****
*
*                               BSOPEN
*
* This routine does a BANK 15 command and then calls
* the KERNAL OPEN routine. Note: You must handle the
* error checking. After calling this routine, if an
* error occurred, the carry flag will be set and the
* error number will be in the accumulator.
*
*****
```

90D8: JSR \$A845 ;BANK 15  
 90DB: JSR \$FFC0 ;KERNAL OPEN routine  
 90DE: RTS ;Exit the routine

```
*****
*
*                               BSOUT
*
* This routine does a BANK 15 command and then calls
* the KERNAL CHROUT routine. Enter this routine with
* the value to output stored in the accumulator.
*
*****
```

90DF: JSR \$9269 ;BANK 15, JMP CHROUT  
 90E2: BCS \$90D0 ;If an error occurred, then branch (C=1)  
 90E4: RTS ;Exit the routine

```
*****
*
*                               BASIN
*
* This routine does a BANK 15 command and then calls
* the KERNAL CHRIN routine.
*
*****
```

90E5: JSR \$9263 ;BANK 15, JMP TO CHRIN  
 90E8: BCS \$90D0 ;If an error occurred, then branch  
 90EA: RTS ;Exit the routine

```

*****
*
*                                     *
*                               BASIC CHKOUT                               *
*                                     *
*                               Set the Output Device                       *
*                                     *
* This routine is used by the BASIC CMD command to set *
* the current output device.  If an error occurs, then *
* IOERR is automatically called.  On entry to this *
* routine, the X register contains the Logical File *
* number. *
* *
*****

90EB: PHA                ;Save the accumulator onto the stack temporarily
90EC: JSR    $A845        ;BANK 15
90EF: JSR    $FFC9        ;Kernal CHKOUT, set the device in the X register
                               ;To output
90F2: JSR    $9243        ;Set the flag to obtain a new DS$
90F5: TAX                ;Move the error number to CHKOUT, if any, to the
                               ;X register
90F6: PLA                ;Restore the accumulator
90F7: BCC    $90FC        ;If no error, then exit
90F9: TXA                ;Move the error number to the accumulator
90FA: BCS    $90D0        ;Branch to handle the error
90FC: RTS                ;Exit the routine

*****
*
*                                     *
*                               BASIC CHKIN                               *
*                                     *
* This routine does a BANK 15 command and then calls *
* the KERNAL CHKIN routine. *
* *
*****

90FD: JSR    $A845        ;BANK 15
9100: JSR    $FFC6        ;Kernal CHKIN, set the device in the X register
                               ;To input
9103: JSR    $9243        ;Set the flag to obtain a new DS$
9106: BCS    $90D0        ;If an error occurred, then branch
9108: RTS                ;Exit the routine

```

```
*****
*
*                               BASIC GETIN
*
* This routine does a BANK 15 command and then calls
* the KERNAL GETIN routine.
*
*****
```

```
9109: JSR  $A845      ;BANK 15
910C: JSR  $FFE4      ;Kernal GETIN, Get a character from the current
                        ;Input device into the accumulator
910F: BCS  $90D3      ;If an error occurred, then branch
9111: RTS
```

```
*****
*
*                               BASIC SAVE COMMAND
*
* Command syntax:  SAVE "filename" [,DV]
*
* NOTE:  DV = the device number to save to
*
* This command allows you to save a program to the
* device specified by 'DV'.
*
*****
```

```
9112: JSR  $91AE      ;Set up the parameters for the SAVE command
9115: LDX  $1210      ;Get the LSB and
9118: LDY  $1211      ;The MSB of the end address of the BASIC program
911B: LDA  #$2D        ;Set the pointer to the start of the
                        ;BASIC program
911D: JSR  $A845      ;BANK 15
9120: JSR  $FFD8      ;KERNAL SAVE
9123: JSR  $9243      ;Set the flag to obtain a new DS$
9126: BCS  $90D0      ;If an error occurred, then branch
9128: RTS
```

```
*****
*
*                               BASIC VERIFY COMMAND
*
* Command syntax:  VERIFY "filename" [,DV]
*
* NOTE:  DV = the device number to verify from
*
*****
```

```

* This command allows you to verify a program that
* is stored in memory with the one that is saved on
* tape or disk.
*

```

```

*****

```

```

9129: LDA #1 ;Flag for VERIFY
912B: .BYTE $2C ;MASK To skip to $912E

```

```

*****

```

```

*

```

```

* BASIC LOAD COMMAND

```

```

*

```

```

* Command syntax: LOAD "filename"[,DV][,RF]

```

```

*

```

```

* NOTE: DV = the device number to load from

```

```

* RF = the relocate flag

```

```

*

```

```

* This command allows you to load a program from the

```

```

* device specified by 'DV'. If the relocate flag 'RF'

```

```

* is equal to zero, then the program will be loaded

```

```

* to memory starting with the start of BASIC address.

```

```

* However, if the relocate flag is equal to one, then

```

```

* the program will be loaded to the address that it

```

```

* was originally saved from.

```

```

*

```

```

*****

```

```

912C: LDA #0 ;Flag for LOAD
912E: STA $0C ;Set the flag for LOAD/VERIFY
9130: JSR $91AE ;Set the parameters for the LOAD/VERIFY command
9133: JSR $A845 ;BANK 15
9136: LDA $0C ;Get the flag for LOAD/VERIFY
9138: LDX $2D ;Get the LSB and the
913A: LDY $2E ;MSB of the address of the start of BASIC
;Program area
913C: JSR $FFD5 ;KERNAL LOAD/VERIFY
913F: PHP ;Save the STATUS register
9140: JSR $9243 ;Set the flag to obtain a new DS$
9143: PLP ;Restore the STATUS register
9144: BCS $91AB ;If an error occurred, then branch
9146: LDA $0C ;Get the flag for LOAD/VERIFY
9148: BEQ $9160 ;If the flag equals zero, then branch to LOAD
914A: LDX #28 ;'VERIFY' message
914C: JSR $9251 ;BANK 15, READST - Read the serial bus status
914F: AND #$00010000 ;Check bit 4 (1 = verify error)
9151: BNE $9169 ;If an error occurred, then branch

```

```

9153: BIT   $7F           ;Check if we are in the DIRECT mode
9155: BMI   $915F        ;If not, then do not print 'OK', instead exit

```

```

*****
*
*   Print OK to the current output device and exit.
*
*****

```

```

9157: JSR   $9281        ;Print the following message
915A: .BYTE $0D          ;<cr>
915B: TXT   'ok'
915D: .BYTE $0D          ;<cr>
915E: .BYTE $00          ;Delimiter of zero
915F: RTS                ;Exit the routine

```

```

*****
*
*   This subroutine will ensure that no errors have
*   occurred by reading ST.  If any of the bits are set
*   with the exception of bit 6 which indicates the end
*   of Information, a 'LOAD' error will be generated.
*
*****

```

```

9160: JSR   $9251        ;BANK15, READST - Read the serial bus status
9163: AND   #$10111111  ;Mask the error (bit 6 EOF is not an error)
9165: BEQ   $916C        ;Bits if no error occurred, then branch
9167: LDX   #29           ;Error number for 'LOAD' error
9169: JMP   $4D3C        ;Jump to the error message routine

```

```

*****
*
*   This subroutine will save the ending address of the
*   program.  It will restart the program without
*   performing a BASIC CLR command.  If you are not in
*   the program mode, the routine will test to see if
*   you have entered the RUN "Filename" command and if
*   so, then the routine will exit with an RTS.  If
*   none of the aforementioned items hold true, the
*   routine will print the 'READY' prompt to the
*   screen, relink the program lines, and then jump to
*   the main loop which will place you into the direct
*   mode.
*
*****

```

```

916C: STX   $1210      ;Save the LSB and
916F: STY   $1211      ;The MSB of the end of the program address
9172: BIT   $7F        ;Check if we are in DIRECT mode
9174: BMI   $9184      ;If we are not in the DIRECT mode, then branch
9176: BVS   $915F      ;If this routine was called by RUN "Filename",
                        ;Then branch
9178: JSR   $4D2A      ;Print 'READY'
917B: JSR   $4F4F      ;Relink the BASIC PROGRAM lines
917E: JSR   $51F3      ;Reset the program pointer and perform a BASIC
                        ;CLR Command
9181: JMP   $4DC3      ;Exit to the input waiting Loop

```

```

*****
*
* This routine is used to terminate a LOAD/VERIFY
* command within a BASIC program by pointing TXTPTR
* to the start of the Basic Program and Relinking the
* Basic Text (line links). This is why you are
* advised to use the new DLOAD command.
*
*****

```

```

9184: JSR   $5254      ;Set TXTPTR to the start of BASIC text
9187: JSR   $4F4F      ;Relink the BASIC program lines
918A: JMP   $5235      ;Reset the stack pointer, perform restore, and
                        ;Reset the temporary string descriptor to $1B

```

```

*****
*
* BASIC OPEN COMMAND
*
* Command syntax: OPEN lfn,DV[,sa]
*
* NOTE: lfn = the logical file number
*        DV = the device number
*        sa = the secondary address
*
* This command allows you to access a file whose number*
* is specified by 'lfn' to the device whose number is *
* specified by 'DV'. This command is used to prepare *
* for writing to or reading from a file with BASIC *
* commands such as PRINT# and INPUT# respectively. *
*
*****

```

```

918D: JSR   $91F6      ;Set the parameters for OPEN
9190: CLC                   ;Clear the carry flag for error checking below

```



```

9191: JSR  $90D8      ;KERNAL open routine
9194: JSR  $9243      ;Set the flag to obtain a new DS$
9197: BCS  $91AB      ;If an error occurred, then branch
9199: RTS                ;If there were no errors, then exit the routine

```

```

*****
*
*          BASIC CLOSE COMMAND          *
*
* Command syntax:  CLOSE lfn          *
*
* NOTE:  lfn = the file number to be closed          *
*
* This command closes the file whose number is          *
* specified by 'lfn'.          *
*
*****

```

```

919A: JSR  $91F6      ;Set the parameters for CLOSE
919D: JSR  $A845      ;BANK 15
91A0: LDA  $4B        ;Get the logical file number
91A2: CLC                ;Clear the carry for error checking
91A3: JSR  $9275      ;BASIC KERNAL CLOSE routine
91A6: JSR  $9243      ;Set the flag to obtain a new DS$
91A9: BCC  $915F      ;If no errors occurred, then branch
91AB: JMP  $90D0      ;If an error has occurred (X=error number)

```

```

*****
*
*          SETPAR          *
*
*   SET the PARAMeters for LOAD, VERIFY and SAVE          *
*
* This routine Sets the FILENAME, DEVICE ADDRESS, and          *
* SECONDARY ADDRESS for the LOAD, VERIFY and SAVE          *
* commands. This routine will default to the          *
* cassette if a DEVICE ADDRESS is not specified.          *
*
*****

```

```

91AE: LDA  #0          ;Default to no FILE NAME
91B0: JSR  $925D      ;BANK 15, KERNAL SETNAM
91B3: LDX  #1          ;Default DEVICE ADDRESS = cassette drive
91B5: LDY  #0          ;Default SECONDARY ADDRESS - none
91B7: JSR  $9257      ;BANK 15, SETLFS
91BA: JSR  $9287      ;BANK 15, SETBNK

```

```
*****
*
* The next JSR will check to see if there are any
* more parameters with the BASIC LOAD command. If
* there are not any, then this routine will exit.
*
*****
```

```
91BD: JSR $91E3 ;Check for end of statement
91C0: JSR $9239 ;Get the string parameters of the filename and
;Call SETNAM
91C3: JSR $91E3 ;Check for end of statement
91C6: JSR $91DD ;Get the DEVICE NUMBER
91C9: LDY #0 ;No SECONDARY ADDRESS
91CB: STX $4B ;Save the DEVICE NUMBER
91CD: JSR $9257 ;BANK 15, SETLFS
91D0: JSR $91E3 ;Check for the end of statement
91D3: JSR $91DD ;Get the SECONDARY ADDRESS
91D6: TXA ;Place it in the accumulator
91D7: TAY ;And the Y register
91D8: LDX $4B ;Get the SECONDARY ADDRESS
91DA: JMP $9257 ;BANK 15,SETLFS
```

```
*****
*
* CHKCOM
*
* Check COMma
*
* This routine will skip the comma if one is found.
* The routine will then get the next character and
* place its numeric value in the X register. If the
* next character is not a comma when you call this
* routine then a 'SYNTAX' error will result.
*
*****
```

```
91DD: JSR $91EB ;Check for a ',' and skip it
91E0: JMP $87F4 ;Get the value into the X register
```

```
*****
*
* CHKEOS
*
* Check for End Of Statement
*
* This routine will check for the statement terminator *
```

```

* or the statement chain. If they are not found,      *
* the routine will return. However, if either the    *
* statement terminator or the statement chain are     *
* found, then the routine will pull the return address *
* off of the stack of the routine that called this   *
* routine. An RTS is then executed which will return *
* execution back to the calling routine.             *
*                                                     *
*****

```

```

91E3: JSR   $0386      ;Get the character after the command:If it is
91E6: BNE   $91EA      ;Not the end of the statement, then branch to
                        ;Process the parameters following the command
91E8: PLA                       ;Pull the return address of the
91E9: PLA                       ;Calling routine off the stack
91EA: RTS                      ;Exit back to the calling routine

```

```

*****
*                                                     *
*                               CHKCOM1                *
*                                                     *
*                               CHeck COMma1         *
*                                                     *
* This routine will check for a ',' and if it is     *
* found, the comma will be skipped by moving the    *
* TXTPTR to the next character past the command.   *
* If the comma was not found, then a 'SYNTAX' error *
* will be generated.                                *
*                                                     *
*****

```

```

91EB: JSR   $795C      ;Check for comma and skip it. If not
                        ;Found, then generate a 'SYNTAX' error

```

```

*****
*                                                     *
*                               CHKEND                *
*                                                     *
* This routine will CHeck for an END of statement flag *
* ($00), or ':'. If one of these characters are not *
* found, then a 'SYNTAX' error is generated.        *
*                                                     *
*****

```

```

91EE: JSR   $0386      ;Get the last character back
91F1: BNE   $91EA      ;If there are more entries, then branch
91F3: JMP   $796C      ;Generate a 'SYNTAX' error

```

```

*****
*
*                               SETPAR1
*
*                               SET PARAMeter1
*
* This routine will set the parameters for the BASIC
* OPEN, and CLOSE commands.
*
*****

```

```

91F6: LDA  #0           ;BANK number for LOAD/SAVE/VERIFY (LSV) commands
91F8: LDX  #1           ;BANK number for the current file name
91FA: JSR  $9287        ;BANK 15, SETBANK
91FD: JSR  $925D        ;BANK 15, SETNAME
9200: JSR  $91EE        ;Check for the end of line/statement
9203: JSR  $87F4        ;Get the logical file number
9206: STX  $4B          ;Save it
9208: TXA              ;Transfer the X register into the accumulator
9209: LDX  #1           ;DEFAULT address number (DEVICE)
920B: LDY  #0           ;DEFAULT SECONDARY ADDRESS
920D: JSR  $9257        ;BANK 15, SETLFS
9210: JSR  $91E3        ;Check for the end of statement
9213: JSR  $91DD        ;Skip the comma,put the value in the X register
9216: STX  $4C          ;Save it as the DEVICE number
9218: LDY  #0           ;DEFAULT SECONDARY ADDRESS
921A: LDA  $4B          ;Get the FILE NUMBER
921C: CPX  #3           ;DEVICE on the serial bus?
921E: BCC  $9221        ;If there is not, then branch (C=0)
9220: DEY              ;SECONDARY ADDRESS = $FF
9221: JSR  $9257        ;BANK 15, SETLFS
9224: JSR  $91E3        ;Check for the end of statement
9227: JSR  $91DD        ;Get the SECONDARY ADDRESS
922A: TXA              ;Move it into the accumulator
922B: TAY              ;Move it into the Y register for SETLFS
922C: LDX  $4C          ;Get the DEVICE ADDRESS
922E: LDA  $4B          ;Get the FILE NUMBER
9230: JSR  $9257        ;BANK 15, SETLFS
9233: JSR  $91E3        ;Check for the end of statement
9236: JSR  $91EB        ;Check for a comma and skip it

```

```

*****
*
*                               FRM
*
* This routine will evaluate and obtain the FILENAME
*

```

```
* memory location and length and then set the filename.*
*
*****
```

```
9239: JSR    $877B    ;Get the string parameters (address in locations
                    ;$24, $25 and the length in the accumulator)
923C: LDX    $24      ;Get the LSB of the address of the string
923E: LDY    $25      ;Get the MSB of the address of the string
                    :(also the length is in accumulator)
9240: JMP    $925D    ;BANK 15, SETNAM
```

```
*****
*
*                               SETDS$
*
*           SET the flag to obtain a new DS$
*
* This routine will check to see if the current
* device number in $BA is on the serial bus and if it
* is, then the routine will deallocate the current DS$
* and set the flag to obtain a new DS$.
*
*****
```

```
9243: PHP                      ;Save the processor status on the stack
9244: PHA                      ;Save the accumulator onto the stack
9245: LDA    $BA                ;Get the current device number
9247: CMP    #4                 ;Is it the serial bus?
9249: BCC    $924E              ;If it is not, then branch
924B: JSR    $A80D              ;Deallocate the current DS$ and set the flag to
                    ;Obtain a new one
924E: PLA                      ;Get the accumulator off of the stack
924F: PLP                      ;Get the processor status off of the stack
9250: RTS                      ;Exit the routine
```

```
*****
*
* The following JUMP table will allow you to enter
* the various KERNAL routines by first performing
* a BASIC BANK 15 command and then jumping to the
* KERNAL routine.
*
*****
```



```
*****
*
*                               BCHROUT (BBSOUT)
*
*                               Banked CHROUT (Banked BSOUT)
*
*****
```

```
9269: JSR  $A845      ;BANK 15
926C: JMP  $FFD2      ;CHROUT
```

```
*****
*
*                               BCLRCH
*
*                               Banked CLRCH
*
*****
```

```
926F: JSR  $A845      ;BANK 15
9272: JMP  $FFCC      ;CLRCH
```

```
*****
*
*                               BCLOSE
*
*                               Banked CLOSE
*
*****
```

```
9275: JSR  $A845      ;BANK 15
9278: JMP  $FFC3      ;CLOSE
```

```
*****
*
*                               BCLALL
*
*                               Banked CLose ALL
*
*****
```

```
927B: JSR  $A845      ;BANK 15
927E: JMP  $FFE7      ;CLALL
```

```

*****
*
*                               BPRIMM
*
*                               Banked PRIMM
*
*****
    
```

```

9281: JSR  $A845      ;BANK 15
9284: JMP  $FF7D      ;PRIMM
    
```

```

*****
*
*                               BSETBNK
*
*                               Banked SETBNK
*
*****
    
```

```

9287: JSR  $A845      ;BANK 15
928A: JMP  $FF68      ;SETBNK
    
```

```

*****
*
*                               BPLOT
*
*                               Banked PLOT
*
*****
    
```

```

928D: STA  $FF03      ;Enable BANK 14
9290: JMP  $FFF0      ;PLOT
    
```

```

*****
*
*                               BSTOP
*
*                               Banked STOP
*
*****
    
```

```

9293: JSR  $A845      ;BANK 15
9296: JMP  $FFE1      ;STOP
    
```



```

*****
*
*                               GETSPA
*
*                               GET SPace
*
* This routine will clear the garbage collection has
* been tried flag. It will then allocate the number
* of bytes requested in the accumulator plus two bytes
* which will be used by the calling routine to save
* the address of the string's descriptor. The routine
* will temporarily place the flag (#$FF) in the MSB
* and the length of the string in the LSB of that
* address. (FRETOP = FRETOP - length of string - 2).
* If the number of bytes needed are not available as
* free memory, a garbage collection is performed.
* If there is still not enough room, an
* 'OUT OF MEMORY ' error will result.
*
*****

```

```

9299: LSR   $11      ;Clear the garbage collection tried flag
929B: TAX                      ;Move the length of the string to the X register
929C: BEQ   $92D9    ;If the length equals zero, then exit
929E: PHA                      ;Save the length of the string onto the stack
929F: LDA   $35      ;Get the pointers for FRETOP
92A1: SEC                      ;Then subtract 2 to make room
92A2: SBC   #$02     ;For the pointer to the descriptor that goes
                      ;After the string
92A4: LDY   $36      ;Get the MSB
92A6: BCS   $92A9    ;If there is not an overflow, then branch
92A8: DEY                      ;Decrement the MSB by 1
92A9: STA   $24      ;Save the new FRETOP
92AB: STY   $25      ;Value in locations $24, $25
92AD: TXA                      ;Restore the length of the string
92AE: EOR   #$FF     ;Subtract the length of
92B0: SEC                      ;The string from
92B1: ADC   $24      ;FRETOP
92B3: BCS   $92B6    ;If there is not an overflow, then branch
92B5: DEY                      ;Decrement the MSB by 1
92B6: CPY   $34      ;Check for conflict with VARTAB
92B8: BCC   $92DA    ;To see if we are
92BA: BNE   $92C0    ;Intruding into the
92BC: CMP   $33      ;ARRAY storage area
92BE: BCC   $92DA    ;If an intrusion has occurred, then branch
92C0: STA   $37      ;Save the address to where
92C2: STY   $38      ;The string is to be stored

```



```

* string descriptor that is stored starting at $0402 *
* and moves upward toward $FF00. Each time a string *
* is changed in anyway or another string is added to *
* or deleted from memory, this routine is called. *
* What this routine will do is start at $FEFF *
* (which contains the MSB of the address which points *
* to the string's descriptor) and check to see if it *
* contains a $FF (which is the flag to inform this *
* routine that this string is no longer needed or *
* used in memory). If it does not contain a $FF, the *
* routine will subtract the length of the string + 2 *
* from this address which will place it at the next *
* string downward in the memory's descriptor pointer. *
* Let's say, for example, that the MSB of this *
* descriptor pointer contains a $FF. Then this *
* routine will move the next string upward into *
* this location (as long as its descriptor pointer *
* does not contain a $FF as the MSB). Then the *
* routine will update the 'good' string's variable *
* descriptor to the string's new address in memory. *
* This process continues until all the 'good' strings *
* have been moved upward toward $FF00 thereby deleting *
* all of the 'unused' strings. *
* *
*****

```

```

92EA: LDX $18 ;Get the pointer to the next open string stack
92EC: CPX #$1B ;Is the temporary string stack empty?
92EE: BEQ $9303 ;If it is, then branch to
92F0: JSR $93F0 ;Get the lowest string address
92F3: BEQ $92EC ;If the length of the string is 0, then branch
92F5: TXA ;Move the LSB of the temporary string descriptor
;Into the accumulator
92F6: LDY #0 ;Zero the index register
92F8: STA $FF04 ;Enable BANK 14 with RAM BANK 1 enabled
92FB: STA ($5E),Y ;Save the LSB of the temporary string descriptor
92FD: TYA ;Move the MSB of the strings to the accumulator
;(Zero)
92FE: INY ;Increment the index pointer
92FF: STA ($5E),Y ;Save the MSB of the descriptor address after
;The string
9301: BNE $92EC ;Fall through
9303: LDY #0
9305: STY $5A
9307: LDX $39 ;Get the highest memory available to
9309: LDY $3A ;Strings, normally ($FF00)
930B: STX $61 ;And save it in the GRBPNT ($50, $51),

```

---

```

930D: STX $50 ;And $61, $62
930F: STX $37
9311: STY $62 ;And $37, $38
9313: STY $51
9315: STY $38
9317: TXA ;Move the LSB of the MAXMEM1 value into the
;Accumulator
9318: JSR $9383 ;And subtract it from $50, $51 (GRBPNT)
931B: BNE $9329 ;If an overflow occurred, then branch
931D: DEY ;Decrement the index pointer to point to the
;Length of the string
931E: JSR $42FB ;Get the length of the string in the accumulator
;Then subtract the length from GRBPNT to move it
;to the 1st
9321: JSR $93D2 ;First character of the string to FREEUP
9324: SEC ;Set the carry flag
9325: ROR $5A ;Then rotate it into Bit 7 (set Bit 7)
9327: BNE $9318
9329: BIT $5A
932B: BPL $936F
932D: LDX #0
932F: STX $5A
9331: LDA #$02
9333: LDY #$01 ;Index over to the MSB of the string's
9335: JSR $42FB ;Descriptor and place it in the
;Accumulator
9338: STA ($61),Y ;And save it after the string
933A: DEY
933B: JSR $42FB ;Get the LSB of the string's descriptor
933E: STA ($61),Y ;And save it after the string
9340: JSR $03B7 ;Get the length of the string whose address is
9343: TAX ;In locations $24, $25 and place it into the X
;register
9344: JSR $93E1 ;Subtract the length of the string from the
;String's address
9347: STA $37 ;Save it
9349: STY $38 ;In FRESPEC
934B: TXA ;Move the length back into the accumulator
934C: JSR $93D2 ;Subtract it from the old string's address so
;The index pointer points to the first character
;After the string to be moved
934F: TXA ;Move the length into the accumulator
9350: TAY ;Move the length into the Y register to be used
;As an index pointer
9351: DEY ;Decrement the index pointer
9352: JSR $42FB ;Get a character from the string to be moved up
;In memory

```

```

9355: STA    ($61),Y    ;Then place it into the new address
9357: DEX                ;Decrement the other index pointer
9358: BNE    $9351      ;If not done, branch to continue
935A: LDY    #$02       ;Move a total of 2 bytes
935C: LDA    $0060,Y    ;Move the address of the string's new
935F: STA    ($24),Y    ;Location into the string's variable
9361: DEY                ;Descriptor which is pointed to by
9362: BNE    $935C      ;Locations $24, $25
9364: LDA    $50        ;Get the address of the character which is
9366: LDY    $51        ;Just prior to the string in its new location
9368: JSR    $9383      ;Test for anymore strings to relocate
936B: BEQ    $931D      ;If the next string is to be deallocated, branch
936D: BNE    $9333      ;If not, then continue looping until the 'good'
                        ;Strings are shifted upward
936F: LDY    #0         ;Zero the index register
9371: JSR    $03B7      ;Get the length of the string in the accumulator
9374: TAX                ;Then move the string length to the X register
9375: JSR    $93E1      ;Subtract the string length from the LSB of the
                        ;String descriptor (located after the string)
                        ;Which will place the address pointing to the
                        ;First character of the string
9378: STA    $37        ;Save the address of
937A: STY    $38        ;The first character of the string
937C: TXA                ;Restore the string length to the accumulator
937D: JSR    $93D2      ;NOTE: See the comments at $9375
9380: JMP    $9318      ;Continue searching for strings to delete
9383: CPY    $36        ;Check to see if the address in the accumulator
9385: BCC    $93B1      ;And the Y register are the same as FRETOP
9387: BNE    $938F      ;If they are equal to or less than, then branch
9389: CMP    $35        ;To check to see if the address is pointing
938B: BEQ    $93B1      ;To the temporary string stack
938D: BCC    $93B1
938F: BIT    $5A        ;Check for a string transfer is in progress
9391: BMI    $9398      ;If one is in progress, then branch
9393: LDA    #$02       ;Subtract 2 from $50, $51
9395: JSR    $93E1      ;GRBPNT-point to the descriptor after the string
9398: LDA    #$02       ;Subtract 2 from $61, $62
939A: JSR    $93D2      ;GRBTOP-point to the next descriptor after the
                        ;String
939D: LDY    #$01      ;Get the MSB of the descriptor after the string
939F: JSR    $42FB      ;To check it to see
93A2: CMP    #$FF      ;If it is the flag for 'string deallocated'
93A4: BNE    $93A7      ;If it is not, then branch
93A6: RTS                ;If it is, then return

```

```

*****
*
*                               MOVDES
*
*                               MOVE DEScriptor
*
* Move the address of the string's descriptor which
* is the two bytes after the string into $24, $25.
* On exiting from this subroutine locations $24, $25
* will contain the address of the string descriptor.
*
*****

93A7: JSR   $42FB      ;Move the address of this
93AA: STA   $0024,Y    ;String's descriptor
93AD: DEY                       ;Into locations $24, $25
93AE: BPL   $93A7      ;In LSB, MSB format
93B0: RTS                       ;And exit the routine
93B1: LDX   $18        ;Get the index pointer to the next available
                               ;Temporary string descriptor
93B3: CPX   #$1B      ;If the stack is empty,
93B5: BEQ   $93C7      ;Then branch to find the highest string address
93B7: JSR   $93F0      ;Deallocate the last temporary string stack used
93BA: BEQ   $93B3      ;If the length of the string was zero, try again
93BC: LDY   #0         ;Save the length of the string
93BE: STA   ($5E),Y    ;In the LSB of the descriptor's address after
                               ;The string
93C0: INY                       ;Increment the index pointer
93C1: LDA   #$FF      ;Place the flag for string is free after
93C3: STA   ($5E),Y    ;The string
93C5: BNE   $93B3      ;Unconditional branch
93C7: PLA                       ;Pull the return address off of the
93C8: PLA                       ;Stack of the routine that called this one
93C9: LDA   $37        ;Make FRETOP
93CB: LDY   $38        ;Equal to
93CD: STA   $35        ;FRESPEC
93CF: STY   $36
93D1: RTS                       ;Exit the routine

*****
*
* Subtract the value in the accumulator from $50, $51
* (GRBPNT).
*
*****

93D2: EOR   #$FF      ;Invert the value

```

```

93D4: SEC                ;Set the carry flag
93D5: ADC    $50        ;Subtract the value in the accumulator from $50
93D7: LDY    $51        ;Get the MSB of the address
93D9: BCS    $93DC      ;If there was no overflow, then branch
93DB: DEY                ;Decrement the MSB by 1
93DC: STA    $50        ;Save the LSB
93DE: STY    $51        ;And the MSB
93E0: RTS                ;Exit the routine
    
```

```

*****
*
* Subtract the value in the accumulator from $61, $62
* (GRBTOP).
*
*****
    
```

```

93E1: EOR    #$FF       ;Invert the value
93E3: SEC                ;Set the carry flag
93E4: ADC    $61        ;Subtract the value in the accumulator from $61
93E6: LDY    $62        ;Get the MSB of the address
93E8: BCS    $93EB      ;If there was no overflow, then branch
93EA: DEY                ;Decrement the MSB by one
93EB: STA    $61        ;Save the LSB
93ED: STY    $62        ;And the MSB
93EF: RTS                ;Exit the routine
    
```

```

*****
*
* This routine subtracts one from the pointers to the
* next available temporary string stack to point to
* the MSB of the string's address in the no longer
* used string descriptor. Locations $5E, $5F will
* contain the address of the deallocated string and
* the accumulator will contain the number of bytes
* deallocated (length of the string). The address in
* locations $5E, $5F will be pointing to the LSB of
* the descriptor.
*
*****
    
```

```

93F0: DEX                ;Decrement the address of the temp. string stack
93F1: LDA    $00,X       ;Get the MSB of the strings address
                        ;From the string stack
93F3: STA    $5F        ;Save the MSB of the string's address
93F5: DEX                ;Decrement the index pointer
93F6: LDA    $00,X       ;Get the LSB of the strings address
                        ;From the string stack
    
```

```

93F8: STA   $5E           ;Save the LSB of the string's address
93FA: DEX                       ;Decrement the index pointer
93FB: LDA   $00,X         ;Get the length of the string from the
                        ;Temp. string stack
93FD: PHA                       ;Save the length of the string onto the
93FE: CLC                       ;Stack and add the length of the
93FF: ADC   $5E           ;String to the address of the
9401: STA   $5E           ;String to free
9403: BCC   $9407         ;That string's space
9405: INC   $5F           ;Up in RAM
9407: PLA                       ;Pull the length of the string (# of Bytes
                        ;Deallocated)
9408: RTS                       ;And exit the routine

```

```

*****
*
*           BASIC COS FUNCTION
*
* Function syntax:  COS (X)
*
* This function will return a value which represents
* the cosine of 'X' where 'X' is an angle that is
* measured in radians.
*
*****

```

```

9409: LDA   #$85           ;Pointer to  $\pi/2$ 
940B: LDY   #$94
940D: JSR   $8A12         ;Add  $\pi/2$  to FAC1

```

```

*****
*
*           BASIC SIN FUNCTION
*
* Function syntax:  SIN (X)
*
* This function will return a value which represents
* the sine of 'X' where 'X' is an angle that is
* measured in radians.
*
*****

```

```

9410: JSR   $8C38         ;Round FAC1 and move it to FAC2
9413: LDA   #$8A           ;Address of
9415: LDY   #$94           ; $\pi * 2$ 
9417: LDX   $6F           ;Get the sign of FAC2
9419: JSR   $8B41         ;Divide FAC2 by ( $\pi*2$ )

```



```

941C: JSR   $8C38      ;Round FAC1 and place the result in FAC2
941F: JSR   $8CFB      ;Convert FAC1 to integer format
9422: LDA   #0         ;Clear the sign
9424: STA   $70        ;Comparison flag
9426: JSR   $8831      ;Subtract FAC2 from FAC1
9429: LDA   #$8F       ;Address of the
942B: LDY   #$94       ;Constant .25
942D: JSR   $8A18      ;Subtract .25 from FAC1
9430: LDA   $68        ;Get the sign of FAC1
9432: PHA                ;Save it onto the stack
9433: BPL   $9442      ;If it is positive, then branch
9435: JSR   $8A0E      ;Round FAC1 by adding .5
9438: LDA   $68        ;Get the sign of FAC1
943A: BMI   $9445      ;If it is negative, then branch
943C: LDA   $14        ;Get the flag for TAN
943E: EOR   #$FF       ;Invert it
9440: STA   $14        ;And save it back
9442: JSR   $8FFA      ;Invert the sign of FAC1
9445: LDA   #$8F       ;Address of the
9447: LDY   #$94       ;Constant .25
9449: JSR   $8A12      ;Add .25 to FAC1
944C: PLA                ;Get the sign of FAC1 off the stack
944D: BPL   $9452      ;If the value is positive, then branch
944F: JSR   $8FFA      ;Make the sign positive (+)
9452: LDA   #$94       ;Address of the
9454: LDY   #$94       ;Polynomial degree
9456: JMP   $9086      ;Calculate polynomial of FAC1

```

```

*****
*
*          BASIC TAN FUNCTION
*
* Function syntax: TAN (X)
*
* This function will return a value which represents
* the tangent of 'X' where 'X' is an angle that is
* measured in radians.
*
*****

```

```

9459: JSR   $8BFC      ;Move FAC1 to locations $59, $5E
945C: LDA   #0         ;Clear the flag
945E: STA   $14        ;For TAN
9460: JSR   $9410      ;Calculate the SINE OF FAC1
9463: LDX   #$50       ;Pointer to the temporary work area
9465: LDY   #0         ;Located at ($0050)
9467: JSR   $8C00      ;Move FAC1 To location $50

```

```

946A: LDA    #59          ;Pointer to the temporary work area
946C: LDY    #0           ;Located at ($0059)
946E: JSR    $8BD4        ;Restore the original value to FAC1
9471: LDA    #0           ;Make the sign of
9473: STA    $68          ;FAC1 positive (+)
9475: LDA    $14          ;Get the flag for TAN
9477: JSR    $9481        ;And perform a COSINE function
947A: LDA    #50          ;Pointer to the temporary work area
947C: LDY    #0           ;Located at ($50)
947E: JMP    $8B49        ;Move the sign of FAC1 back to FAC1
9481: PHA                    ;Save the sign of FAC1 onto the stack
9482: JMP    $9442        ;Calculate the COSINE of FAC1
    
```

```

*****
*
*           Constant for SIN and COS
*
*****
    
```

	EX	M1	M2	M3	M4		
	--	--	--	--	--		
9485:	.BYTE	\$81,\$49,\$0F,\$DA,\$A2	;	1.57079633	$\pi/2$		
948A:	.BYTE	\$83,\$49,\$0F,\$DA,\$A2	;	6.28318531	$\pi*2$		
948F:	.BYTE	\$7F,\$00,\$00,\$00,\$00	;	.25			
9494:	.BYTE	\$05	;	5	POLYNOMIAL DEGREE		
9495:	.BYTE	\$B4,\$EA,\$1A,\$2D,\$1B	;	-14.3813907			
949A:	.BYTE	\$86,\$28,\$07,\$FB,\$F8	;	42.0077971			
949F:	.BYTE	\$87,\$99,\$68,\$89,\$01	;	-76.7041703			
94A4:	.BYTE	\$87,\$23,\$35,\$DF,\$E1	;	81.6052237			
94A9:	.BYTE	\$86,\$A5,\$5D,\$E7,\$28	;	-41.3147021			
94AE:	.BYTE	\$83,\$49,\$0F,\$DA,\$A2	;	6.28318531	$\pi * 2$		

```

*****
*
*           BASIC ATN FUNCTION
*
* Function syntax:  ATN (X)
*
* This function will return a value which represents
* the angle in radians whose tangent is 'X'.
*
*****
    
```

```

94B3: LDA    $68          ;Get the SIGN of FAC1
94B5: PHA                    ;Save it on the stack
94B6: BPL    $94BB        ;If the sign is positive, then branch
94B8: JSR    $8FFA        ;Invert the sign of FAC1
    
```

```

94BB: LDA $63 ;Get the exponent of FAC1
94BD: PHA ;And save it onto the stack for recall later
94BE: CMP #$81 ;If the value of FAC1 is
94C0: BCC $94C9 ;Less than one, then branch
94C2: LDA #$9C ;Address of the
94C4: LDY #$89 ;Constant 1 in the log tables
94C6: JSR $8A1E ;Divide 1 by FAC1
94C9: LDA #$E3 ;Pointer to constant for polynomial degree
94CB: LDY #$94 ;Calculate polynomial
94CD: JSR $9086 ;And leave it in FAC1
94D0: PLA ;Get the exponent of FAC1 back
94D1: CMP #$81 ;See if the number was less than one
94D3: BCC $94DC ;If it was less than one, then branch
94D5: LDA #$85 ;Address of
94D7: LDY #$94 ;PI / 2
94D9: JSR $8A18 ;Subtract FAC1 from PI / 2
94DC: PLA ;Get the sign of FAC1 back
94DD: BPL $94E2 ;If it is positive, then branch to exit
94DF: JMP $8FFA ;Invert the sign of FAC1
94E2: RTS ;Exit the routine

```

```

*****
*
*          FLOATING POINT CONSTANTS FOR ATN          *
*
*****

```

```

          EX M1 M2 M3 M4
          -- -- -- -- --
94E3: .BYTE $81,$38,$AA,$3B,$29 ; 1.44269504 = 1/LOG(2)
94E8: .BYTE $0B ; 11 = POLYNOMIAL DEGREE
94E9: .BYTE $76,$B3,$B3,$BD,$D3 ; -6.84793912E-4
94EE: .BYTE $79,$1E,$F4,$A6,$F5 ; 4/85094216E-3
94F3: .BYTE $7B,$8E,$FC,$B0,$10 ; - .0161117015
94F8: .BYTE $7C,$0C,$1F,$67,$CA ; .034209638
94FD: .BYTE $7C,$DE,$53,$CB,$C1 ; - .054279133
9502: .BYTE $7D,$14,$64,$70,$4C ; .0724571965
9507: .BYTE $7D,$B7,$EA,$51,$7A ; - .0898019185
950C: .BYTE $7D,$63,$30,$88,$7E ; .110932413
9511: .BYTE $7E,$92,$44,$99,$3A ; - .142839808
9516: .BYTE $7E,$4C,$CC,$91,$C7 ; .19999912
951B: .BYTE $81,$00,$00,$00,$00 ; 1

```

```

*****
*
*                               BASIC PRINT USING
*
* Command syntax: PRINT [#fn,] USING "xx"; pl
*
* NOTE:  fn = the file number for a PRINT#fn USING
*        xx = the format list
*        pl = the print list
*
* This function is used to define the format of
* string and numeric characters that are printed to
* the screen, printer, or other device.
*
* This routine is called by the BASIC PRINT COMMAND
* and not by the BASIC INTERPRETER.
*
* On entry to this routine, TXTPTR is pointing to the
* first character before the opening quote mark in the
* FORMAT LIST.
*
* On exit from this routine, TXTPTR is pointing to the
* first character after the PRINT LIST, and you are
* returned to BANK 15.
*
*****

```

```

9520: LDX  #$FF      ;Set the pointer to the
9522: STX  $0136     ;The end of the field
9525: JSR  $0380     ;Get the next character after the USING token
9528: JSR  $77EF     ;Evaluate the string (Format List) and place the
                    ;Address of the temporary descriptor in $66, $67
                    ;(INDICE)
952B: JSR  $77DD     ;Make sure it is a string and if not,
                    ;Then generate a 'TYPE MISMATCH' error
952E: LDA  $66       ;Save the address of
9530: PHA                ;Which temporary descriptor
9531: LDA  $67       ;Is being
9533: PHA                ;Used onto the stack (LSB,MSB)
9534: LDY  #$02      ;Move the strings
9536: JSR  $42E7     ;Descriptor out of the
9539: DEY                ;Temporary string stack and place
953A: STA  $003F,Y   ;It into FORM (FORMAT LIST) which will
953D: BNE  $9536     ;Be the string's address in LSB, MSB format
953F: JSR  $42E7     ;Get the length of the format list
9542: STA  $0135     ;And save it in LFOR
9545: TAY                ;Move the length into the accumulator

```

---

```

9546: BEQ   $9553   ;If the length is zero (Null String), then
                    ;Branch to generate a 'SYNTAX' error
9548: DEY                   ;Subtract one from the length to create an Index
                    ;Into the string
9549: JSR   $42D3   ;Get a character from the format list
954C: CMP   #'#'     ;Is it the flag to reserve space for a character
                    ;In the print list?
954E: BEQ   $9556   ;If it is, then branch
9550: TYA                   ;Move the string length -1 to the accumulator
9551: BNE   $9548   ;If not done, then keep checking
9553: JMP   $796C   ;Generate a 'SYNTAX' error
9556: LDA   #';'     ;Check for a semicolon after the format list
                    ;And if there is
9558: JSR   $795E   ;Not, then generate a 'SYNTAX' error. If there
                    ;Is, get the first character after the semicolon
955B: STY   $77     ;Zero ZPTMPI
955D: STY   $0123   ;Zero BNR
9560: JSR   $77EF   ;Evaluate the expression after the semicolon
                    ;(the Print List)
9563: BIT   $0F     ;Check to see if the VALTYP was string ($FF) or
                    ;Numeric ($00)
9565: BPL   $95A0   ;If it is numeric, then branch
9567: JSR   $979F   ;Deallocate the format list and set flag to
                    ;Indicate the Print List will be a string
956A: JSR   $98F2
956D: LDX   $012B   ;Get the flag for centering '=' and
                    ;right justify '>'
9570: BEQ   $9587   ;If none were specified, then branch
9572: LDX   #0
9574: SEC
9575: LDA   $0131
9578: SBC   $78
957A: BCC   $9587
957C: LDX   #'='     ;Check if the centering character
957E: CPX   $012B   ;Was specified
9581: BNE   $9586   ;If it was not, then branch
9583: LSR
9584: ADC   #0
9586: TAX
9587: LDY   #0
9589: TXA
958A: BEQ   $9591
958C: DEX
958D: LDA   #$20
958F: BNE   $9599
9591: CPY   $78
9593: BCS   $958D

```

---

```

9595: JSR   $03B7
9598: INY
9599: JSR   $98EB
959C: BNE   $9589
959E: BEQ   $95C7
95A0: JSR   $8E42      ;Convert the numeric value to a string at $0100
95A3: LDY   #$FF
95A5: INY
95A6: LDA   $0100,Y    ;Get a character of the string
95A9: BNE   $95A5      ;If it is not the end of the string, then
                        ;Continue getting the characters
95AB: TYA
                        ;Move the string length to the accumulator
95AC: JSR   $8690      ;Create a space in memory for the string
95AF: LDY   #0        ;Clear the index
95B1: STA   $FF04      ;Enable BANK 14 with RAM BANK 1
95B4: LDA   $0100,Y    ;Get a byte of the string
95B7: BEQ   $95BE      ;If we are at the end, then branch
95B9: STA   ($64),Y    ;Transfer the string
95BB: INY
                        ;From $0100
95BC: BNE   $95B4      ;To its area in RAM BANK 1
95BE: JSR   $86E3      ;Move the string descriptor to the string stack
95C1: JSR   $979F
95C4: JSR   $95E7
95C7: JSR   $0386      ;Get the character after the expression
95CA: CMP   #','      ;Is it a comma?
95CC: BEQ   $9558      ;If so, branch to evaluate the next expression
95CE: SEC
95CF: ROR   $77
95D1: JSR   $98F2
95D4: PLA
95D5: TAY
95D6: PLA
95D7: JSR   $8785
95DA: JSR   $0386
95DD: CMP   #','
95DF: BEQ   $95E4
95E1: JMP   $5598
95E4: JMP   $0380
95E7: STA   $FF03      ;Enable BASIC BANK 14
95EA: LDA   $1204
95ED: STA   $0133
95F0: LDA   #$FF
95F2: STA   $0132
95F5: JMP   $95FA
95F8: STX   $80
95FA: CPY   $78
95FC: BEQ   $9631

```

---

```
95FE: LDA $0100,Y
9601: INY
9602: CMP #' ' ;Is it a space?
9604: BEQ $95FA ;If it is, then branch
9606: CMP #'-' ;Is it a negative sign?
9608: BEQ $95F2 ;If it is, then branch
960A: CMP #'.' ;Is it a decimal point?
960C: BEQ $95F8 ;If it is, then branch
960E: CMP #'e' ;Is it an 'e' for scientific notation?
9610: BEQ $9623 ;If it is, then branch
9612: STA $0100,X
9615: STX $0124
9618: INX
9619: BIT $80
961B: BPL $95FA
961D: INC $012A
9620: JMP $95FA
9623: LDA $0100,Y
9626: CMP #'-'
9628: BNE $962D
962A: ROR $0128
962D: INY
962E: STY $0129
9631: LDA $80
9633: BPL $9637
9635: STX $80
9637: JSR $98F2
963A: LDA $012C
963D: CMP #FF
963F: BEQ $966A
9641: LDA $012F
9644: BEQ $9685
9646: LDA $0129
9649: BNE $965D
964B: LDX $0124
964E: JSR $9774
9651: DEC $0102,X
9654: INX
9655: STX $0129
9658: JSR $97FB
965B: BEQ $9682
965D: LDY $012E
9660: BNE $9679
9662: LDY $0132
9665: BMI $9679
9667: LDA $012C
966A: BEQ $96D6
```

966C: DEC \$012C  
966F: BNE \$9676  
9671: LDA \$012D  
9674: BEQ \$96D6  
9676: INC \$0127  
9679: JSR \$96EE  
967C: JSR \$97B9  
967F: JSR \$96EE  
9682: JMP \$981C  
9685: LDY \$0129  
9688: BEQ \$96A0  
968A: STA \$78  
968C: SEC  
968D: ROR \$0130  
9690: LDY \$80  
9692: LDA \$0128  
9695: BPL \$969D  
9697: JSR \$9727  
969A: JMP \$96A9  
969D: JSR \$9708  
96A0: LDY \$80  
96A2: BEQ \$96A9  
96A4: JSR \$97FF  
96A7: BEQ \$96AF  
96A9: JSR \$97B9  
96AC: JMP \$96B2  
96AF: DEC \$012A  
96B2: SEC  
96B3: LDA \$012C  
96B6: SBC \$012A  
96B9: BCC \$96D6  
96BB: STA \$0127  
96BE: LDY \$012E  
96C1: BNE \$96DE  
96C3: LDY \$0132  
96C6: BMI \$96DE  
96C8: TAY  
96C9: BEQ \$96D6  
96CB: DEY  
96CC: BNE \$96E1  
96CE: LDA \$012D  
96D1: ORA \$012A  
96D4: BNE \$9682  
96D6: LDA #\$2A  
96D8: JSR \$98EB  
96DB: BNE \$96D8  
96DD: RTS



96DE: TAY  
96DF: BEQ \$9682  
96E1: LDA \$012A  
96E4: BNE \$9682  
96E6: DEC \$0127  
96E9: INC \$77  
96EB: JMP \$9682  
96EE: SEC  
96EF: LDA \$012C  
96F2: SBC \$012A  
96F5: BEQ \$9730  
96F7: LDY \$80  
96F9: BCC \$9711  
96FB: STA \$78  
96FD: CPY \$0124  
9700: BEQ \$9704  
9702: BCS \$9705  
9704: INY  
9705: INC \$012A  
9708: JSR \$973D  
970B: DEC \$78  
970D: BNE \$96FD  
970F: BEQ \$972E  
9711: EOR #\$\$FF  
9713: ADC #\$\$01  
9715: STA \$78  
9717: CPY \$0123  
971A: BEQ \$9723  
971C: DEY  
971D: DEC \$012A  
9720: JMP \$9725  
9723: INC \$77  
9725: LDA #\$\$80  
9727: JSR \$973F  
972A: DEC \$78  
972C: BNE \$9717  
972E: STY \$80  
9730: RTS  
9731: BNE \$976C  
9733: EOR #\$\$09  
9735: STA \$0100,X  
9738: DEX  
9739: CPX \$0129  
973C: RTS  
973D: LDA #0  
973F: LDX \$0129  
9742: INX

```

9743: BIT    $0130
9746: BMI    $9758
9748: EOR    $0128
974B: BEQ    $9758
974D: JSR    $9782
9750: JSR    $9731
9753: BCS    $974D
9755: JMP    $895D
9758: LDA    $0100,X
975B: DEC    $0100,X
975E: CMP    #$30
9760: JSR    $9731
9763: BCS    $9758
9765: BIT    $0130
9768: BPL    $976F
976A: STY    $80
976C: PLA
976D: PLA
976E: RTS
976F: LDA    $0128    ;Get the sign of the exponent flag
9772: EOR    #$80     ;Condition Bit 7 which is the sign (1 =
                    ;Negative)
9774: STA    $0128    ;Save it back
9777: LDA    #'0'
9779: STA    $0101,X
977C: LDA    #'1'
977E: STA    $0102,X
9781: RTS                ;Exit the routine
9782: LDA    $0100,X
9785: INC    $0100,X
9788: CMP    #$39
978A: RTS

```

```

*****
*
* This subroutine will test the value that is in the
* Y register to see if it is $FF. If it is, then
* the routine will get the number of leading zeros
* in the Y register. The routine will then pull
* the return address of the calling routine and exit.
* If the value is not $FF, the value is compared
* against the length of the format text. If the value
* is less than the length of the format text, then
* the routine will get the next character from the
* format text, increment CFORM and exit.
*
*****

```

```

978B: CLC                ;Clear the carry flag
978C: INY                ;Increment the ENDFD
978D: BEQ $9794         ;And if it was $FF, then branch
978F: CPY $0135
9792: BCC $9798
9794: LDY $77
9796: BNE $976C         ;Branch to pull the address of the calling
                        ;Routine off the stack-return to the main loop
9798: JSR $42D3         ;Get a character from the Format List
979B: INC $0131         ;Increment the counter to the next character to
                        ;Be read in the Format List
979E: RTS                ;Exit the routine

```

```

*****
*
* This subroutine is used to zero out the Print
* USING's WORK AREA and deallocate the last string
* that was accessed by the Print Using command by
* calling FRESTR, to deallocate the print list.
*
*****

```

```

979F: JSR $8781         ;Deallocate the string whose address is in $66,
                        ;$67
97A2: STA $78           ;Save the length of the string in HULP
97A4: LDX #$0A         ;Zero out the PRINT USING
97A6: LDA #0           ;Work area from $0127 to $0131
97A8: STA $0127,X
97AB: DEX
97AC: BPL $97A8
97AE: STX $0126         ;Set FLAG to isignal the format list is a string
97B1: STX $80           ;Set POINT to signal that no decimal point was
                        ;Used in the format list
97B3: STX $0125         ;Set DOLR to indicate the Format List does not
                        ;Have a floating dollar sign ($)
97B6: TAX
97B7: TAY
97B8: RTS                ;Exit the routine

```

97B9: CLC  
97BA: LDA \$80  
97BC: ADC \$012D  
97BF: BCS \$97FA  
97C1: SEC  
97C2: SBC \$77  
97C4: BCC \$97FA  
97C6: CMP \$0124  
97C9: BEQ \$97CD  
97CB: BCS \$97FA  
97CD: CMP \$0123  
97D0: BCC \$97FA  
97D2: TAX  
97D3: LDA \$0100,X  
97D6: CMP #\$35  
97D8: BCC \$97FA  
97DA: CPX \$0123  
97DD: BEQ \$97E9  
97DF: DEX  
97E0: JSR \$9782  
97E3: STX \$0124  
97E6: BEQ \$97DA  
97E8: RTS  
97E9: LDA #\$31  
97EB: STA \$0100,X  
97EE: INX  
97EF: STX \$80  
97F1: DEC \$77  
97F3: BPL \$97FA  
97F5: INC \$77  
97F7: INC \$012A  
97FA: RTS  
97FB: LDY \$80  
97FD: BEQ \$9816  
97FF: LDY \$0123  
9802: LDA \$0100,Y  
9805: CMP #\$30  
9807: RTS  
9808: INC \$80  
980A: JSR \$973D  
980D: INC \$0123  
9810: CPY \$0124  
9813: BEQ \$97FA  
9815: INY  
9816: JSR \$9802  
9819: BEQ \$9808  
981B: RTS

981C: LDA \$0125  
981F: BMI \$9823  
9821: INC \$77  
9823: LDX \$0123  
9826: DEX  
9827: LDY \$0134  
982A: JSR \$42D3  
982D: INY  
982E: CMP # \$2C  
9830: BNE \$9846  
9832: BIT \$0126  
9835: BMI \$9840  
9837: STA \$FF03  
983A: LDA \$1205  
983D: JMP \$98AB  
9840: LDA \$0133  
9843: JMP \$98AB  
9846: CMP # \$2E  
9848: BNE \$9853  
984A: STA \$FF03  
984D: LDA \$1206  
9850: JMP \$98AB  
9853: CMP #'+'  
9855: BEQ \$9892  
9857: CMP #'-'  
9859: BEQ \$988D  
985B: CMP #'^'  
985D: BNE \$98C8  
985F: LDA # \$45  
9861: JSR \$98EB  
9864: LDY \$0129  
9867: JSR \$9802  
986A: BNE \$9872  
986C: INY  
986D: JSR \$9802  
9870: BEQ \$9879  
9872: LDA # \$2D  
9874: BIT \$0128  
9877: BMI \$987B  
9879: LDA # \$2B  
987B: JSR \$98EB  
987E: LDX \$0129  
9881: LDA \$0100,X  
9884: JSR \$98EB  
9887: LDY \$0136  
988A: JMP \$98A1  
988D: LDA \$0132

```
9890: BMI    $9840
9892: LDA    $0132
9895: JMP    $98AB
9898: LDA    $77
989A: BNE    $98B4
989C: CPX    $0124
989F: BEQ    $98A6
98A1: INX
98A2: LDA    $0100,X
98A5: .BYTE $2C
98A6: LDA    #48
98A8: LSR    $0126
98AB: JSR    $98EB
98AE: BEQ    $98B3
98B0: JMP    $982A
98B3: RTS
98B4: DEC    $77
98B6: LDA    $0125
98B9: BMI    $98A6
98BB: SEC
98BC: ROR    $0125
98BF: STA    $FF03
98C2: LDA    $1207
98C5: JMP    $98A8
98C8: LDA    $0127
98CB: BEQ    $9898
98CD: DEC    $0127
98D0: BEQ    $98D5
98D2: JMP    $9840
98D5: LDA    $012E
98D8: BMI    $98D0
98DA: JSR    $42D3
98DD: CMP    #', '
98DF: BNE    $988D
98E1: LDA    $0133
98E4: JSR    $98EB
98E7: INY
98E8: JMP    $98DA
98EB: JSR    $560C    ;Print the character that is in the accumulator
98EE: DEC    $0131
98F1: RTS
98F2: LDY    $0136    ;Get the pointer to the end of the Print List
```

```

*****
*
*                               PUPAS
*
*                               Print Using PARSer
*
*****

```

```

98F5: JSR   $978B   ;Get the next character from the format list
98F8: JSR   $99A7   ;Test the format list characters (See $978B)
98FB: BNE   $9911   ;If it is not one of the special characters,
                    ;Branch to print the character
98FD: STY   $0134   ;Save the index pointer to point to the start
                    ;of the Print List
9900: BCC   $991C   ;If the control character was the pound sign,
                    ;Then branch
9902: TAX
9903: JSR   $978B   ;Get the next character from the format list
9906: BCS   $990D   ;If the format list end has been found, branch
9908: JSR   $99AF   ;Check for a decimal point, equal sign, greater
                    ;Than, or pound sign
990B: BEQ   $9917   ;If there is a match, then branch
990D: LDY   $0134   ;Get the first index
9910: TXA
9911: JSR   $560C   ;Print the character that is in the accumulator
9914: JMP   $98F5   ;Continue to parse the format list
9917: BCS   $9903
9919: LDY   $0134
991C: LDX   $77
991E: BNE   $999A
9920: STX   $0131
9923: DEY
9924: DEC   $0131   ;Decrement the current character being
                    ;Processed counter
9927: JSR   $978B
992A: BCS   $99A0
992C: CMP   #' ,'
992E: BEQ   $9927
9930: JSR   $997E
9933: BCC   $9924
9935: CMP   #' .'
9937: BNE   $9941
9939: INX
993A: CPX   #$02
993C: BCC   $9927
993E: JMP   $796C
9941: JSR   $99B3

```

9944: BNE \$9951  
9946: BCC \$994B  
9948: STA \$012B  
994B: INC \$012C,X  
994E: JMP \$9927  
9951: CMP #'\$'  
9953: BNE \$9964  
9955: BIT \$0125  
9958: BPL \$994B  
995A: CLC  
995B: ROR \$0125  
995E: DEC \$012C  
9961: JMP \$994B  
9964: CMP #'^'  
9966: BNE \$997E  
9968: LDX #\$02  
996A: JSR \$978B  
996D: BCS \$993E  
996F: CMP #'^'  
9971: BNE \$993E  
9973: DEX  
9974: BPL \$996A  
9976: INC \$012F  
9979: JSR \$978B  
997C: BCS \$99A0  
997E: CMP #'+'  
9980: BNE \$999B  
9982: LDA \$0132  
9985: BPL \$998C  
9987: LDA #'+'  
9989: STA \$0132  
998C: LDA \$012E  
998F: BNE \$993E  
9991: ROR \$012E  
9994: STY \$0136  
9997: INC \$0131  
999A: RTS  
999B: CMP #'-'  
999D: BEQ \$998C  
999F: SEC  
99A0: STY \$0136  
99A3: DEC \$0136  
99A6: RTS



```
*****
*
* This routine will check to see if the character in
* the accumulator is one of the control characters for
* the format list and if it is, the carry flag will be
* set with the exception of the pound sign which the
* flags are reversed.
*
*****
```

```
99A7: CMP    #'+'      ;Is it a plus sign?
99A9: BEQ    $99C0     ;If it is, then branch
99AB: CMP    #'-'      ;Is it a minus sign?
99AD: BEQ    $99C0     ;If it is, then branch
99AF: CMP    #'.'      ;Is it a decimal point?
99B1: BEQ    $99C0     ;If it is, then branch
99B3: CMP    #'='      ;Is it an equal sign?
99B5: BEQ    $99C0     ;If it is, then branch
99B7: CMP    #'>'     ;Is it a greater than sign?
99B9: BEQ    $99C0     ;If it is, then branch
99BB: CMP    #'#'      ;Is it a pound sign?
99BD: BNE    $99C0     ;If it is not a pound sign, then branch
99BF: CLC                      ;Flag for pound sign (carry clear)
99C0: RTS                      ;Carry flag remains set if character was found
```

```
*****
*
*                               BASIC INSTR COMMAND
*
* Command syntax:  INSTR (str.#1,str.#2[,strtpos])
*
* NOTE:  str.#1 = the string to be searched
*        str.#2 = the string to search for
*        strtpos = the starting position in string
*                #1 where the search will begin
*
* This command will return the position of the string
* specified by 'str.#2' in the string specified by
* 'str.#1'. The optional starting position specified
* by 'strtpos' will determine at what position in
* string #1 the search will begin. If the starting
* position specified in 'strtpos' is greater than the
* length of string #1 or if string #1 is a null string
* or if no match was found, then INSTR will return a
* value of zero.
*
*****
```

```

99C1: LDA $66 ;Get the address of the descriptor
99C3: STA $03D6 ;For string #1 and save it
99C6: LDA $67 ;Into TMPDES
99C8: STA $03D7
99CB: JSR $77EF ;Evaluate string #2 and place the address of its
;Descriptor into locations $66, $67
99CE: JSR $77DD ;Ensure it was a string that was evaluated and
;If it was not a string, then
;Generate a 'TYPE MISMATCH' error
99D1: LDA $66 ;Get the address
99D3: STA $03D8 ;Of the descriptor
99D6: LDA $67 ;For string #2 and save it
99D8: STA $03D9 ;Into TMPDES + 2
99DB: LDX #$01 ;Set the optional starting
99DD: STX $67 ;Position to one
99DF: JSR $0386 ;Get the character after string #2's '$' and
99E2: CMP #' ) ' ;See if it is the closing parenthesis and
99E4: BEQ $99E9 ;If it is, then branch past the code for
;Obtaining the optional starting position
99E6: JSR $8809 ;Skip the comma and get the value of the
;Starting position and save it to location $67
99E9: JSR $7956 ;If the character following the starting
;Position value is not a closing parenthesis,
;Generate a 'SYNTAX' error
99EC: LDX $67 ;Get the starting position requested by the user
99EE: BNE $99F3 ;And if it is greater than zero, do not generate
;An 'ILLEGAL QUANTITY' error
99F0: JMP $7D28 ;Generate an 'ILLEGAL QUANTITY' error
99F3: DEX ;Subtract one from the starting position and
99F4: STX $63 ;Save it to be used as an index value
99F6: LDX #$03 ;Copy the address of string #1
99F8: LDA $03D6,X ;And the string's descriptor
99FB: STA $59,X ;Into locations $59, $5A (PTARG1) and
99FD: DEX ;Locations $5B, $5C (PTARG2) with PTARG1
99FE: BPL $99F8 ;Being string #1's descriptor and PTARG2 being
;String #2's descriptor
9A00: LDY #$02 ;Copy string #1's descriptor into
9A02: LDA #$59 ;$5D = length of string #1
9A04: JSR $03AB ;$5E = LSB of string #1's address in RAM BANK 1
9A07: STA $005D,Y ;$5F = MSB of string #1's address in RAM BANK 1
9A0A: LDA #$5B ;$60 = length of string #2
9A0C: JSR $03AB ;$61 = LSB of string #2's address in RAM BANK 1
9A0F: STA $0060,Y ;$62 = MSB of string #2's address in RAM BANK 1
9A12: DEY ;The above addresses are used internally
9A13: BPL $9A02 ;By INSTR, leaving original descriptors intact
9A15: LDA $60 ;Get the length of string #2
9A17: BEQ $9A54 ;And if it is zero, then return a position of 0

```

---

```

9A19: LDA #0 ;Zero the index
9A1B: STA $64 ;Pointer
9A1D: CLC ;Clear the carry flag for addition
9A1E: LDA $60 ;Add the starting position to the length of
9A20: ADC $63 ;String #2 to index over to where the search is
;To begin inside string #1
9A22: BCS $9A54 ;If the total length is greater than 255, then
;Return a position of zero
9A24: CMP $5D ;If the total is less than
9A26: BCC $9A2A ;The length of string #1, then branch to find
;The 'position'
9A28: BNE $9A54 ;If they're unequal, then return a position of 0
9A2A: LDY $64 ;Get the index pointer and if
9A2C: CPY $60 ;It equals the length of string #1,
9A2E: BEQ $9A4F ;Then branch to increment the value and exit
9A30: TYA ;Move the index pointer into the X register
9A31: CLC ;Then add it to the
9A32: ADC $63 ;Position of string #2 in string #1
9A34: TAY ;Then move it into the Y register as an index
;To string #1
9A35: LDA #$5E ;Get a character
9A37: JSR $03AB ;From string #1
9A3A: STA $79 ;And save it in SYNTMP
9A3C: LDY $64 ;Get the index pointer
9A3E: LDA #$61 ;Get a character
9A40: JSR $03AB ;From string #2
9A43: CMP $79 ;And see if it equals string #1 and
9A45: BEQ $9A4B ;If it does, then increment the pointers
9A47: INC $63 ;Increment the 'position' counter
9A49: BNE $9A19 ;And restart the main loop
9A4B: INC $64 ;Increment the index pointer
9A4D: BNE $9A2A ;And branch to continue the comparison
9A4F: INC $63 ;Increment the 'position' counter
9A51: LDA $63 ;Get the position of string #2 in string #1
9A53: .BYTE $2C ;MASK to fall through to $9A56
9A54: LDA #0 ;Return a position of zero
9A56: STA $FF03 ;Enable BANK 14
9A59: PHA ;Save the position onto the stack
9A5A: LDA $03D8 ;Get the address of string #1
9A5D: LDY $03D9 ;From TEMPDES + 2 and free
9A60: JSR $8785 ;Up the space from the temporary string stack
9A63: STA $FF03 ;Enable BANK 14
9A66: LDA $03D6 ;Get the address of string #2
9A69: LDY $03D7 ;From TEMPDES and free up
9A6C: JSR $8785 ;The space from the temporary string stack
9A6F: PLA ;Get the position off of the stack,
9A70: TAY ;Move it into the Y register

```

```

9A71: JMP    $84D4      ;And convert it to a Floating Point number
                          ;In FAC1 and exit
9A74: JSR    $9D8F
9A77: LDX    #0
9A79: INX
9A7A: SEC
9A7B: SBC    #$5A
9A7D: BCS    $9A79
9A7F: DEY      ;Decrement the MSB
9A80: BPL    $9A79      ;Continue until done
9A82: STX    $1149      ;Save the index
9A85: PHA      ;Save the LSB onto the stack
9A86: ADC    #$5A      ;Add 90 to the LSB
9A88: JSR    $9A94
9A8B: PLA      ;Get the LSB off of the stack
9A8C: CLC
9A8D: EOR    #$FF      ;Invert the value and
9A8F: ADC    #$01      ;Add one to generate the two's complement of the
value
9A91: DEC    $1149      ;Decrement the index
9A94: LDX    #$FF
9A96: INX
9A97: SEC
9A98: SBC    #$0A
9A9A: BCS    $9A96
9A9C: ADC    #$0A
9A9E: STA    $8E
9AA0: TXA
9AA1: ASL
9AA2: TAX
9AA3: LDA    $9F2A,X
9AA6: LDY    $9F29,X
9AA9: CLC
9AAA: DEC    $8E
9AAC: BMI    $9ABA
9AAE: ADC    $9F3D+1,X
9AB1: PHA
9AB2: TYA
9AB3: ADC    $9F3D,X
9AB6: TAY
9AB7: PLA
9AB8: BCC    $9AA9
9ABA: PHA
9ABB: LDX    #0
9ABD: LDA    $1149
9AC0: LSR
9AC1: BCS    $9AC5

```

```

9AC3: LDX    #$02
9AC5: PLA
9AC6: STA    $114A,X
9AC9: TYA
9ACA: STA    $114B,X
9ACD: RTS
9ACE: LDY    #$19
9AD0: BCC    $9AD4
9AD2: LDY    #$1B
9AD4: LDA    $1149
9AD7: ADC    #$02
9AD9: LSR
9ADA: LSR
9ADB: PHP
9ADC: JSR    $9D8F
9ADF: CPY    #$FF
9AE1: BCC    $9AEA
9AE3: TXA
9AE4: TAY
9AE5: JSR    $9D8F
9AE8: BCS    $9AED
9AEA: JSR    $9DAE
9AED: PLP
9AEE: BCS    $9B0B
9AF0: JMP    $9D9E
9AF3: STA    $114E
9AF6: LDX    #$23
9AF8: ASL    $114E
9AFB: JSR    $9ACE
9AFE: STA    $1131,X
9B01: TYA
9B02: STA    $1132,X
9B05: INX
9B06: INX
9B07: CPX    #$2B
9B09: BCC    $9AF8
9B0B: RTS

```

```

*****
*
*          BASIC RDOT FUNCTION          *
*
* Command syntax:  RDOT (X)            *
*
* NOTE:  If 'X' equals zero, then RDOT will return *
*        the X coordinate of the pixel cursor.    *

```

```

*           If 'X' equals one, then RDOT will return      *
*           the Y coordinate of the pixel cursor.         *
*                                                       *
*           If 'X' equals two, then RDOT will return      *
*           the color source of the pixel cursor.         *
*                                                       *
* This function will return the coordinates or color      *
* source of the pixel cursor depending on the value      *
* specified by 'X'.                                     *
*                                                       *
*****

```

```

9B0C: JSR   $87F7      ;Get the argument into the X register
9B0F: CPX   #2         ;If it less than two,
9B11: BCC   $9B23      ;Branch to get the pixel cursor's coordinates
9B13: BEQ   $9B18      ;If the value is equal to2 , then branch to get
                        ;The color source of the pixel cursor
9B15: JMP   $7D28      ;Generate an 'ILLEGAL QUANTITY' error
9B18: JSR   $9C49      ;Get the color source of the pixel cursor
9B1B: TAY                   ;Move the color source value to the X register
9B1C: BCC   $9B20      ;If the color source was in range, then branch
9B1E: LDY   #0         ;If the color source was not in range, return
                        ;A value of zero
9B20: JMP   $84D4      ;Convert the value to Floating Point number-exit
9B23: TXA                   ;Multiply the argument
9B24: ASL                   ;By 2
9B25: TAX                   ;And store it in the X register
9B26: LDA   $1131,X      ;Get the LSB of the pixel cursor's coordinate
9B29: TAY                   ;Save it in the Y register
9B2A: LDA   $1132,X      ;Get the MSB of the pixel cursor's coordinate
9B2D: JMP   $793C      ;Convert the coordinate value to Floating
                        ;Point format and exit
9B30: LDX   #$02        ;Set up an index to the DRAW line starting Y
                        ;coordinate
9B32: LDY   #$06        ;Set up an index to the destination Y coordinate
9B34: LDA   #0          ;Clear the comparison
9B36: STA   $113D,X      ;Flags
9B39: STA   $113E,X
9B3C: JSR   $9D99      ;Calculate the difference between the starting
                        ;And destination coordinates
9B3F: BPL   $9B49      ;If the starting coordinate was less than the
                        ;Destination coordinate, then branch
9B41: DEC   $113D,X      ;If the starting coordinate is greater than the
9B44: DEC   $113E,X      ;Destination coordinate, then set the sign to
                        ;negative
9B47: BNE   $9B54      ;If the sign is negative, then branch
9B49: CMP   #0          ;If the LSB of the difference is not zero,

```

```

9B4B: BNE    $9B51    ;Then branch
9B4D: CPY    #0       ;If the MSB of the difference is zero,
9B4F: BEQ    $9B54    ;Then branch to leave the compare flag at zero
                          ;(equal)
9B51: INC    $113D,X  ;If it is not equal to zero, then get the flag
                          ;For destination coordinate is greater
9B54: STA    $1139,X  ;Save the LSB of the difference
9B57: ASL                    ;And multiply it by 2
9B58: STA    $1141,X  ;Save the result
9B5B: TYA                    ;Save the MSB
9B5C: STA    $113A,X  ;Of the difference
9B5F: ROL                    ;Add the carry to it
9B60: STA    $1142,X  ;And save the result
9B63: DEX                    ;Decrement the starting coordinate
9B64: DEX                    ;Index by 2
9B65: LDY    #$04     ;Set the destination coordinate index
                          ;To the X coordinate
9B67: CPX    #0       ;Check to see if the index is pointing to the
9B69: BEQ    $9B34    ;Starting X coordinate and if so, then branch
9B6B: LDX    #$0A
9B6D: LDY    #$08
9B6F: JSR    $9D7C
9B72: LDA    #0
9B74: ROL                    ;Shift the carry, if any
9B75: ROL                    ;Into bit 2 of the accumulator
9B76: STA    $1147    ;Save it
9B79: EOR    #$02     ;Invert bit 2
9B7B: STA    $1148    ;And save it
9B7E: CLC
9B7F: LDA    #$10
9B81: ADC    $1147
9B84: TAY
9B85: PHA
9B86: EOR    #$02
9B88: TAX
9B89: JSR    $9D7C
9B8C: STA    $1131,X
9B8F: TYA
9B90: STA    $1132,X
9B93: PLA
9B94: TAY
9B95: CLC
9B96: LDA    #$08
9B98: ADC    $1148
9B9B: TAX
9B9C: JSR    $9D7C
9B9F: STA    $1145

```

```

9BA2: STY    $1146
9BA5: JSR    $9BFB      ;Plot a point of the line
9BA8: LDY    $1148
9BAB: SEC
9BAC: LDA    $1139,Y
9BAF: SBC    #$01
9BB1: STA    $1139,Y
9BB4: BCS    $9BC1
9BB6: LDA    $113A,Y
9BB9: SBC    #0
9BBB: STA    $113A,Y
9BBE: BCS    $9BC1
9BC0: RTS
9BC1: LDX    $1147
9BC4: LDA    $1146
9BC7: BMI    $9BCF
9BC9: JSR    $9BEA
9BCC: LDX    $1148
9BCF: CLC
9BD0: LDA    $1145
9BD3: ADC    $1141,X
9BD6: STA    $1145
9BD9: LDA    $1146
9BDC: ADC    $1142,X
9BDF: STA    $1146
9BE2: LDX    $1148
9BE5: JSR    $9BEA
9BE8: BEQ    $9BA5      ;Branch to plot the next point
9BEA: LDY    #$02      ;Load the Y register with the number of
                        ;repetitions
9BEC: CLC          ;Clear the carry for addition
9BED: LDA    $1131,X   ;Get the LSB of the coordinate
9BF0: ADC    $113D,X   ;Add the index to the next point
9BF3: STA    $1131,X   ;And save it back
9BF6: INX          ;Move to the next
9BF7: DEY          ;Coordinate
9BF8: BNE    $9BED     ;Continue until both the X and Y coordinates are
9BFA: RTS          ;Updated and exit

*****
*
*          PLOT THE PIXEL
*
*****

9BFB: LDA    $116C      ;Check to see if we are
9BFE: ORA    $116B      ;In the double width mode

```



---

```

9C01: BEQ   $9C19      ;If we are not, then branch
9C03: INC   $1131      ;Increment the X coordinate's LSB
9C06: BNE   $9C0B      ;If there was an overflow, then branch
9C08: INC   $1132      ;Increment the X coordinate's MSB
9C0B: JSR   $9C19      ;Plot a single width point
9C0E: LDX   $1131      ;Get the X coordinate's LSB
9C11: BNE   $9C16      ;If it is not equal to zero, then branch
9C13: DEC   $1132      ;Decrement the MSB
9C16: DEC   $1131      ;Decrement the LSB by one and plot the 2nd point
9C19: JSR   $9D24      ;Check the range of the coordinates
9C1C: BCS   $9C42      ;If they are out of range, then exit
9C1E: JSR   $9C70      ;Set the color of the point to be plotted
9C21: JSR   $9CE8      ;Calculate the HIRES address and bit position
9C24: STA   $116D      ;Save the bit value
9C27: LDA   ($8C),Y    ;Get the current value of the HIRES location
9C29: ORA   $116D      ;Set the specified point
9C2C: BIT   $D8        ;Check for the multicolor mode
9C2E: BPL   $9C43      ;If multicolor mode is not being used, branch
9C30: PHA                   ;Temporarily save the point value
9C31: LDX   $83        ;Get the color source value
9C33: LDA   $116D      ;Get the bit position value
9C36: AND   $9F25,X    ;Mask proper bit for multicolor
9C39: STA   $116D      ;Save the value
9C3C: PLA                   ;Get the point value back
9C3D: EOR   $116D      ;Set the proper point
9C40: STA   ($8C),Y    ;Save it back to the HIRES screen
9C42: RTS                   ;Exit the routine
9C43: LDX   $83        ;Get the current color source being used
9C45: BNE   $9C40      ;If the color source is not the background
                        ;Color, then branch
9C47: BEQ   $9C3D      ;If the color source is the background
                        ;Color, then branch
9C49: JSR   $9CE3      ;Calculate the screen address and the bit value
9C4C: BCS   $9C6F      ;If the coordinate was out of range, then exit
9C4E: STA   $116D      ;Save the bit position value
9C51: LDA   ($8C),Y    ;Get the value at the current coordinates
9C53: AND   $116D      ;Mask the selected bit
9C56: ROL                   ;Shift the bit
9C57: DEX                   ;To its proper location according to its
9C58: BPL   $9C56      ;Bit position
9C5A: ROL                   ;In the X register
9C5B: BIT   $8B
9C5D: BMI   $9C65
9C5F: AND   #$03
9C61: CMP   $83
9C63: CLC
9C64: RTS

```

```

9C65: CLC
9C66: AND    #$03
9C68: BEQ    $9C6D
9C6A: LDX    #0
9C6C: RTS
9C6D: LDX    #$FF
9C6F: RTS
9C70: LDA    $C033,X    ;Get line starts low byte
9C73: STA    $8C        ;Save it
9C75: LDA    $9CCA,X    ;Get line starts high byte
9C78: STA    $8D        ;Save it
9C7A: LDA    $83        ;Get the current color source selected
9C7C: BNE    $9C86      ;If it is not equal to zero, then branch
9C7E: LDA    $03E2      ;Get the packed foreground/background color
9C81: BIT    $D8        ;Check to see which graphic mode we are in
9C83: BPL    $9C8D      ;If it is not the multicolor mode, then branch
9C85: RTS                ;Exit the routine
9C86: CMP    #$02        ;Is the color source Multicolor 1?
9C88: BNE    $9C9A      ;If it is not, then branch
9C8A: LDA    $03E3      ;Get the packed foreground/multicolor 1 color
9C8D: AND    #$0F        ;Drop the foreground color
9C8F: STA    $77        ;Save the multicolor 1 value
9C91: LDA    ($8C),Y    ;Get the current color in the specified cell
9C93: AND    #$F0        ;Drop the old multicolor value
9C95: ORA    $77        ;Replace it with the new multicolor value
9C97: STA    ($8C),Y    ;And save it back
9C99: RTS                ;Exit the routine
9C9A: BCS    $9CAC      ;If the color source is not the background
                        ;Color, then branch
9C9C: LDA    $03E2      ;Get the packed foreground/background color
9C9F: AND    #$F0        ;Drop the background color
9CA1: STA    $77        ;Save it
9CA3: LDA    ($8C),Y    ;Get the current color in this color cell
9CA5: AND    #$0F        ;Drop the old foreground color
9CA7: ORA    $77        ;Replace it with the new foreground color
9CA9: STA    ($8C),Y    ;And save it back into the color cell
9CAB: RTS                ;Exit the routine
9CAC: LDA    $8D        ;Get the color address MSB
9CAE: AND    #$03        ;Drop all bits except bits 0 and 1
9CB0: ORA    #$D8        ;Point the MSB to the color memory at $D800
9CB2: STA    $8D        ;Save the MSB
9CB4: LDA    #0        ;Set the MMU
9CB6: STA    $FF00      ;Configuration to BANK 15
9CB9: SEI                ;Stop the background operations
9CBA: LDA    $01        ;Get the Data Register
9CBC: PHA                ;Save it onto the stack
9CBD: AND    #$FE        ;Drop bit 0

```

```

9CBF: STA  $01      ;To select color RAM 0
9CC1: LDA  $85      ;Get the Multicolor 2 value
9CC3: STA  ($8C),Y  ;Save it in the current color cell
9CC5: PLA                      ;Get the old Data Register value
9CC6: STA  $01      ;And store it back
9CC8: CLI                      ;Restart the background operations
9CC9: RTS                      ;And exit the routine

```

```

*****
*
* This is a table of the BASIC line start MSBs for
* color memory at $1C00 when the GRAPHICS screen is
* enabled.
*
*****

```

```

9CCA: .BYTE $1C,$1C,$1C
9CCD: .BYTE $1C,$1C,$1C
9CD0: .BYTE $1C,$1D,$1D
9CD3: .BYTE $1D,$1D,$1D
9CD6: .BYTE $1D,$1E,$1E
9CD9: .BYTE $1E,$1E,$1E
9CDC: .BYTE $1E,$1F,$1F
9CDF: .BYTE $1F,$1F,$1F

```

```

*****
*
* This routine gets the HIRES address and bit value
* for plotting.
*
*****

```

```

9CE3: JSR  $9D24      ;Check the range of the coordinates
9CE6: BCS  $9D1B      ;If they are out of range, then exit
9CE8: TYA                      ;Put the X coordinate into the accumulator
9CE9: CLC                      ;Clear the carry for addition
9CEA: ADC  $C033,X     ;Add the LSB of the screen line
9CED: STA  $8C          ;And save it
9CEF: LDA  $C04C,X     ;Get the MSB of the screen line
9CF2: ADC  #0           ;Add the carry
9CF4: ASL  $8C          ;Calculate the corresponding
9CF6: ROL                      ;HIRES screen address
9CF7: ASL  $8C          ;From the coordinates
9CF9: ROL
9CFA: ASL  $8C
9CFC: ROL
9CFD: STA  $8D          ;And save the MSB

```

```

9CFF: LDA $1133 ;Get the Y coordinate
9D02: AND #$07 ;Drop all the bits except bit position
9D04: TAY ;Save it in the Y register
9D05: LDA $1131 ;Get the X coordinate LSB
9D08: BIT $D8 ;Check to see if we are in Standard or
;Multicolor graphics
9D0A: PHP ;Save the status register
9D0B: BPL $9D0E ;If it is not the Multicolor mode, then branch
9D0D: ASL ;If it is the Multicolor mode, then multiply the
;LSB by 2
9D0E: AND #$07 ;Calculate the bit position
9D10: TAX ;Save the result in the X register
9D11: LDA $9D1C,X ;Get the appropriate bit value
9D14: PLP ;Get the status back
9D15: BPL $9D1B ;If it is not the Multicolor mode, then exit
9D17: INX ;Add one to the bit position
9D18: ORA $9D1C,X ;Get the bit value to set
9D1B: RTS ;And exit the routine

```

```

*****
*
*
*           MSKTBL
*
*           MaSK TaBLe
*
* This table contains the value for bits 7 - 0
* respectively, and is used by the graphics routines
* and sound routines.
*
*****

```

	HEX	BIT - 76543210	DECIMAL VALUE	PLAY CHARACTER
	---	-----	-----	-----
9D1C:	.BYTE \$80	; 10000000	128	V (VOICE)
9D1D:	.BYTE \$40	; 01000000	64	O (OCTAVE)
9D1E:	.BYTE \$20	; 00100000	32	T (TUNE
ENVELOPE)				
9D1F:	.BYTE \$10	; 00010000	16	X (FILTER
ON/OFF)				
9D20:	.BYTE \$08	; 00001000	8	U (VOLUME)
9D21:	.BYTE \$04	; 00000100	4	NOT USED
9D22:	.BYTE \$02	; 00000010	2	NOT USED
9D23:	.BYTE \$01	; 00000001	1	NOT USED

```

*****
*
*   This routine checks the range of the X, Y
*   coordinates and sets the carry if they are out of
*   range.
*
*****

9D24: LDA   $1132      ;Get the MSB of the X coordinate
9D27: LSR                ;Shift bit 0 into the carry
9D28: BNE   $9D48      ;If the MSB is greater than 1, then exit
9D2A: LDA   $1131      ;Get the LSB of the X coordinate
9D2D: ROR                ;Shift bit 0 into the carry
9D2E: LSR                ;Return the LSB to the original without bit 0
9D2F: BIT   $D8        ;Check to see which GRAPHIC mode we are in
9D31: BMI   $9D34      ;If it is the multicolor mode, then branch
9D33: LSR                ;Divide the X coordinate LSB by 2
9D34: TAY                ;Transfer it to the Y register
9D35: CPY   #$28       ;Compare the X coordinate to its maximum
9D37: BCS   $9D48      ;If the X coordinate is out of range, then exit
9D39: LDA   $1134      ;Get the Y coordinate MSB
9D3C: BNE   $9D48      ;If the Y coordinate has an MSB greater
                        ;Than zero, then exit
9D3E: LDA   $1133      ;Get the Y coordinate LSB
9D41: LSR                ;Divide it by 8
9D42: LSR
9D43: LSR
9D44: TAX                ;And save it to the X register
9D45: CMP   #$19       ;Compare the value to the maximum value
                        ;For the Y coordinate
9D47: RTS                ;Exit the routine
9D48: SEC                ;Set the carry for value out of range
9D49: RTS                ;Exit the routine

*****
*
*   This routine first checks if scaling is selected
*   and if scaling is selected, then the coordinates
*   which are indexed by the X register are adjusted
*   according to the scaling factor.
*
*****

9D4A: LDA   $116A      ;Check to see if scaling is selected
9D4D: BEQ   $9D66      ;If not, then exit
9D4F: LDA   $87        ;Get the LSB
9D51: LDY   $88        ;And the MSB of the X scaling factor

```

```

9D53: JSR   $9D5A      ;Adjust the coordinates according to the scaling
                        ;Factor
9D56: LDA   $89        ;Get the LSB
9D58: LDY   $8A        ;And the MSB of the Y scaling factor
9D5A: JSR   $9DAE      ;Calculate the coordinate according to the
                        ;Scaling factor
9D5D: STA   $1131,X    ;Save the adjusted coordinate LSB
9D60: TYA                   ;Move the MSB to the accumulator
9D61: INX                   ;Increment the index
9D62: STA   $1131,X    ;Save the adjusted coordinate MSB
9D65: INX                   ;Increment the index
9D66: RTS                   ;Exit the routine
9D67: BCC   $9D70
9D69: BCS   $9D7F
9D6B: BCS   $9D7C      ;If the start coordinate is greater, then branch
9D6D: JSR   $9D8F      ;Get the destination coordinates into the
                        ;Accumulator and Y register
9D70: CLC                   ;Clear the carry for addition
9D71: ADC   $1131,X    ;Add the starting coordinate LSB to the
                        ;destination LSB
9D74: PHA                   ;Save it onto the stack
9D75: TYA                   ;Move the MSB of the destination coordinate into
                        ;The accumulator
9D76: ADC   $1132,X    ;Add it to the MSB of the starting coordinate
9D79: TAY                   ;And move it into the Y register
9D7A: PLA                   ;Get the LSB back off of the stack
9D7B: RTS                   ;And exit the routine
9D7C: JSR   $9D8F      ;Get the destination coordinates into the
                        ;Accumulator and Y register
9D7F: SEC                   ;Set the carry for subtraction
9D80: SBC   $1131,X    ;Subtract the starting coordinate LSB from the
                        ;Destination LSB
9D83: STA   $59         ;Save it
9D85: TYA                   ;Move the MSB of the destination coordinate
                        ;Into the accumulator
9D86: SBC   $1132,X    ;Subtract the starting coordinate MSB from the
                        ;Destination MSB
9D89: TAY                   ;Move the result to the Y register
9D8A: PHP                   ;Save the status register onto the stack
9D8B: LDA   $59         ;Get the LSB of the difference
9D8D: PLP                   ;Restore the status register
9D8E: RTS                   ;And exit the routine

```

```

*****
*
* This routine is used to get the LSB and the MSB of
* pixel cursor coordinate which is indexed with the
*

```

```

*   Y register into the accumulator and the Y register   *
*   respectively.                                       *
*                                                         *
*****
9D8F: LDA   $1131,Y   ;Get the LSB of the coordinate
9D92: PHA           ;Save it onto the stack
9D93: LDA   $1132,Y   ;Get the MSB of the coordinate
9D96: TAY           ;Save it into the Y register
9D97: PLA           ;Get the LSB back off of the stack
9D98: RTS           ;And exit the routine

*****
*                                                         *
*   This routine calculates the difference between 2     *
*   coordinates which are indexed by the X and Y       *
*   registers.  If the ending coordinate is less than  *
*   the starting coordinate, the 2's complement of the *
*   result is calculated.  The result is stored in the *
*   accumulator and the Y register (LSB, MSB).        *
*                                                         *
*   On entry to this routine, the X register is indexed *
*   to the starting coordinate and the Y register to the *
*   ending coordinate.                                  *
*                                                         *
*****
9D99: JSR   $9D7C     ;Calculate the difference between the starting
                    ;And destination coordinates
9D9C: BPL   $9DAD     ;If the starting coordinate was greater than the
                    ;Destination coordinate, then exit

*****
*                                                         *
*   This routine calculates the twos complement        *
*   (EOR #$FF + 1) of the 16 bit value that is in the *
*   accumulator and the Y register (LSB, MSB).        *
*                                                         *
*****
9D9E: PHP           ;Save the status register onto the stack
9D9F: CLC           ;Clear the carry for addition
9DA0: EOR   #$FF     ;Compute the two's complement
9DA2: ADC   #$01     ;Of the LSB
9DA4: PHA           ;And save it onto the stack
9DA5: TYA           ;Move the MSB into the accumulator
9DA6: EOR   #$FF     ;And compute the two's complement

```

```

9DA8:  ADC    #0           ;Of it
9DAA:  TAY                ;Move the MSB back into the Y register
9DAB:  PLA                ;Get the LSB back off of the stack
9DAC:  PLP                ;Restore the status register
9DAD:  RTS                ;And exit the routine
    
```

```

*****
*
* This routine is used to adjust the coordinate which
* is indexed by the X register according to the
* scaling factor.
*
* On entry to this routine, the Accumulator and the
* Y register must hold the current scaling factor,
* which is normally stored in locations $87, $88 for
* the X coordinate and in locations $89, $8A for the
* Y coordinate.
*
*****
    
```

```

9DAE:  STY    $8E          ;Save the MSB
9DB0:  STA    $8F          ;And the LSB of the scaling factor
9DB2:  LDA    $1131,X      ;Get the LSB
9DB5:  LDY    $1132,X      ;And the MSB of the coordinate
9DB8:  PHP                ;Save the status register onto the stack
9DB9:  JSR    $9D9C
9DBC:  STA    $1131,X
9DBF:  TYA
9DC0:  STA    $1132,X
9DC3:  LDA    #0
9DC5:  STA    $1177
9DC8:  LDY    #$10
9DCA:  LSR    $8E
9DCC:  ROR    $8F
9DCE:  BCC    $9DDF
9DD0:  CLC
9DD1:  ADC    $1131,X
9DD4:  PHA
9DD5:  LDA    $1177
9DD8:  ADC    $1132,X
9ddb:  STA    $1177
9DDE:  PLA
9DDF:  LSR    $1177
9DE2:  ROR
9DE3:  DEY
9DE4:  BNE    $9DCA
9DE6:  ADC    #0
    
```



```

9DE8: LDY    $1177
9DEB: BCC    $9DEE
9DED: INY
9DEE: PLP                      ;Get the status register back off of the stack
9DEF: JMP    $9D9C

```

```

*****
*
* This routine moves the coordinates that are stored
* at locations $1135 - $1138 to $1131 - $1134
* (X and Y destination to X and Y position).
*
*****

```

```

9DF2: LDY    #0
9DF4: JSR    $9DF9      ;Move the values at locations $1135 - $1136
                        ;To $1131 - 1132
9DF7: LDY    #$02      ;Set the index to 2 to move locations
                        ;$1137 - $1138 to $1133 - $1134
9DF9: LDA    $1135,Y   ;Move the values indexed
9DFC: STA    $1131,Y   ;By the Y register from locations
9DFF: LDA    $1136,Y   ;$1135+Y - $1136+Y to $1131+Y - $1132+Y
9E02: STA    $1132,Y
9E05: RTS                      ;Exit the routine

```

```

*****
*
* This routine is used to get a 16 bit coordinate
* into the accumulator and the Y register (MSB, LSB).
* If a coordinate is specified, the carry flag is set.
* If a coordinate is not specified, then the
* accumulator, Y register, and carry flag are cleared.
* Upon entry to this routine, TXTPTR must point to the
* first character of the coordinate.
*
*****

```

```

9E06: JSR    $0386      ;Get the character that TXTPTR is pointing to
9E09: BEQ    $9E17      ;If it is a colon or an end of line terminator,
                        ;Then branch
9E0B: JSR    $795C      ;Check for a comma. If a comma is not found,
                        ;Generate a 'SYNTAX' error. If a comma is
                        ;Found, then get the next character
9E0E: CMP    #','        ;Is the next character a comma?
9E10: BEQ    $9E17      ;If it is a comma, then branch
9E12: JSR    $8812      ;If it is not a comma, then get the value
                        ;Into the accumulator and the Y register

```

```

9E15: SEC                ;Set the carry
9E16: RTS                ;And exit the routine
9E17: LDA    #0          ;Clear the accumulator,
9E19: TAY                ;The Y register,
9E1A: CLC                ;And the carry flag to indicate that there are
                        ;No parameters specified
9E1B: RTS                ;Exit the routine

```

```

*****
*
* This routine is used to get an 8 bit coordinate
* in the X register. If a coordinate is specified,
* the carry flag is set.
*
*****

```

```

9E1C: LDX    #0          ;Set the default value to zero
9E1E: JSR    $0386       ;Get a character
9E21: BEQ    $9E1A       ;If it is a zero or a colon, then exit
9E23: JSR    $795C       ;Check for a comma and get the next character
9E26: CMP    #','        ;Is the next character a comma?
9E28: BEQ    $9E1A       ;If it is, then branch
9E2A: JSR    $87F4       ;Convert the value to HEX and store it in the
                        ;X register
9E2D: SEC                ;Set the carry for character found
9E2E: RTS                ;And exit the routine

```

```

*****
*
* This routine parses the graphics commands.
*
*****

```

```

9E2F: JSR    $A074       ;Check to see if the HIRES screen is allocated
9E32: LDX    #$01        ;Default color source
9E34: JSR    $0386       ;Get the last accessed character
9E37: BEQ    $9E4C       ;If there is none there, then exit
9E39: CMP    #','        ;Is it a comma?
9E3B: BEQ    $9E4C       ;If it is, then exit
9E3D: JSR    $87F4       ;If not, then get the value into the X register
9E40: CPX    #$04        ;Compare it to the maximum value + 1
9E42: BCS    $9E4F       ;If the value is too large, then generate an
                        ;'ILLEGAL QUANTITY' error
9E44: CPX    #$02        ;If the Multicolor color source was selected,
9E46: BIT    $D8          ;Then check to see if the Multicolor HIRES
                        ;Screen was allocated
9E48: BMI    $9E4C       ;If it was, then branch

```

---

```

9E4A: BCS $9E4F ;If it was not allocated, then error
9E4C: STX $83 ;Save the color source
9E4E: RTS ;And exit the routine
9E4F: JMP $7D28 ;Generate an 'ILLEGAL QUANTITY' error
9E52: JSR $0386 ;Get the last character that was accessed
9E55: BEQ $9E5E ;If there is nothing there, then branch
9E57: JSR $795C ;Check for a comma and get the next character in
;The accumulator

9E5A: CMP #',' ;Is the next character a comma?
9E5C: BNE $9E70 ;If it is not, then branch
9E5E: LDY #0 ;If it is a comma,
9E60: LDA $1131,Y ;Then make the current X, Y coordinates
9E63: STA $1131,X ;The destination coordinates
9E66: INX ;Increment the index to the next value
9E67: INY ;And continue until
9E68: CPY #$04 ;Four bytes
9E6A: BNE $9E60 ;Are done
9E6C: RTS ;Exit the routine
9E6D: JSR $795C ;Check for a comma and get the next character in
;The accumulator

9E70: STX $1178 ;Save the index
9E73: JSR $9F08 ;Get the X coordinate and store it
9E76: JSR $0386 ;Get the character after the coordinate
9E79: CMP #',' ;Is it a comma?
9E7B: BEQ $9ED3 ;If it is, then branch
9E7D: CMP #';' ;Is it a semicolon?
9E7F: BEQ $9E84 ;If it is, then branch
9E81: JMP $796C ;If it is not a semicolon or a comma, then
;Generate a 'SYNTAX' error

9E84: JSR $0380 ;Get the next character
9E87: JSR $8812 ;Get the Y coordinate and convert it to HEX in
;The Y register and the accumulator

9E8A: STA $77 ;Save the MSB
9E8C: TYA ;Put the LSB in the accumulator
9E8D: LDY $77 ;Get the MSB into the Y register
9E8F: JSR $9A77
9E92: LDX $1178
9E95: LDA $1131,X
9E98: STA $1133,X
9E9B: LDA $1132,X
9E9E: STA $1134,X
9EA1: JSR $9D4A
9EA4: LDA #$0E
9EA6: STA $1179
9EA9: CLC
9EAA: LDX $1178
9EAD: JSR $9ACE

```

```

9EB0: STA  $1131,X
9EB3: TYA
9EB4: STA  $1132,X
9EB7: LDY  #0
9EB9: LSR  $1179
9EBC: BCC  $9EC0
9EBE: LDY  #$02
9EC0: JSR  $9D6B
9EC3: STA  $1131,X
9EC6: TYA
9EC7: STA  $1132,X
9ECA: INX
9ECB: INX
9ECC: LSR  $1179
9ECF: BNE  $9EAD
9ED1: CLC
9ED2: RTS

```

```

*****
*
* This routine is used to get the coordinates and
* then adjust them according to the scaling factor
* and the relative coordinate offset.
*
*****

```

```

9ED3: JSR  $0380      ;Increment TXTPTR
9ED6: INC  $1178      ;Increment the coordinate
9ED9: INC  $1178      ;Index by 2
9EDC: JSR  $9F08      ;Get the coordinate and store it
9EDF: LDX  $1178      ;Get the index
9EE2: DEX
9EE3: DEX              ;And decrement
9EE4: JSR  $9D4A      ;It by 2
                      ;Adjust the coordinate according to the scaling
                      ;Factor and save it

9EE7: LDY  #$02
9EE9: LDX  $1178      ;Get the index
9EEC: INX              ;And increment it
9EED: INX              ;By 2
9EEE: DEX              ;Decrement the
9EEF: DEX              ;Index by 2
9EF0: LSR  $1179      ;See if a relative coordinate was specified
9EF3: BCC  $9EFF      ;If the coordinate was not relative, then branch
9EF5: JSR  $9D6D      ;Add the offset to the present coordinate
9EF8: STA  $1131,X    ;Save the resulting
9EFB: TYA              ;Coordinate
9EFC: STA  $1132,X

```

```

9EFF: LDY #0 ;Set the index to the next coordinate
9F01: CPX $1178 ;And continue until
9F04: BEQ $9EEE ;Both of the coordinates have been stored
9F06: CLC ;Clear the carry
9F07: RTS ;And exit the routine

```

```

*****
*
* This routine first checks to see if a relative
* coordinate (one preceded by a '+' or '-') is
* specified. If it is, then bit 0 of location $1179
* is set and the coordinate is stored.
*
* NOTE: If a negative coordinate such as -10 or -100
* is specified, the routine at $8812 will cause
* an 'ILLEGAL QUANTITY' error to be generated.
* This is because TXTPTR is pointing at the '+'
* or '-' and not at the first character of the
* coordinate where it should have been pointed.
*
*****

```

```

9F08: JSR $0386 ;Get the last character accessed by CHRGET
9F0B: CMP #SAA ;Is it the token for '+'?
9F0D: BEQ $9F14 ;If it is, then branch
9F0F: CMP #SAB ;Is it the token for '-'?
9F11: BEQ $9F14 ;If it is, then branch
9F13: CLC ;Clear the carry to indicate a normal coordinate
9F14: ROL $1179 ;Shift the carry into bit 0
9F17: JSR $8812 ;Get the coordinate and store it in the
;Accumulator
;And the Y register (MSB, LSB)
9F1A: LDX $1178 ;Get the stored index
9F1D: STA $1132,X ;Save the MSB of the coordinate
9F20: TYA ;Move the LSB to the accumulator
9F21: STA $1131,X ;Save the LSB of the coordinate
9F24: RTS ;And exit the routine

9F25: .BYTE $FF,$AA,$55
9F28: .BYTE $00,$00,$00
9F2B: .BYTE $2C,$71,$57
9F2E: .BYTE $8D,$80,$00
9F31: .BYTE $A4,$8F,$C4
9F34: .BYTE $19,$DD,$B2
9F37: .BYTE $F0,$90,$FC
9F3A: .BYTE $1C,$FF,$FF
9F3D: .BYTE $04,$72,$04

```

---

```

9F40: .BYTE $50,$04,$0B
9F43: .BYTE $03,$A8,$03
9F46: .BYTE $28,$02,$90
9F49: .BYTE $01,$E3,$01
9F4C: .BYTE $28,$00,$63
9F4F: LDA $76 ;Get the flag to see if the GRAPHICS screen is
9F51: BEQ $9F54 ;Currently allocated and if it is, then branch
9F53: RTS ;Exit this routine
9F54: LDA $1211 ;Get the end of the program's MSB and add #$24
9F57: CLC ;To it to point to $4000 + the end of
9F58: ADC #$24 ;The program's address
9F5A: BCS $9F6A ;If an overflow has occurred, then generate
;An 'OUT OF MEMORY' error
9F5C: STA $62 ;Save the MSB of the new End of BASIC address
9F5E: CMP $1213 ;Check to ensure that it is not greater than the
9F61: BCC $9F6D ;Highest available to BASIC (usually $FF00) and
;If it is not greater, then branch
9F63: BNE $9F6A ;If the MSBs are not equal, then generate an
;'OUT OF MEMORY' error
9F65: CPY $1212 ;NOTE: Before you think "useless code",
9F68: BCC $9F6D ;Commodore put this instruction here in case you
;Move MAXMEMO
9F6A: JMP $4D3A ;Generate an 'OUT OF MEMORY' error
9F6D: DEC $76 ;Set MVDFLG to $FF to indicate the HIRES screen
;Is now allocated
9F6F: LDA $1210 ;Get the LSB of the End of Program Text (TXTTOP)
9F72: STA $24 ;And save it
9F74: LDA $62 ;Get the MSB of the address of where to move
9F76: STA $25 ;The program to set up the move to address
9F78: LDX $1210 ;Get the End of the Program Text
9F7B: STX $26 ;And save it as the Move
9F7D: LDA $1211 ;From address
9F80: STA $27
9F82: SEC ;Set the carry for subtraction
9F83: SBC #$1C ;Subtract $1Cxx from the End of Program Text
;Address to
9F85: TAY ;Get the number of bytes that need to be moved
9F86: TXA ;Move the LSB of the End of Text into the
;Accumulator
9F87: EOR #$FF ;Subtract that value from $FF to give
;The two's complement
9F89: STA $50 ;And save it in TEMPF3
9F8B: TYA ;Move the MSB of the End of Text into the
;Accumulator
9F8C: EOR #$FF ;Subtract that value from $FF to give the
;Two's complement
9F8E: STA $51 ;Save it in TEMPF3 + 1, the two's complement

```

```

;Will give the result of $FFFF minus the number
;Of bytes to be moved with the result of the
;Subtraction placed into locations $50, $51
9F90: LDY #0 ;Zero the index pointer
9F92: INC $50 ;Add 1 to the LSB of the number of bytes to be
;Moved
9F94: BNE $9F9A ;If the value is not equal to zero, then branch
9F96: INC $51 ;Add one to the MSB of the number of bytes to be
;Moved
9F98: BEQ $9FB2 ;If the value is equal to zero, then we are done
;Transferring the bytes
9F9A: LDA $24 ;Get the LSB of the address to move the program
9F9C: BNE $9FA0 ;To and if it is not equal to zero, branch
9F9E: DEC $25 ;Subtract one from the MSB
9FA0: DEC $24 ;Subtract one from the LSB
9FA2: LDA $26 ;Get the LSB of the address to move the program
9FA4: BNE $9FA8 ;To and if it is not equal to zero, branch
9FA6: DEC $27 ;Subtract one from the MSB
9FA8: DEC $26 ;Subtract one from the LSB
9FAA: JSR $03C0
9FAD: STA ($24),Y
9FAF: JMP $9F92 ;Continue until all the bytes have been moved
9FB2: CLC ;Clear the carry for addition
9FB3: LDA $1211 ;Get the MSB of TXTTOP
9FB6: ADC #$24 ;Add #$24 to point to $4000
9FB8: STA $1211 ;Save the value back
9FBB: LDA $2E ;Get the MSB of VARTAB
9FBD: ADC #$24 ;Add #$24 to point to $4000
9FBF: STA $2E ;Save the value back
9FC1: LDA $44 ;Get the MSB of DATPTR
9FC3: ADC #$24 ;Add #$24 to point to $4000
9FC5: STA $44 ;Save the value back
9FC7: JSR $4F4F ;Relink the BASIC lines
9FCA: JSR $4F82 ;Update the end of program pointers
9FCD: BIT $7F ;Test to see if the system is in the DIRECT mode
9FCF: BPL $9FFE ;And if it is, then branch
9FD1: LDX #$24 ;Get the offset to add to the MSB of TXTPTR and
;Other addresses to point to the new RAM area
9FD3: BIT $76 ;If the BIT MAP screen
9FD5: BMI $9FD9 ;Is allocated, then branch
9FD7: LDX #$DC ;Get the offset to add to the MSB of TXTPTR and
;Other addresses to point to the new RAM area
9FD9: TXA ;Move the offset into the accumulator
9FDA: CLC ;Clear the carry for addition
9FDB: ADC $3E ;Add the MSB of TXTPTR to the offset
9FDD: STA $3E ;And save it back
9FDF: TXA ;Move the offset to the accumulator

```

```

9FE0: CLC                ;Clear the carry for addition
9FE1: ADC    $1203        ;Add the offset
9FE4: STA    $1203        ;To next BASIC statement for CONT
9FE7: TXA                ;Move the offset back into the accumulator
9FE8: CLC                ;Clear the carry for addition
9FE9: ADC    $120F        ;Add the offset to the
9FEC: STA    $120F        ;Address of the line that generated the error
9FEF: JSR    $5047        ;Move the address of the pseudo stack
9FF2: LDA    $3F          ;The next section of code is used to see if
9FF4: CMP    #$FF         ;There are any addresses on the pseudo stack
9FF6: BNE    $9FFF        ;That need to be updated to the new location of
9FF8: LDA    $40          ;The BASIC text and if not, then the routine
9FFA: CMP    #$09        ;Will exit (the stack is empty if the pseudo
                        ;Stack pointer is pointing to $09FF)
9FFC: BNE    $9FFF        ;If there are addresses on the stack to be
                        ;Updated, then branch
9FFE: RTS                ;Exit the routine

*****
*
* This routine is used to update the pseudo stack      *
* after the graphics screen has been allocated or     *
* deallocated. Note: If you moved the start of your   *
* program to anywhere but $4000 or $1C00, you cannot  *
* use the BASIC graphic commands to use the graphics *
* screen. Also, use the DEF command after you allocate *
* a graphics area or the FN funtion will not work.    *
*
*****

9FFF: LDY    #$00        ;Zero the index to the first value on the
                        ;Pseudo stack
A001: LDA    ($3F),Y     ;Get a byte from the pseudo stack
A003: CMP    #$81        ;Is it the token for 'FOR'?
A005: BNE    $A010        ;If not, then it's 'GOSUB' or 'DO', branch
A007: LDY    #$10        ;Point to address to update (1 byte after 'FOR')
A009: JSR    $A062        ;Update the address
A00C: LDA    #$12        ;Update the pseudo stack pointer
A00E: BNE    $A017        ;To the next entry on the stack
A010: LDY    #$04        ;Point to the address to update (1 byte after
                        ;'GOSUB' or 'DO')
A012: JSR    $A062        ;Update the address
A015: LDA    #$05        ;Get the no. of bytes to the next pseudo stack
A017: CLC                ;Entry. If the previous entry was GOSUB or DO,
A018: ADC    $3F         ;Add that value to the LSB
A01A: STA    $3F         ;Of the pseudo stack pointer.
A01C: BCC    $9FF2        ;If the LSB did not overflow, then exit

```



```
A01E: INC    $40          ;If the LSB overflowed, then increment MSB of
A020: BNE    $9FF2       ;The pseudo stack pointer by one and exit
```

```
*-----*
*
* This routine is entered here by the GRAPHIC CLR
* command. If location $76 is equal to zero, then the
* HI-RES screen is deallocated.
*
*-----*
```

```
A022: LDA    $76          ;Check to see if graphics screen is allocated
A024: BNE    $A027       ;If it is, then deallocate it
A026: RTS
```

```
*****
*
* DEALLOCATE THE HIRES SCREEN
*
* Deallocate the HI-RES screen and move the BASIC
* text from $4000 to $1C00.
*
* NOTE: This is the reason why you cannot change the
* start of BASIC if you use the HI-RES screen.
*
*****
```

```
A027: LDY    #0           ;Clear the flag
A029: STY    $76          ;To denote that Graphics screen is deallocated
A02B: STY    $24          ;Clear two storage areas for
A02D: STY    $26          ;The BASIC start address LSB
A02F: LDA    #$1C        ;Get the MSB of where the BASIC text
A031: STA    $25          ;Must be moved to and save it
A033: LDA    #$40        ;Get the MSB of where the BASIC text
A035: STA    $27          ;Must be moved from and save it
A037: JSR    $03C0       ;Transfer all of the BASIC text
A03A: STA    ($24),Y     ;Starting from $4000
A03C: INY
A03D: BNE    $A037
A03F: INC    $25
A041: INC    $27
A043: LDA    $1211       ;Make sure that we have transferred
A046: CMP    $27          ;All of the BASIC text
A048: BCS    $A037       ;And continue if not.
A04A: SEC
A04B: LDA    $2E          ;If we are done moving the BASIC text
A04D: SBC    $2E          ;Subtract 36
A04D: SBC    #$24        ;From the MSB of the
```

```

A04F: STA   $2E           ;Start of BASIC text address and
A051: LDA   $1211        ;Subtract 36
A054: SBC   #$24         ;From the MSB of the
A056: STA   $1211        ;End of BASIC text address
A059: LDA   $44          ;And subtract 36
A05B: SBC   #$24         ;From the MSB of DATPTR
A05D: STA   $44
A05F: JMP   $9FC7        ;Relink lines, update end of program
                               ;pointers and exit.

```

```

*****
*
*   This routine is used to update the line addresses
*   on the pseudo stack after the graphics screen has
*   been allocated or deallocated.  If the graphics
*   screen is allocated, then 36 is added to all of the
*   MSBs to point to the BASIC program which starts at
*   $4000.  If the graphics screen is deallocated, then
*   36 is subtracted from all of the MSBs to point to
*   the BASIC program which starts at $1C00.
*
*   NOTE:  Since the BASIC variable storage area is not
*   checked or updated by this routine, if you are going
*   to use the graphics screens along with the DEFine
*   FuNction routine, you MUST define the function AFTER
*   you allocate the screen memory and after you
*   deallocate it or the DEFFN command will not work.
*
*****

```

```

A062: LDA   ($3F),Y
A064: BIT   $76
A066: BNE   $A06E
A068: SEC
A069: SBC   #$24
A06B: STA   ($3F),Y
A06D: RTS
A06E: CLC
A06F: ADC   #$24
A071: STA   ($3F),Y
A073: RTS

```

```
*****
*
*                                     *
*           CHKGRP                     *
*
* This routine checks to see if a graphics screen has *
* been allocated and if it has not, then a           *
* 'NO GRAPHICS AREA' error is generated.             *
*
*
*****
```

```
A074: LDA    $76           ;Check if the graphics screen is allocated
                               ;(the value is less than or greater than zero if
                               ;the graphics screen is allocated)
A076: BEQ    $A079        ;If the graphics screen is not allocated, then
                               ;Generate a 'NO GRAPHICS AREA' error
A078: RTS
A079: LDX    #35          ;Error number for the 'NO GRAPHICS AREA' error
A07B: JMP    $4D3C        ;Generate the error message
```

```
*****
*
*           BASIC CATALOG AND DIRECTORY COMMANDS    *
*
* Command syntax:  CATALOG [DR] [<ON,>DV] [,WC]     *
*                  DIRECTORY [DR] [<ON,>DV] [,WC]    *
*
* NOTE:  DR = 'D' + the drive #                      *
*        DV = 'U' + the device #                      *
*        WC = wildcard string                         *
*
* These commands allow you to see which files are in *
* the disk directory without disturbing what is in  *
* stored in memory.                                  *
*
*****
```

```
A07E: JSR    $A3BF        ;Clear DOS Work Area, parse the command, and note
                               ;types of parameters after the comma in $80, $81
A081: LDA    $80          ;Get the types of parameters that follow command
A083: AND    #%11100110  ;Check if any illegal parameters were used
A085: BEQ    $A08A        ;If there are none, then continue
A087: JMP    $796C        ;If any illegal parameters were used, generate a
                               ;'SYNTAX' error
A08A: LDY    #$01         ;Set up an index to the command table
A08C: LDX    #$01         ;Set up a counter to number of characters to get
                               ;From the table
A08E: LDA    $80          ;Get the types of parameters that follow command
```

A090: AND    %00010001 ;Check if there is a wildcard or drive # after  
  ;The command  
A092: BEQ    \$A09A       ;If not, then branch  
A094: LSR                     ;Check if there is a wildcard  
A095: BCC    \$A099       ;If not, then branch  
A097: INX                     ;Increment the counter by 3 to do a  
A098: INX                     ;Directory with  
A099: INX                     ;A wildcard  
A09A: TXA                    ;Save the counter  
A09B: JSR    \$A667       ;Set up the parameters for reading the directory  
A09E: LDA    #\$00       ;Set the memory bank to zero  
AOA0: TAX                    ;Set the filename bank to zero  
AOA1: JSR    \$9287       ;Call SETBNK to set up the memory configuration  
AOA4: LDY    #\$60       ;Set the secondary address to 0 + \$60  
AOA6: LDX    \$011C       ;Get the correct device #  
AOA9: LDA    #\$00       ;Set the logical file number to zero  
AOAB: JSR    \$9257       ;Call BASIC SETLFS to set up the file parameters  
AOAE: SEC                    ;Set the carry as a flag to check for errors  
AOAF: JSR    \$90D8       ;Open the file  
AOB2: BCC    \$A0BD       ;If there are no errors, then branch  
AOB4: PHA                    ;Save the error number onto the stack  
AOB5: JSR    \$A114       ;Reset the system to default I/O configuration  
  ;And close the file that was opened  
AOB8: PLA                    ;Get back the error number  
AOB9: TAX                    ;Move the error number to the X register  
AOBA: JMP    \$4D3C       ;Jump to process the error

```
*****  
*                                                   *  
*                   OUTDIR                       *  
*                                                   *  
*                   OUTput the DIRectory of a disk  *  
*                                                   *  
* This routine does the actual outputting of the   *  
* directory of a disk.  When the routine is called, a *  
* file must already be opened with a file number of *  
* zero and a name of '$'.  NOTE: This routine can be *  
* used to output any file to the screen just by   *  
* opening a file as explained above and substituting *  
* the '$' with the name of the file you want to   *  
* output.  The first four bytes will be skipped, and *  
* the next two will be printed as a decimal value. *  
*                                                   *  
*****
```

A0BD: LDX    #0           ;Set the X register to the Logical File number  
AOBF: JSR    \$A845       ;Enable BANK 15

```

A0C2: JSR   $FFC6      ;Designate the file as an input file
A0C5: LDY   #$03       ;Set the number of double bytes
A0C7: STY   $1174      ;To skip
A0CA: JSR   $9263      ;Get a byte from the file
A0CD: STA   $1175      ;And save it
A0D0: JSR   $9251      ;Call the BASIC READST routine to check status
                        ;of the disk drive
A0D3: BNE   $A114      ;If the status is not zero, the end of file was
                        ;Found or there was an error, then exit
A0D5: JSR   $9263      ;Get the next byte from the file
A0D8: STA   $1176      ;And save it
A0DB: JSR   $9251      ;Check the status again and exit
A0DE: BNE   $A114      ;If the status is not zero
A0E0: DEC   $1174      ;Decrement the counter by one
A0E3: BNE   $A0CA      ;Continue skipping until the counter is = to 0
A0E5: LDX   $1175      ;Get the LSB and
A0E8: LDA   $1176      ;The MSB of the number of blocks in the file
A0EB: JSR   $8E32      ;Output the number to the screen
A0EE: LDA   #$20       ;Output a space
A0F0: JSR   $9269      ;To the screen
A0F3: JSR   $9263      ;Get a character from the filename
A0F6: PHA                   ;And save it
A0F7: JSR   $9251      ;Check the status
A0FA: BNE   $A113      ;If the status is not zero, then exit
A0FC: PLA                   ;Get the character back
A0FD: BEQ   $A105      ;If 0, this is the end of the filename -branch
A0FF: JSR   $9269      ;Output the character of the filename
A102: JMP   $A0F3      ;And continue with the loop
A105: LDA   #$0D       ;Output a carriage return
A107: JSR   $9269      ;To the screen
A10A: JSR   $9293      ;Check the stop key
A10D: BEQ   $A114      ;And exit if it is pressed
A10F: LDY   #$02       ;Set the number of double bytes to skip
A111: BNE   $A0C7      ;And branch to output the next filename
A113: PLA                   ;Get the accumulator back off of the stack
A114: JSR   $926F      ;Call the BASIC CLRCHN to reset the default I/O
                        ;Configuration
A117: LDA   #$00       ;Get the file number to close
A119: CLC                   ;Clear the error flag
A11A: JMP   $9275      ;Close the file and exit

```

```

*****
*
*          BASIC DOPEN COMMAND          *
*
* Command syntax:  DOPEN# file number," filename "
*                  [,<S/P>]"[,RL][,DR][<ON,>DV][,W]
*

```

```

* NOTE: S = sequential file
*       P = program file
*       RL = 'L' + the file length (relative files)
*       DR = 'D' + the drive #
*       DV = 'U' + the device #
*       W = 'W' for the write mode
*
* This command allows you to open a disk file for a
* read or write operation.
*
*****

```

```

A11D: LDA  #\$22      ;Parse the command and ensure that a
A11F: JSR  \$A3C1     ;Second filename or second drive # was unused
A122: JSR  \$A76F     ;Ensure that a logical file no. and filename
                    ;Were specified
A125: JSR  \$A157     ;Find an available secondary address and save
                    ;It at location \$011D
A128: LDY  #\$05     ;Index to the command table minus one
A12A: LDX  #\$04     ;Number of characters in the command
A12C: BIT  \$80       ;Check if an 'L' for record length was specified
A12E: BVC  \$A143     ;If not, then branch
A130: LDX  #\$08     ;Number of characters in the command
A132: BNE  \$A143     ;Branch to execute the command

```

```

*****
*
*           BASIC APPEND COMMAND
*
* Command syntax: APPEND# file number," filename "
*                 [,DR][,<ON,>DV]
*
* NOTE: DR = 'D' + the drive #
*       DV = 'U' + the device #
*
* This command allows you to open a file and any
* PRINT# commands will append the data specified to
* the end of the file specified by 'filename'.
*
*****

```

```

A134: LDA  #\$E2      ;Parse the command & ensure that a 2nd filename,
                    ;Record length,
A136: JSR  \$A3C1     ;Second drive #, or '@' symbol was not used
A139: JSR  \$A76F     ;Ensure that a filename and logical file no. was
                    ;Specified
A13C: JSR  \$A157     ;Search for an unused secondary address

```

```

A13F: LDY  #$16      ;Set up an index to the command table minus one
A141: LDX  #$05      ;Set up the number of characters
A143: TXA                      ;In the command
A144: JSR  $A667      ;Place the proper DOS command into the
                        ;Buffer at starting at location $1100
A147: JSR  $926F      ;Reset the system to default I/O configuration
A14A: LDA  #$00      ;Get the bank number of the
A14C: TAX                      ;DOS command
A14D: JSR  $9287      ;Set the bank number for this operation
A150: JSR  $90D8      ;Open the file
A153: SEC                      ;Flag to designate a special CLOSE
A154: JMP  $9275      ;Close down the file in the computer only.

```

```

*****
*
* This routine is to find an unused secondary address. *
*
*****

```

```

A157: LDY  #$61      ;Get the first secondary address to lookup + $60
A159: INY                      ;Add one to the secondary address
A15A: CPY  #$6F      ;Is it the last available secondary address+1?
A15C: BEQ  $A16A      ;Yes, then generate a 'TOO MANY FILES' error
A15E: JSR  $A845      ;Enable BASIC BANK 15
A161: JSR  $FF5C      ;See if the secondary address is in use
A164: BCC  $A159      ;If it is, then branch to check the next one
A166: STY  $011D      ;If not, then save it
A169: RTS                      ;And exit

```

```

*****
*
* This routine generates a 'TOO MANY FILES' error. *
*
*****

```

```

A16A: LDX  #1          ;Error number for 'TOO MANY FILES' error
A16C: JMP  $4D3C      ;Generate the error message

```

```

*****
*
* BASIC DCLOSE COMMAND *
*
* Command syntax: DCLOSE [# file number][ON,DV] *
*
* NOTE: DV = 'U' + the device # *

```

```

* This command closes a single file or all files on *
* the disk drive that is specified. *
* *
*****

```

```

A16F: LDA  #$F3      ;Parse the command
A171: JSR  $A3C1     ;No parameters but a logical file no. and/or
                   ;drive # may be specified
A174: JSR  $A80D     ;Deallocate DS$, set the flag to read a new DS$
A177: LDA  $80       ;Check if a drive #
A179: AND  #$04      ;Was specified after the command
A17B: BEQ  $A183     ;If not, then branch
A17D: LDA  $011B     ;Get the current logical file number
A180: JMP  $9275     ;And CLOSE it
A183: LDA  $011C     ;Get the current device #
A186: JSR  $A845     ;Enable BANK 15
A189: JMP  $FF4A     ;And CLOSE all of the open files on that device

```

```

*****
*
*                               BASIC DSAVE COMMAND
*
* Command syntax:  DSAVE " filename " [,DR][<ON,>DV]
*
* NOTE:  DR = 'D' + the drive #
*        DV = 'U' + the device #
*
* This command will allow you to save a BASIC program
* file to a disk.
*
*****

```

```

A18C: LDA  #$66      ;Parse command & ensure that the only parameters
A18E: JSR  $A3C1     ;Are the filename, the device #, and the drive
                   ;Number, or the save with replace '@'
A191: JSR  $A750     ;Ensure that a filename is after the command
A194: LDY  #$05      ;Set up an index to the command table minus one
A196: LDA  #$04      ;Set up the number of characters in the command
A198: JSR  $A667     ;Place the proper command in the buffer at $1100
A19B: LDA  #$00      ;Set up the RAM BANK to save from
A19D: TAX
A19E: JSR  $9287     ;Call SETBNK to select the RAM BANK
A1A1: JMP  $9115     ;Perform a BASIC SAVE

```



```

*****
*
*                               *
*          BASIC DVERIFY COMMAND *
*                               *
* Command syntax:  DVERIFY " filename " [,DR][<ON,>DV] *
*                               *
* NOTE:  DR = 'D' + the drive # *
*        DV = 'U' + the device # *
*                               *
* This command allows you to verify the program that *
* is on the disk with the program that is stored in *
* memory.  This command is usually used to make sure *
* that a BASIC program was saved to the disk properly. *
*                               *
*****

```

```

A1A4: LDA    #1          ;Set the flag for Verify
A1A6: .BYTE $2C        ;Mask to fall through to the DLOAD command

```

```

*****
*
*                               *
*          BASIC DLOAD COMMAND *
*                               *
* Command syntax:  DLOAD " filename " [,DR][,DV] *
*                               *
* NOTE:  DR = 'D' + the drive # *
*        DV = 'U' + the device # *
*                               *
* This command allows you to load a BASIC program *
* from disk into the computer's memory at the current *
* starting address of BASIC. *
*                               *
*****

```

```

A1A7: LDA    #00          ;Set the flag for LOAD
A1A9: STA    $0C          ;Save the LOAD/VERIFY Flag
A1AB: LDA    #$E6          ;Parse the command
A1AD: JSR    $A3C1        ;No other parameters but the filename, device #,
                        ;Or drive # may be specified
A1B0: JSR    $A750        ;Ensure that a filename is after the command
A1B3: LDA    #$00          ;Set the current secondary
A1B5: STA    $011D        ;Address to $00
A1B8: LDY    #$05          ;Set up an index to the command table minus one
A1BA: LDA    #$04          ;Set up the number of characters in the command
A1BC: JSR    $A667        ;Put the proper command in the buffer at $1100
A1BF: LDA    #$00          ;Set the Bank number for LOAD
A1C1: TAX                      ;To Bank 0

```

```
A1C2: JSR   $9287      ;Call SETBNK to select RAM BANK
A1C5: JMP   $9133      ;Perform a BASIC LOAD
```

```
*****
*
*          BASIC BSAVE COMMAND          *
*
* Command syntax: BSAVE " filename " [,DR][,DV]
*                  [,BK],SA <TO> EA    *
*
* NOTE:  DR = 'D' + the drive #        *
*        DV = 'U' + the device #       *
*        BK = 'B' + the bank number    *
*        SA = 'P' + the starting address for the save *
*        EA = 'P' + the ending address for the save *
*
* This command allows you to save a binary file to *
* disk of the memory area that is specified.      *
*
*****
```

```
A1C8: LDA   #$66      ;Parse the command
A1CA: LDX   #$F8      ;No other parameters but filename, the device
                        ;Number, the drive #, the save and replace '@',
A1CC: JSR   $A3C3      ;The bank #, and the start and end addresses,
                        ;May be specified
A1CF: JSR   $A750      ;Ensure that a filename follows the command
A1D2: LDA   $81       ;Mask appropriate bits for start / end addresses
A1D4: AND   #$06      ;And check if both
A1D6: CMP   #$06      ;Are specified
A1D8: BEQ   $A1DD      ;If they are specified, then branch
A1DA: JMP   $796C      ;If unspecified, then generate a 'SYNTAX' error
A1DD: LDA   $011A      ;If the starting
A1E0: CMP   $0118      ;And ending addresses
A1E3: BCC   $A215      ;Are the same
A1E5: BNE   $A1F1      ;Or if the
A1E7: LDA   $0119      ;Starting address is
A1EA: CMP   $0117      ;Larger than
A1ED: BCC   $A215      ;The ending address,
A1EF: BEQ   $A215      ;Then error
A1F1: LDY   #$05      ;Set up an index to the command table minus one
A1F3: LDA   #$04      ;Set up the number of characters in the command
A1F5: JSR   $A667      ;Put the proper command in the buffer at $1100
A1F8: LDA   $011F      ;Get the bank number for the save
A1FB: LDX   #$00
A1FD: JSR   $9287      ;Call SETBNK to select the RAM BANK
A200: LDX   $0117      ;Get the LSB
```

```

A203: LDY   $0118      ;And the MSB of the save starting address
A206: LDA   #$5A       ;Set the accumulator to the address of where
                        ;The starting address is saved
A208: STX   $5A        ;Save the starting address
A20A: STY   $5B        ;Of the save
A20C: LDX   $0119      ;Get the LSB
A20F: LDY   $011A      ;And the MSB of the ending address for the SAVE
A212: JMP   $911D      ;And SAVE the file
A215: JMP   $7D28      ;Generate an 'ILLEGAL QUANTITY' error

```

```

*****
*
*           BASIC BLOAD COMMAND           *
*
* Command syntax:  BLOAD " filename " [,DR][,DV]
*                  [,BK][,SA]            *
*
* NOTE:  DR = 'D' + the drive #          *
*         DV = 'U' + the device #        *
*         BK = 'B' + the bank number     *
*         SA = 'P' + the starting address for the load *
*
* This command allows you to load a binary file from *
* the disk to the memory area that is specified.    *
*
*****

```

```

A218: LDA   #$E6       ;Ensure that the only parameters specified are
A21A: LDX   #$FC       ;The filename, the device #, the drive #,
                        ;And the save with replace '@'
A21C: JSR   $A3C3      ;Bank number of the starting address
A21F: JSR   $A750      ;Ensure that a filename follows the command
A222: LDX   $0117      ;Get the LSB
A225: LDY   $0118      ;And the MSB of the starting address for LOAD
A228: LDA   #$00       ;Set the initial secondary address to zero
A22A: CPX   #$FF       ;If the starting address
A22C: BNE   $A234      ;Has not been specified,
A22E: CPY   #$FF       ;Then set the current
A230: BNE   $A234      ;Secondary address
A232: LDA   #$FF       ;To $FF. If the starting address was specified
A234: STA   $011D      ;Then set the secondary address to zero
A237: LDY   #$05       ;Set up an index to the command buffer minus one
A239: LDA   #$04       ;Set up the number of characters in the command
A23B: JSR   $A667      ;Put the proper command into the buffer at $1100
A23E: LDA   $011F      ;Get the the bank number to LOAD to
A241: LDX   #$00       ;And call SETBNK
A243: JSR   $9287      ;To set the RAM BANK number

```

```

A246: LDA    #$00        ;Set the flag for LOAD
A248: LDX    $0117      ;Get the LSB
A24B: LDY    $0118      ;And the MSB of the starting address
A24E: JSR    $FFD5      ;LOAD the file
A251: PHP                    ;Save the error flag
A252: JSR    $9243      ;Deallocate DS$
A255: PLP                    ;Restore the error flag
A256: BCC    $A25B      ;If there was no error, then branch
A258: JMP    $90D0      ;Generate an error, and exit.
A25B: JSR    $9251      ;Read the current I/O status
A25E: AND    #$BF       ;If there were no errors,
A260: BEQ    $A265      ;Then branch
A262: JMP    $9167      ;Output the error message
A265: CLC                    ;Clear the error flag
A266: RTS                    ;And exit

```

```

*****
*
*                               *
*          BASIC HEADER COMMAND *
*                               *
* Command syntax: HEADER "diskname" [,ID] [,DR] [<ON,>DV] *
*                               *
* NOTE: ID = 'I' + a two character identification *
*         DR = 'D' + the drive # *
*         DV = 'U' + the device # *
*                               *
* This command allows you to format a disk for data *
* storage with the 'diskname' and 'ID' specified. *
*                               *
*****

```

```

A267: JSR    $A3BF        ;Parse the command
A26A: JSR    $A749        ;Ensure that no other parameters but filename,
                        ;The device #, or the drive # were specified
A26D: AND    #$01        ;Ensure that there is a filename after
A26F: CMP    #$01        ;The command
A271: BNE    $A2D4        ;If not, then generate a 'SYNTAX' error
A273: JSR    $927B        ;Make the screen and keyboard the current I/O
A276: JSR    $A7E1        ;Ask the question 'ARE YOU SURE?'
A279: BNE    $A2A0        ;Abort if an 'N' was entered
A27B: LDY    #$1B        ;Set up a pointer to the command table minus one
A27D: LDA    #$04        ;Number of chars in a command for a "short" NEW
A27F: LDX    $0120        ;Check if an ID was specified
A282: BEQ    $A286        ;If not, then perform a "short" NEW
A284: LDA    #$06        ;Number of chars in a command for a "full" NEW
A286: JSR    $A397        ;Format the disk
A289: JSR    $A778        ;Read in DS$ from the disk drive

```

```

A28C: BIT    $7F          ;Check if we are in the PROGRAM mode
A28E: BMI    $A2A0        ;If so, then exit
A290: LDY    #0           ;Index to the first character of DS$
A292: LDA    #$7B        ;Pointer to address of DS$
A294: JSR    $03AB        ;Get the first character of DS$
A297: CMP    #'2'        ;If the error number is
A299: BCC    $A2A0        ;Less than 20, then exit
A29B: LDX    #36         ;Error number for 'BAD DISK' error
A29D: JMP    $4D3C        ;Generate the error message
A2A0: RTS                ;Exit the routine

```

```

*****
*
*                               BASIC SCRATCH COMMAND
*
* Command syntax: SCRATCH " filename " [,DR] [,DV]
*
* NOTE: DR = 'D' + the drive #
*        DV = 'U' + the device #
*
* This command allows you to scratch a file or files
* from a disk.
*
*****

```

```

A2A1: JSR    $A3BF        ;Parse the command
A2A4: JSR    $A749        ;Ensure that no other parameters but filename,
                        ;The device #, or the drive # were specified
A2A7: JSR    $A7E1        ;Ask the question 'ARE YOU SURE?'
A2AA: BNE    $A2D3        ;Abort if any character other than 'Y' is
                        ;entered
A2AC: LDY    #$37         ;Set up a pointer to the command table minus one
A2AE: LDA    #$04         ;Number of characters in the command
A2B0: JSR    $A397        ;SCRATCH the file
A2B3: JSR    $A778        ;Read in DS$ from the disk drive
A2B6: BIT    $7F          ;Check if we are in the Program mode
A2B8: BMI    $A2D3        ;If so, then exit
A2BA: LDA    #$0D         ;Value for a carriage return
A2BC: JSR    $9269        ;Output it
A2BF: LDY    #$00         ;Index to the first character of DS$
A2C1: LDA    #$7B        ;Point to the address of DS$
A2C3: JSR    $03AB        ;If there is no error message,
A2C6: BEQ    $A2CE        ;Then exit
A2C8: JSR    $9269        ;Output the entire
A2CB: INY                ;Error
A2CC: BNE    $A2C1        ;message
A2CE: LDA    #$0D         ;Output a

```

```

A2D0: JSR   $90DF      ;Carriage return
A2D3: RTS                               ;And exit the routine
A2D4: JMP   $796C      ;Generate a 'SYNTAX' error

```

```

*****
*
*                               BASIC RECORD COMMAND                               *
*
* Command syntax: RECORD# LFN, RN [,BN]                                         *
*
* NOTE: LFN = the logical file number                                           *
*        RN  = the record number                                                *
*        BN  = the byte number                                                  *
*
* This command allows you to access a relative file                             *
* by setting the pointers to that file.                                         *
*
*****

```

```

A2D7: LDA   #$23      ;Value for a pound sign '#'
A2D9: JSR   $795E      ;Check for pound sign. If the pound sign is not
                       ;Found, then generate a 'SYNTAX' error
A2DC: JSR   $87F4      ;Get the file number into the X register
A2DF: CPX   #$00      ;If the file number is equal to zero
A2E1: BEQ   $A31A      ;Then generate an 'ILLEGAL QUANTITY' error
A2E3: STX   $011B      ;Save the file number in the DOS work area
A2E6: JSR   $880F      ;Check for a comma and get the record number
A2E9: LDX   #$01      ;Set the default offset to 1
A2EB: JSR   $9E1E      ;Get byte # (offset in record) into X register
A2EE: CPX   #$00      ;If the byte number is zero
A2F0: BEQ   $A31A      ;Then generate an 'ILLEGAL QUANTITY' error
A2F2: CPX   #$FF      ;If the byte number is 255
A2F4: BEQ   $A31A      ;Then generate an 'ILLEGAL QUANTITY' error
A2F6: STX   $011E      ;Save the byte number in DOS work area
A2F9: LDA   $011B      ;Get the special logical file number
A2FC: JSR   $A845      ;Enable BANK 15
A2FF: JSR   $FF59      ;Search for the logical file number
A302: BCS   $A31D      ;And if the file is not open,
                       ;Then generate a 'FILE NOT FOUND' error
A304: STY   $11ED      ;Save the secondary address
A307: STX   $011C      ;And the device #
A30A: LDA   #$00      ;Set the logical file number to zero
A30C: STA   $011B      ;Save it in the DOS work area
A30F: LDA   #$6F      ;Set the secondary address to 15 + $60
A311: STA   $011D      ;Save it in the DOS work area
A314: LDY   #$3B      ;Set up an index to the command table minus one
A316: LDA   #$04      ;Number of characters in the command

```

```

A318: BNE   $A397      ;Execute the command
A31A: JMP   $7D28      ;Generate an 'ILLEGAL QUANTITY' error
A31D: LDX   #$04       ;Error number of 'FILE NOT FOUND' error
A31F: JMP   $4D3C      ;Generate the error message

```

```

*****
*
*                               *
*          BASIC DCLEAR COMMAND *
*                               *
* Command syntax:  DCLEAR [,DR] [<ON,>DV] *
*                               *
* NOTE:  DR = 'D' + the drive # *
*        DV = 'U' + the device # *
*                               *
* This command when executed will intialize the disk *
* drive that is specified. *
*                               *
*****

```

```

A322: JSR   $A3BF      ;Parse the command
A325: LDY   #$FF       ;Index to the command table minus one
A327: LDA   #$02       ;Number of characters in the command
A329: JSR   $A397      ;Execute the DOS command
A32C: JMP   $A183      ;Close all files and exit

```

```

*****
*
*                               *
*          BASIC COLLECT COMMAND *
*                               *
* Command syntax:  COLLECT [,DR] [<ON,>DV] *
*                               *
* NOTE:  DR = 'D' + the drive # *
*        DV = 'U' + the device # *
*                               *
* This command allows you to free up the unused space *
* on a disk. This process is called validating. *
*                               *
*****

```

```

A32F: JSR   $A3BF      ;Parse the command
A332: JSR   $A75B      ;Ensure that parameters used are the device #
                          ;Or the drive #
A335: JSR   $927B      ;Close all open files and make the screen and
                          ;keyboard current I/O
A338: LDY   #$21       ;Index to the command table
A33A: LDX   #$01       ;Number of characters in the command
A33C: LDA   $80        ;Check if a drive #

```

```

A33E: AND   #$10      ;Was specified after the command
A340: BEQ   $A343     ;If not, then default to zero
A342: INX                   ;Increment the number of chars in the command
                          ;To include the drive #
A343: TXA                   ;Move that value to the accumulator
A344: BNE   $A397     ;Execute the command
    
```

```

*****
*
*                               BASIC COPY COMMAND
*
* Command syntax: COPY "source filename" [,DR] TO
*                 "destination filename" [,DR] [<ON,>DV]
*
* NOTE:  DR = 'D' + the drive #
*        DV = 'U' + the device #
*
* This command allows you to copy a file within a
* single disk drive or from one disk to another in a
* dual disk drive.
*
*****
    
```

```

A346: JSR   $A3BF     ;Parse the command
A349: AND   #$30     ;Check if the copy is to be performed
A34B: CMP   #$30     ;Between Drive 1 and Drive 0
A34D: BNE   $A355     ;If not, then branch
A34F: LDA   $80      ;If there are not any
A351: AND   #$C7     ;Illegal parameters,
A353: BEQ   $A35C     ;Then branch
A355: LDA   $80      ;Ensure that two filenames follow the
A357: JSR   $A760     ;Command, check for illegal parameters
A35A: LDA   $80      ;Useless code
A35C: LDY   #$27     ;Index to the command table minus one
A35E: LDA   #$08     ;Number of characters in the command
A360: BNE   $A397     ;Execute the command
    
```

```

*****
*
*                               BASIC CONCAT COMMAND
*
* Command syntax:  CONCAT "file #2" [,DR] TO "file #1"
*                 [,DR] [<ON,>DV]
*
* NOTE:  DR = 'D' + the drive number
*        DV = 'U' + the device number
*
*****
    
```



```

* This command allows you to combine two files on disk *
* into one file on the disk. The filename that is *
* given to the new file is the one that was specified *
* for 'file #1'. *
* *
*****

```

```

A362: JSR   $A3BF      ;Parse the command
A365: JSR   $A760      ;Ensure that the two filenames are specified
                        ;after the command
A368: LDY   #$0D       ;Index to the command table minus one
A36A: LDA   #$0C       ;Number of characters in the command
A36C: BNE   $A397      ;Execute the command

```

```

*****
*
*                               BASIC RENAME COMMAND
*
* Command syntax:  RENAME "old" TO "new" [,DR][,DV]
*
* NOTE:  old = the filename of the file that you wish
*         to RENAME
*        new = the filename that is to be assigned
*         to the file being RENAMED
*        DR = 'D' + the drive number
*        DV = 'U' + the device number
*
* This command allows you to RENAME a file from the
* old filename specified by 'old' to the new filename
* specified by 'new'.
*
*****

```

```

A36E: LDA   #$E4       ;Parse the command and ensure that
A370: JSR   $A3C1      ;There are no illegal parameters present
A373: JSR   $A766      ;Ensure that 2 filenames follow the command
A376: LDY   #$2F       ;Index to the command table minus one
A378: LDA   #$08       ;Number of characters in command
A37A: BNE   $A397      ;Execute the command

```

```

*****
*
*                               BASIC BACKUP COMMAND
*
* Command syntax:  BACKUP SDR to DDR [<ON,>DV]
*

```

```

* NOTE:  SDR = 'D' + the source drive number      *
*         DDR = 'D' + the destination drive number *
*         DV  = 'U' + the device number           *
*                                                 *
* This command allows you to copy the contents of one *
* disk on to another on a dual disk drive only.     *
*                                                 *
*****

```

```

A37C: LDA  #$C7      ;Parse the command and ensure that
A37E: JSR  $A3C1     ;There are no illegal parameters present
A381: AND  #$30      ;Ensure that two drive
A383: CMP  #$30      ;Numbers were specified
A385: BEQ  $A38A     ;If two were specified, then branch
A387: JMP  $796C     ;Generate a 'SYNTAX' error
A38A: JSR  $A7E1     ;Ask the question 'ARE YOU SURE?'
A38D: BEQ  $A390     ;Branch if 'Y' was entered
A38F: RTS           ;Exit the routine
A390: JSR  $A183     ;Close all open files
A393: LDY  #$23      ;Index to the command table minus one
A395: LDA  #$04      ;Number of characters in the command
A397: JSR  $A667     ;Put the proper command into the buffer at $1100
A39A: JSR  $926F     ;CLRCHN - Set system to default I/O
A39D: LDA  #$00      ;Set up for RAM BANK 0
A39F: TAX           ;And call SETBNK
A3A0: JSR  $9287     ;To set I/O for RAM BANK 0
A3A3: SEC           ;Useless code
A3A4: JSR  $90D8     ;Open the file to execute the BACKUP command
A3A7: PHP           ;Save the error flag
A3A8: PHA           ;And the error number
A3A9: LDA  $011B     ;Get the logical file number flag to perform
A3AC: SEC           ;Internal close (don't send close to disk)
A3AD: JSR  $9275     ;And close it
A3B0: PLA           ;Get the error number back
A3B1: PLP           ;And the error flag
A3B2: BCS  $A3B5     ;If there was an error, then branch
A3B4: RTS           ;If not, then exit
A3B5: JMP  $90D0     ;Generate the appropriate error message

```

```

*****
*
*           DOS INITIALIZATION DATA TABLE
*
*****

```

```

A3B8: .BYTE $FF,$FF ;Default starting address
A3BA: .BYTE $FF,$FF ;Default ending address

```

A3BC: .BYTE \$00 ;Default logical file number of zero  
A3BD: .BYTE \$08 ;Default device number of 8  
A3BE: .BYTE \$6F ;Default secondary address of 15 + \$60

\*\*\*\*\*  
\*  
\* BDCPP \*  
\*  
\* Basic DOS Command Parameter Parser \*  
\*  
\* This routine is used to set up which parameters \*  
\* are allowed to be entered after each of the DOS \*  
\* commands. This routine will also clear out the \*  
\* DOS buffer and copy the default DOS table into \*  
\* the DOS work area. Note: There are several entry \*  
\* points into this routine due to the fact that \*  
\* there are so many different combinations \*  
\* available. It is better to start at the beginning \*  
\* of the DOS command and walk through it instead of \*  
\* starting here. \*  
\*  
\*\*\*\*\*

A3BF: LDA #\$00 ;Flag to enable any parameters to be used except  
;Those specified in the next line  
A3C1: LDX #\$FF ;Flag to indicate the BANK #, starting and/or  
;Ending cannot be specified  
A3C3: PHA ;Save them  
A3C4: TXA ;Onto the  
A3C5: PHA ;Stack  
A3C6: LDA #0 ;Clear the two bytes that are reserved for the  
A3C8: STA \$80 ;Bit representation of the parameters  
A3CA: STA \$81 ;That follow any BASIC DOS command  
A3CC: LDX #\$22 ;For 34 bytes to clear  
A3CE: STA \$0100,X ;Clear the DOS work area  
A3D1: DEX ;Starting from \$0101  
A3D2: BNE \$A3CE ;To \$0122  
A3D4: LDX #6 ;Transfer the default  
A3D6: LDA \$A3B8,X ;Values  
A3D9: STA \$0117,X ;Into the  
A3DC: DEX ;DOS  
A3DD: BPL \$A3D6 ;Work area  
A3DF: LDX \$03D5 ;Save the BANK number  
A3E2: STX \$011F ;For BLOAD/BSAVE  
A3E5: JSR \$0386 ;Get 1st character after BASIC DOS command token  
A3E8: BNE \$A3F8 ;If there are parameters, then branch to process  
A3EA: PLA ;Get the parameter off of the stack

```

A3EB:  AND  $81      ;Compare it to the parameter that is specified
A3ED:  BNE  $A45A    ;If not equal, then generate a 'SYNTAX' error
A3EF:  PLA                      ;Get the parameter off of the stack
A3F0:  JSR  $A61D    ;Check to see if the parameter has been used yet
                        ;And if so, then generate a 'SYNTAX' error
A3F3:  LDA  $80      ;Load the accumulator and X register
A3F5:  LDX  $81      ;With the user specified parameters
                        ;(For more information, see locations $80, $81
                        ;of a zero page listing)
A3F7:  RTS                      ;Exit the routine

```

```

*****
*
* This routine checks for the various parameters and
* if the parameter exists, the proper subroutine to
* process that parameter will be called.
*
*****

```

```

A3F8:  CMP  #$23      ;Is it the pound sign '#'?
A3FA:  BEQ  $A447    ;Yes, then branch to process the file number
A3FC:  CMP  #$57      ;Is it a 'W' for write?
A3FE:  BEQ  $A45D    ;Yes, then branch
A400:  CMP  #$4C      ;Is it an 'L' for record length?
A402:  BEQ  $A45D    ;Yes, then branch
A404:  CMP  #$52      ;Is it an 'R' for read?
A406:  BEQ  $A431    ;Yes, then branch
A408:  CMP  #$44      ;Is it a 'D' for drive number?
A40A:  BEQ  $A47F    ;Yes, then branch to process the drive number
A40C:  CMP  #$91      ;Is it the token for 'ON'?
A40E:  BEQ  $A437    ;Yes, then branch to process the parameter
                        ;After the 'ON'
A410:  CMP  #$42      ;Is it a 'B' to designate the BANK number?
A412:  BEQ  $A442    ;Yes, then branch to process the BANK number
A414:  CMP  #$55      ;Is it a 'U' to designate the device number?
A416:  BEQ  $A43D    ;Yes, then branch to process the device number
A418:  CMP  #$50      ;Is it a 'P' to designate a starting or
                        ;Ending address?
A41A:  BNE  $A41F    ;If not, then branch
A41C:  JMP  $A4B4    ;Process the address
A41F:  CMP  #$49      ;Is it an 'I' to designate an ID for
                        ;The 'HEADER' command?
A421:  BEQ  $A498    ;Yes, then branch to process the ID
A423:  CMP  #$22      ;Is it a quote mark?
A425:  BEQ  $A42E    ;Yes, then branch to process the string
A427:  CMP  #$28      ;Is it an opening parenthesis?
A429:  BEQ  $A42E    ;Yes, then branch to process the string

```

```
*****
*
*           'SYNTAX' error generator
*
*****

A42B:  JMP    $796C    ;Generate a 'SYNTAX' error

*****
*
* This routine is called to process the string that is *
* in quotes or parentheses after a BASIC DOS command. *
*
*****

A42E:  JMP    $A4DC    ;Process the string in quotes or parentheses

*****
*
* This routine is used to move to the next character *
* in the command and process it.
*
*****

A431:  JSR    $0380    ;Get the next character
A434:  JMP    $A4FB    ;And process it

*****
*
* This routine is called to process the parameter *
* that follows the token for 'ON'.
*
*****

A437:  JSR    $A582    ;Process the parameter after the 'ON'
A43A:  JMP    $A4F7    ;Continue with the next parameter, if any

*****
*
* This routine is used to get the device number that *
* is specified after the ' U ' and store it in the DOS *
* work area at $011C. This routine then continues *
* with the routine that parses the command.
*
*****
```

```

A43D: JSR   $A58D      ;Get the device number and store it into the DOS
                        ;Work area
A440: BNE   $A43A      ;Branch back to the Main Loop
A442: JSR   $A59E      ;Get the BANK #, store it into DOS work area
A445: BEQ   $A43A      ;Branch back to the Main Loop
A447: LDA   #$04       ;Check if the logical file number
A449: JSR   $A61D      ;Parameter has been used yet
A44C: JSR   $A5F2      ;Get the logical file number into the X register
A44F: CPX   #$00       ;If the logical file number is zero
A451: BEQ   $A495      ;Then generate an 'ILLEGAL QUANTITY' error
A453: STX   $011B      ;Save the logical file number into DOS work area
A456: LDA   #$04       ;Branch to get the proper bit (bit 3)
A458: BNE   $A43A      ;For the logical file number
A45A: JMP   $796C      ;Generate a 'SYNTAX' error

```

```

*****
*
*   This routine processes the 'W' or 'L' parameter.
*
*****

```

```

A45D: TAX                       ;Save the 'W' or 'L' into the X register
A45E: LDA   #$40                 ;Make sure that the parameter
A460: JSR   $A61D                 ;Has not been used yet
A463: CPX   #$57                 ;If the parameter
A465: BNE   $A46D                 ;Is an 'L', then branch
A467: JSR   $0380                 ;Get the next character after the 'W'
A46A: JMP   $A47B                 ;Set proper bit in location $80 to specify this
                        ;Parameter was used (Bit 6)

```

```

*****
*
*   This routine is used to get the record length that
*   is specified after the parameter 'L' and process it.
*
*****

```

```

A46D: JSR   $A5F2      ;Get the record length into the X register
A470: CPX   #$00       ;If the record length is zero,
A472: BEQ   $A495      ;Then generate an 'ILLEGAL QUANTITY' error
A474: CPX   #$FF       ;If the record length is 255
A476: BEQ   $A495      ;Then generate an 'ILLEGAL QUANTITY' error
A478: STX   $011E      ;Save the record length into the DOS work area
A47B: LDA   #$40       ;And set the proper bit in location $80
A47D: BNE   $A493      ;For this parameter (Bit 6)

```

```
*****
*
* This routine is used to get the drive number that
* is specified after the 'D'.
*
*****
```

```
A47F: LDA    #$10      ;Check if this parameter has been used yet
A481: JSR    $A61D     ;And if it has, then generate a 'SYNTAX' error
A484: JSR    $A5F2     ;Get the drive number into the X register
A487: CPX    #$02     ;If the drive number is greater than one, then
A489: BCS    $A495     ;Branch to generate 'ILLEGAL QUANTITY' error
A48B: STX    $0112     ;Save the drive number
A48E: STX    $0114     ;To the DOS work area
A491: LDA    #$10     ;And get the proper bit in location $80
A493: BNE    $A4F7     ;For this parameter (Bit 4)
A495: JMP    $7D28     ;Generate an 'ILLEGAL QUANTITY' error
```

```
*****
*
* This routine is used to get the ID value that is
* specified after the 'I' parameter.
*
*****
```

```
A498: LDA    $0122     ;Check if the ID has already been defined
A49B: BNE    $A45A     ;And if it has, then generate a 'SYNTAX' error
A49D: JSR    $0380     ;Get the first character of the ID
A4A0: STA    $0120     ;Save it to the DOS work area
A4A3: JSR    $0380     ;Get the second character of the ID
A4A6: STA    $0121     ;Save it to the DOS work area
A4A9: LDA    #$FF     ;Set the flag for
A4AB: STA    $0122     ;ID defined
A4AE: JSR    $0380     ;Get the next character
A4B1: JMP    $A4FB     ;And process it
```

```
*****
*
* This routine is used to get the starting address
* that is specified after the 'P'.
*
*****
```

```
A4B4: LDA    #$02     ;Check if the parameter has been used yet
A4B6: JSR    $A622     ;And if it has, then generate a 'SYNTAX' error
A4B9: JSR    $A605     ;Get starting address into the Y register and
                        ;The accumulator
```

```

A4BC: STY $0117 ;Save the LSB and the MSB
A4BF: STA $0118 ;Of the starting address into the DOS work area
A4C2: LDA #$02 ;Set the proper bit
A4C4: ORA $81 ;In location $81
A4C6: STA $81 ;For this parameter (BIT 1)
A4C8: BNE $A4FB ;And continue processing

*****
*
* This routine is used to get the ending address that *
* is specified after the 'P'. *
*
*****

A4CA: LDA #$04 ;Check if this parameter has been used yet.
A4CC: JSR $A622 ;And if it has, then generate a 'SYNTAX' error
A4CF: JSR $A605 ;Get the end address into the Y register and the
;Accumulator

A4D2: STY $0119 ;Save the LSB and the MSB
A4D5: STA $011A ;Of the ending address into the DOS work area
A4D8: LDA #$04 ;Set the proper bit in location $81
A4DA: BNE $A4C4 ;For this parameter (Bit 2)
A4DC: LDA #$01 ;Set bit 0 to indicate this is the 1st filename
A4DE: JSR $A5B9 ;Then process string-get its address and length
A4E1: STA $0111 ;Save the length of the filename
A4E4: LDY #$00 ;Zero the index pointer
A4E6: JSR $03B7 ;And copy filename from the temporary strings in
A4E9: STA $FF03 ;RAM BANK 1 into SAVRAM in RAM
A4EC: STA $12B7,Y ;BANK 0
A4EF: INY ;Increment the index pointer, see if it has
A4F0: CPY $0111 ;Copied string from temporary string to SAVRAM
;By comparing index pointer to string length
A4F3: BCC $A4E6 ;And if it has not, then continue transferring
;The string
A4F5: LDA #$01 ;Get bit to represent that the 1st filename has
;Been specified
A4F7: ORA $80 ;Set the proper bit for the parameter
A4F9: STA $80 ;Save it
A4FB: JSR $0386 ;Get next character after the previous parameter
A4FE: BNE $A519 ;If have more parameters to process, then branch
A500: JMP $A3EA ;If not, jump back to the Main Loop

```



```
*****
*
* This routine is used to process the parameter that
* is specified after the 'ON'. If 'ON' is not found,
* then the routine branches to check for a 'TO'.
*
*****
```

```
A503: CMP    #$91      ;Is it the token for 'ON'?
A505: BNE    $A50A     ;No, then branch
A507: JMP    $A437     ;Yes, then process the parameter after 'ON'
```

```
*****
*
* This routine checks for the parameter 'TO' and
* generates a 'SYNTAX' error if it is not found.
* This routine then processes the parameters that are
* specified after the 'TO'.
*
*****
```

```
A50A: CMP    #$A4      ;Is it the token for 'TO'?
A50C: BEQ    $A510     ;Yes, then process the parameter after the 'TO'
A50E: BNE    $A57D     ;If not, then generate a 'SYNTAX' error
A510: JSR    $0380     ;Get the next character
A513: CMP    #$50      ;Is it a 'P'?
A515: BNE    $A526     ;If it is not, then branch
A517: BEQ    $A4CA     ;If it is, then process the address
A519: CMP    #$2C      ;Is the character a comma?
A51B: BNE    $A503     ;If it is not, then branch
A51D: JSR    $0380     ;If it is, then get the next character
A520: JMP    $A3F8     ;And continue processing
```

```
*****
*
* This routine processes the values for the filenames
* in quotes or parentheses and the values that are
* specified after the parameters 'D', 'U', and 'ON'.
*
*****
```

```
A523: JSR    $0380     ;Get the next character
A526: CMP    #$44      ;Is it a 'D' for drive number?
A528: BEQ    $A53A     ;Yes, then branch to process it
A52A: CMP    #$91      ;Is it the token for 'ON'?
A52C: BEQ    $A54D     ;Yes, then branch to process it
A52E: CMP    #$55      ;Is it a 'U' for device number?
```

```

A530: BEQ   $A553      ;Yes, then branch to process it
A532: CMP   #$22      ;Is it a quote?
A534: BEQ   $A558      ;Yes, then branch to get the second filename
A536: CMP   #$28      ;Is it an opening parenthesis?
A538: BEQ   $A558      ;Yes, then branch to get the string that is
                        ;Assigned to the variable
A53A: LDA   #$20      ;Check if the parameter has been used yet
A53C: JSR   $A61D      ;And if it has, then generate a 'SYNTAX' error
A53F: JSR   $A5F2      ;Get the drive number into the X register
A542: CPX   #$02      ;If the drive number
A544: BCS   $A57F      ;Is greater than one, then generate an
                        ;'ILLEGAL QUNATITY' error
A546: STX   $0114      ;Save the drive number to the DOS work area
A549: LDA   #$20      ;And get the proper bit in Location $80
A54B: BNE   $A568      ;For this parameter (Bit 5)
A54D: JSR   $A582      ;Process the parameter after 'ON'
A550: JMP   $A568      ;And set the proper bit in Location $80

*****
*
*   This routine is used to get the second device
*   number and use it as the current device number.
*
*****

A553: JSR   $A58D      ;Process the device number
A556: BNE   $A568      ;And set the proper bit in Location $80 for this
                        ;Parameter
A558: LDA   #$02      ;Set bit one to indicate this is the second
A55A: JSR   $A5B9      ;Filename, then process the string and get its
                        ;Address and length
A55D: STA   $0113      ;Save the length of the second filename
A560: STX   $0115      ;And save the address of the second filename
A563: STY   $0116      ;Which is in RAM BANK 1
A566: LDA   #$02      ;Set bit one to indicate the second filename
A568: ORA   $80        ;Has been specified in PARSTX
A56A: STA   $80
A56C: JSR   $0386      ;Get the 1st character after closing quote mark
A56F: BEQ   $A500      ;If not followed by another parameter, branch
A571: CMP   #','       ;If the character after the closing quote is a
A573: BEQ   $A523      ;Comma, then branch to process next parameter
A575: CMP   #$91       ;If not a comma, then is it the token for 'ON'?
A577: BEQ   $A54D      ;If so, then branch to process it
A579: CMP   #'U'       ;Check to see if it is the flag to designate a
A57B: BEQ   $A553      ;New device number and if it is, then
                        ;Branch to process it
A57D: BNE   $A5B6      ;If it is not, then generate a 'SYNTAX' error

```

```
*****
*
*           'ILLEGAL QUANTITY' error generator
*
*****
```

A57F: JMP \$7D28 ;Generate an 'ILLEGAL QUANTITY' error message

```
*****
*
* This routine is used to process the parameters that
* are specified after the token for 'ON'. The only
* two parameters that can be used after the 'ON'
* are 'B' and 'U'.
*
*****
```

```
A582: JSR $0380 ;Get the next character in the command
A585: CMP #$42 ;Is it a 'B' for BANK number?
A587: BEQ $A59E ;Yes, then branch
A589: CMP #$55 ;Is it a 'U' for device number?
A58B: BNE $A5B6 ;If not, then generate a 'SYNTAX' error
A58D: JSR $A5F2 ;Get the device # specified into the X register
A590: CPX #$1F ;If the device number
A592: BCS $A5EA ;Is greater than 30, then generate an
; 'ILLEGAL DEVICE NUMBER' error
A594: CPX #$04 ;If the device number
A596: BCC $A5EA ;Is less than 4, then generate an
; 'ILLEGAL DEVICE NUMBER' error
A598: STX $011C ;Save the device number to the DOS work area
A59B: LDA #$08 ;Value to set bit 3 of Location $80 to
A59D: RTS ;Designate that a device number was specified
```

```
*****
*
* This routine is used to process the BANK number
* that is specified after the 'B' parameter.
*
*****
```

```
A59E: LDA #$01 ;Check if this parameter has been used yet
A5A0: JSR $A622 ;And if it has, then generate a 'SYNTAX' error
A5A3: JSR $A5F2 ;Get the BANK number into the X register
A5A6: CPX #$10 ;If the BANK is greater than 15,
A5A8: BCS $A57F ;Then generate an 'ILLEGAL QUANTITY' error
A5AA: STX $011F ;Save it in the DOS work area
A5AD: LDA #$01 ;And set the proper bit (Bit 1)
```

```

A5AF:  ORA   $81      ;In Location $81
A5B1:  STA   $81      ;For this parameter
A5B3:  LDA   #$00     ;Set the zero flag
A5B5:  RTS                    ;And exit the routine

```

```

*****
*
*          Another 'SYNTAX' error generator
*
*****

```

```

A5B6:  JMP   $796C    ;Generate a 'SYNTAX' error

```

```

*****
*
*          Syntax for Save with Replace
*
* Option syntax:  DSAVE "@Filename"
*
* This routine is used to 'PARSE' the filename
* for the DOS routine.  It will first check the
* accumulator to see which filename it is about to
* parse ( $01 = filename 1 and $02 = filename 2).
* Next it will ensure that the filename has not
* already been parsed.  If it has already been
* parsed, then a 'SYNTAX' error will occur.  Next,
* the routine will check to see if the user has used
* the '@' symbol which will indicate to the disk drive
* a save with replace function is being requested.
* If a save with replace has been specified, bit 7
* of location $80 will be set and the starting address
* of the string will be incremented by one to remove
* the '@' symbol from the filename.  This is to enable
* the individual routines to see if a save with
* replace is allowed.  If it is allowed, then the
* individual routine will prefix the filename with
* '@(drive number):'.  If it is not allowed,
* (say with the DLOAD command) a 'SYNTAX' error will
* be generated.  Note: If you enter
* DSAVE "@0:Filename 1", the C-128 operating system
* will send the following filename to the disk drive:
* "0:0:filename 1".  This bug is because a check is
* not performed on the "0:" the user inputted.
* Therefore, ensure you use the new command syntaxes
* correctly.  When this routine exits, the

```

```

* X and Y registers, and locations $24, $25 will      *
* contain the address of the string in RAM BANK 1 and *
* the accumulator will contain the length of the     *
* string.                                             *
*                                                     *
* NOTE: The use of the save with replace option is   *
* not recommended due to software problems with     *
* the Commodore disk drives. Instead of using      *
* the save with replace option, first scratch      *
* the old version of the file and then save        *
* the new version to disk.                          *
*                                                     *
*****

```

```

A5B9: JSR  $A61D      ;See if the filename in the accumulator has been
                    ;Specified. If so, generate a 'SYNTAX' error
A5BC: JSR  $877B      ;If filename has not been specified yet, then
                    ;Place address of the string in RAM BANK 1 and
                    ;In locations $24, $25, and the string's length
                    ;In the accumulator
A5BF: TAX             ;Then save the length of string in the
                    ;Accumulator to the X register
A5C0: BEQ  $A5E7      ;If the filename length is zero, then generate a
                    ;'FILENAME MISSING' error
A5C2: LDY  #$00       ;Zero the index register
A5C4: JSR  $03B7      ;And get the first character of the filename
A5C7: CMP  #'@'       ;To check if a save with replace is being
A5C9: BNE  $A5DD      ;Attempted and if not, then branch
A5CB: LDA  #$80       ;Check bit 7 in $80 (PARSTX) to see if a
                    ;Save with replace
A5CD: JSR  $A61D      ;Has already been attempted and if so, then
                    ;Generate a 'SYNTAX' error
A5D0: LDA  $80        ;Since save with replace has not been attempted
                    ;Before
A5D2: ORA  #$80       ;Set bit 7 to indicate that it is being
A5D4: STA  $80        ;Attempted now
A5D6: DEX             ;Subtract 1 from length of the string, then add
A5D7: INC  $24        ;One to the address of the string in RAM BANK 1
A5D9: BNE  $A5DD      ;This is done to 'Remove' user's '@' symbol from
                    ;The filename
A5DB: INC  $25        ;So the system can prefix '@ (drive number):' to
                    ;The filename
A5DD: TXA             ;Move the length of filename into accumulator
A5DE: CMP  #171       ;And ensure the length of the filename is not
                    ;Over 16 characters
A5E0: BCS  $A5ED      ;If it is, then generate 'STRING TOO LONG' error
A5E2: LDX  $24        ;Exit routine with the length of the filename

```

```
A5E4: LDY   $25           ;In accumulator and address of the string which
A5E6: RTS                   ;Is in RAM BANK 1 in the X and Y registers
```

```
*****
*
*                               Error Generator
*
*   Enter at:  A5E7 for 'MISSING FILENAME' error
*               A5EA for 'ILLEGAL DRIVE NUMBER' error
*               A5ED for 'STRING TOO LONG' error
*
*****
```

```
A5E7: LDX   #8           ;Error number for 'MISSING FILENAME' error
A5E9: .BYTE $2C         ;Mask to skip the next two instructions
A5EA: LDX   #9           ;Error number for 'ILLEGAL DRIVE NUMBER' error
A5EC: .BYTE $2C         ;Mask to skip the next instruction
A5ED: LDX   #23          ;Error number for 'STRING TOO LONG' error
A5EF: JMP   $4D3C        ;Generate the error message
```

```
*****
*
*   This routine is used to get the eight bit numeric
*   value that is specified after the 'B', 'D', 'U',
*   '#', and 'L' parameters.
*
*****
```

```
A5F2: JSR   $0380        ;Get the first character after the parameter
A5F5: BEQ   $A5B6        ;If it is a zero or colon, then generate a
                        ;'SYNTAX' error
A5F7: BCC   $A602        ;If it is not a variable, then branch
A5F9: JSR   $7959        ;Check for an opening parenthesis
A5FC: JSR   $87F4        ;Get value of the variable into the X register
A5FF: JMP   $7956        ;Check for a closing parenthesis and exit
```

```
*****
*
*   This routine is used to convert the eight bit ASCII
*   value that is specified after the 'B', 'D', 'U',
*   and 'L' parameters to hex and then store the value
*   into the X register.
*
*****
```

```
A602: JMP   $87F4        ;Get the numeric value into the X register
```

```

*****
*
* This routine is used to get the sixteen bit numeric
* address that is specified after the 'D' parameter
* into the Y register and the accumulator (LSB,MSB).
*
*****

A605: JSR   $0380      ;Get the first character after the parameter
A608: BEQ   $A5B6      ;If it is a zero or a colon, then generate
                        ;A 'SYNTAX' error
A60A: BCC   $A61A      ;If it is not a variable, then branch
A60C: JSR   $7959      ;Check for an opening parenthesis
A60F: JSR   $8812      ;Put value of variable into locations $16,$17
A612: JSR   $7956      ;Check for a closing parenthesis
A615: LDY   $16        ;Get the LSB
A617: LDA   $17        ;And the MSB of the address
A619: RTS                      ;And exit

*****
*
* This routine is used to get the sixteen bit ASCII
* address that is specified after the 'P' parameter
* into the Y register and the accumulator (LSB,MSB).
*
*****

A61A: JMP   $8812      ;Get the address into the Y register and the
                        ;Accumulator

*****
*
* This routine is used to check if a parameter has
* been used yet. Enter with the value of the bit you
* wish to check in the accumulator. If the parameter
* has already been used, a 'SYNTAX' error will be
* generated.
*
*****

A61D: AND   $80        ;Mask specified bit
A61F: BNE   $A5B6      ;If the parameter has been used, then generate
                        ;A 'SYNTAX' error
A621: RTS                      ;Else, exit the routine

```

```

*****
*
* This routine operates the same as the routine at
* $A61D except location $81 is checked instead.
*
*****

```

```

A622: AND   $81           ;Mask the proper bit
A624: BNE   $A5B6        ;If the parameter has been used, then generate
                        ;A 'SYNTAX' error
A626: RTS                   ;Else, exit the routine

```

```

*****
*
* This is the command table of the characters and
* control values to generate the appropriate disk
* command for each of the BASIC DOS commands.
*
* The following is a table of the control values in
* the command table and their meanings.
*
* CONTROL VALUE           MEANING
* -----
* D0 -- Place the two character ID in the command
*      buffer
* D1 -- Place the first drive number in the command
*      buffer
* D2 -- Place the second drive number in the command
*      buffer
* E0 -- Place the read/write character in the command
*      buffer
* E1 -- Place the file type (P,S,L) in the command
*      buffer
* E2 -- Place the record number in the command
*      buffer
* F0 -- Place the save and replace character '@'
*      in the command buffer
* F1 -- Place the first filename in the command
*      buffer
* F2 -- Place the second filename in the command
*      buffer
* C2 -- Place the channel number in the command
*      buffer
*
*****

```



```

*****
*
*                               DCLEAR
*
*   I (Drive number)
*
*****

```

A627: .BYTE \$49,\$D1

```

*****
*
*                               DIRECTORY
*
*   $ (Drive number):(wildcard)
*
*****

```

A629: .BYTE \$24,\$D1

A62B: .BYTE \$3A,\$F1

```

*****
*
*   DOPEN, DSAVE, DLOAD, DVERIFY, BSAVE, BLOAD
*
*   '@' (Drive number): (Filename), (Filetype), (L or W)
*
*****

```

A62D: .BYTE \$F0,\$D1

A62F: .BYTE \$3A,\$F1

A631: .BYTE \$2C,\$E1

A633: .BYTE \$2C,\$E0

```

*****
*
*                               CONCAT
*
*   C (Drive number): (First Filename) = (Drive
*   number):(First Filename), (Drive number):
*   (Second Filename)
*
*****

```

A635: .BYTE \$43,\$D2

A637: .BYTE \$3A,\$F2

A639: .BYTE \$3D,\$D2

A63B: .BYTE \$3A,\$F2

A63D: .BYTE \$2C,\$D1

```
*****
*
*                               APPEND
*
*      (Drive number): (Filename), A
*
*****
```

A63E: .BYTE \$D1

A63F: .BYTE \$3A,\$F1

A641: .BYTE \$2C,\$41

```
*****
*
*                               HEADER
*
*      N (Drive number):(Disk name), (ID)
*
*****
```

A643: .BYTE \$4E,\$D1

A645: .BYTE \$3A,\$F1

A647: .BYTE \$2C,\$D0

```
*****
*
*                               COLLECT
*
*      V(Drive number)
*
*****
```

A649: .BYTE \$56,\$D1

```
*****
*
*                               BACKUP
*
*      D(Second Drive number) = (First Drive number)
*
*****
```

A64B: .BYTE \$44,\$D2

A64D: .BYTE \$3D,\$D1



A663: .BYTE \$50,\$C2

A665: .BYTE \$E2,\$E0

```
*****
*
* This routine is used to place the proper sequence
* of characters in the buffer at $1100 for the DOS
* commands.
*
* Ex: DCLEAR
*
* Buffer = IO
*
* Enter with the accumulator holding the number of
* characters in the command, and the Y register
* holding the offset minus one to the proper command
* in the command table.
*
*****
```

```
A667: STA    $0110    ;Save the number of characters in the command
A66A: TYA
A66B: PHA
A66C: JSR    $A80D    ;Deallocate DS$
A66F: LDX    #$00
A671: PLA
A672: DEC    $0110    ;Get back the index to the command table
A675: BMI    $A6BF    ;If we have finished getting
A677: TAY
A678: INY
A679: TYA
A67A: PHA
A67B: LDA    $A627,Y  ;This command, then branch
A67E: BPL    $A6B7    ;Increment the command table index
                        ;By one and save
                        ;It back onto
                        ;The stack
A67B: LDA    $A627,Y  ;Get a character of the command
A67E: BPL    $A6B7    ;If it is not a control character, then branch
                        ;To place it in the command buffer
```

```
*****
*
* This routine is used to check the control value
* from the command and store the proper character
* or sequence of characters in the command buffer.
*
*****
```

```
A680: CMP    #$C2
A682: BEQ    $A6D6    ;Is it the control value to get the channel #?
                        ;Yes, then branch to store the channel # into
                        ;Buffer
```

---

```

A684:  CMP    #$D0      ;Is it the control value to get the second
                        ;character ID?
A686:  BEQ    $A6E5     ;Yes, then branch to put the ID in the buffer
A688:  CMP    #$E2      ;Is it the control value to get the record #?
A68A:  BEQ    $A703     ;Yes, then branch to store the record number
                        ;In the buffer
A68C:  CMP    #$E1      ;Is it the control value to get the file type?
A68E:  BEQ    $A6F1     ;Yes, check to see if an 'L' was used and if not
                        ;Then designate the type as sequential
A690:  CMP    #$F0      ;Is it the control value to get the '@' flag?
A692:  BEQ    $A6DB     ;Yes, then place the '@' in the buffer
A694:  CMP    #$F1      ;Is it the control value to get 1st filename?
A696:  BEQ    $A70D     ;Yes, then branch to put the 1st filename into
                        ;The buffer
A698:  CMP    #$F2      ;Is it the control value to get 2nd filename?
A69A:  BEQ    $A6BD     ;Yes, then branch to put the 2nd filename
                        ;Into the buffer
A69C:  CMP    #$E0      ;Is it control value to get the file operation?
A69E:  BNE    $A6A5     ;No, then branch
A6A0:  LDA    $011E     ;Get type of file operation 'W' or default to
                        ;Read
A6A3:  BNE    $A6B7     ;And store it in the buffer
A6A5:  CMP    #$D1      ;Is it the control value to get the drive #?
A6A7:  BNE    $A6AE     ;No, then branch
A6A9:  LDA    $0112     ;Get the current drive number
A6AC:  BPL    $A6B5     ;Branch to store it in the command buffer
A6AE:  CMP    #$D2      ;Is it the control value to get the second drive
                        ;Number?
A6B0:  BNE    $A671     ;No, then back to Loop
A6B2:  LDA    $0114     ;Get the drive number for the second filename
A6B5:  ORA    #$30       ;Create an ASCII digit
A6B7:  STA    $1100,X   ;And save it to the command buffer
A6BA:  INX                ;Move index to the next byte in command buffer
A6BB:  BNE    $A671     ;If index is not 0, then branch to get the next
                        ;Command
A6BD:  BEQ    $A723     ;If index is 0, then branch to get the second
                        ;Filename
A6BF:  TXA                ;Save the index to the command buffer
A6C0:  PHA                ;Onto the stack
A6C1:  LDX    #$00      ;Set up the LSB
A6C3:  LDY    #$11      ;And the MSB of the command or name
A6C5:  JSR    $925D     ;Call BASIC SETNAM to set up the parameters for
                        ;Command or name
A6C8:  LDA    $011B     ;Get the current logical file number
A6CB:  LDX    $011C     ;Get the current device number
A6CE:  LDY    $011D     ;Get the current secondary address
A6D1:  JSR    $9257     ;Call BASIC SETLFS to set up the file parameters

```

```
A6D4: PLA          ;Get the command buffer index back
A6D5: RTS          ;Exit the routine
```

```
*****
*
* This routine gets the channel number for the RECORD *
* command and branches to store it in the command *
* buffer.                                           *
*
*****
```

```
A6D6: LDA  $11ED      ;Get the channel number for the RECORD command
A6D9: BNE  $A6B7      ;And branch back to Loop
```

```
*****
*
* This routine checks if the save with replace flag *
* '@' was used in the filename and if so, the routine *
* then stores an '@' in the command buffer.         *
*
*****
```

```
A6DB: BIT  $80        ;Check if the save and replace flag '@' was used
A6DD: BMI  $A6E1      ;And if so, then branch
A6DF: BPL  $A671      ;If not, then continue with Loop
A6E1: LDA  #$40        ;Get the save and replace flag '@'
A6E3: BNE  $A6B7      ;And branch to include it in the command
```

```
*****
*
* This routine stores the two character ID for the *
* HEADER command into the command buffer.         *
*
*****
```

```
A6E5: LDA  $0120      ;Get the first character of the ID
A6E8: STA  $1100,X    ;Store it in the command buffer
A6EB: INX          ;Move to the next byte in the buffer
A6EC: LDA  $0121      ;Get the second character of the ID
A6EF: BNE  $A6B7      ;And branch to store it in the command buffer
```

```
*****
*
* This routine checks location $011E to see if a file *
* length was specified and if it was, then the *
* routine stores an 'L' in the command buffer. If a *
* a record length was not specified, then the file is *
* designated as a sequential write file. *
*
*****
```

```
A6F1: LDA $011E ;Check if a record length was specified
A6F4: BEQ $A6FA ;If not, then designate the file as sequential
A6F6: LDA #$4C ;If a record length was specified then place
A6F8: BNE $A6B7 ;An 'L' in the command buffer and continue
;With the Loop
A6FA: LDA #$53 ;Store an 'S' in the DOS work area to
A6FC: STA $011E ;Designate the file as a sequential file
A6FF: LDA #$57 ;Get a 'W' to designate a write file
A701: BNE $A6B7 ;And branch to store it in the buffer
```

```
*****
*
* This routine stores the sixteen bit record number *
* in LSB,MSB format to the command buffer. *
*
*****
```

```
A703: LDA $16 ;Get the LSB of the record number
A705: STA $1100,X ;Save it in the command buffer
A708: LDA $17 ;Get the MSB of the record number
A70A: INX ;Move to the next byte of the command buffer
A70B: BNE $A6B7 ;And branch to store MSB in the command buffer
```

```
*****
*
* This routine is used to transfer the first filename *
* from $12B7 to the command buffer at $1100. *
*
*****
```

```
A70D: LDY $0111 ;Get the length of the first filename
A710: BEQ $A745 ;If it is zero, then branch
A712: LDY #$00 ;Zero an index
A714: LDA $12B7,Y ;Transfer the
A717: STA $1100,X ;First filename
A71A: INX ;To the
A71B: INY ;Command buffer
```

```

A71C: CPY   $0111      ;Continue until the entire filename
A71F: BNE   $A714      ;Has been transferred
A721: BEQ   $A746      ;Branch to continue with the Loop

```

```

*****
*
* This routine is used to transfer the second filename *
* from RAM BANK 1 to the command buffer at $1100.    *
*
*****

```

```

A723: LDA   $0115      ;Save the BANK 1
A726: STA   $24        ;Address of the second filename
A728: LDA   $0116      ;Into $24, $25
A72B: STA   $25
A72D: LDY   $0113      ;Check the length of the second filename
A730: BEQ   $A745      ;And if it is zero, then branch
A732: LDY   #$00       ;Zero an index
A734: JSR   $03B7      ;Get a character of the second filename
A737: STA   $FF03      ;Enable BANK 0
A73A: STA   $1100,X    ;And store the character in the command buffer
A73D: INX
A73E: INY
A73F: CPY   $0113      ;Until done
A742: BNE   $A734
A744: .BYTE $24        ;Mask to skip the next instruction
A745: DEX
A746: JMP   $A671      ;Continue with the Loop

```

```

*****
*
* This routine is used to ensure that no other      *
* parameters but the filename, the device number, or *
* the drive number are used. If others are used, then *
* a 'SYNTAX' error is generated. The routine must be *
* entered with the value of location $80 in the      *
* accumulator.
*
*****

```

```

A749: AND   #$E6       ;Mask to test for illegal parameters
A74B: BEQ   $A750      ;If no illegal parameters were specified, branch
A74D: JMP   $796C      ;Generate a 'SYNTAX' error

```



```
*****
*
*                          CHKNAM
*
*           Check for a fileName after the command
*
* This routine checks if there is a filename after
* the command that was parsed and if a filename is
* not present, then the routine generates a 'SYNTAX'
* error.
*
*****
```

```
A750: LDA   $80           ;Get type of parameters following the command
A752: AND   #$01          ;Mask out the bit for the filename
A754: CMP   #$01          ;If a filename was not specified after command,
A756: BNE   $A74D         ;Then branch to generate a 'SYNTAX' error
A758: LDA   $80           ;Get the types of parameters in the accumulator
A75A: RTS                  ;And exit
```

```
*****
*
* This routine ensures that the only parameters that
* are specified after the command are the device
* number or the drive number. The routine must be
* entered with the value of location $80 in the
* accumulator. If any other parameters are specified,
* a 'SYNTAX' error is generated.
*
*****
```

```
A75B: AND   #$E7           ;Ensure that only the drive # or device # is
                               ;Specified
A75D: BNE   $A74D         ;If other parameters are specified, generate
                               ;A 'SYNTAX' error
A75F: RTS                  ;If no other parameters specified, then exit
```

```
*****
*
* This routine is used to ensure that two filenames
* are present after the command being processed.
* If they are not, then a 'SYNTAX' error is generated.
* This routine also ensures that a save and replace
* '@', logical file number, or record length is not
* present.
*
*****
```

```

A760:  AND  #$C4      ;Ensure that the parameters, save & replace '@',
                    ;Record length, or logical file number are not
762:   BNE  $A74D     ;Specified
A762:  BNE  $A74D     ;If any of these are specified, then generate
                    ;A 'SYNTAX' error
A764:  LDA  $80      ;Get the bit representation of the parameters
                    ;After the command
A766:  AND  #%0011   ;Mask bits for the first and second filename
A768:  CMP  #$03     ;If the two filenames are not present,
A76A:  BNE  $A74D     ;Then branch to generate a 'SYNTAX' error
A76C:  LDA  $80      ;Get the types of parameters
A76E:  RTS          ;And exit

```

```

*****
*
*   This routine checks if a logical file number and
*   a filename were specified after the command.  If
*   they are not specified after the command, then a
*   'SYNTAX' error is generated.  This routine is
*   entered with the value from location $80 in the
*   accumulator.
*
*****

```

```

A76F:  AND  #%00000101 ;Mask the bits for filename, and logical file #
A771:  CMP  #$05      ;If a filename and a logical file #
A773:  BNE  $A74D     ;Are not present, then generate a 'SYNTAX' error
A775:  LDA  $80      ;Get the type of parameters
A777:  RTS          ;And exit

```

```

*****
*
*                               HANDLE DS$
*
* This routine will read the current disk drive's
* error channel (message) and will either assign
* it to a new DS$ or to the old DS$. If location $7A
* is not equal to zero, then this routine will assign
* the error message to the old DS$. However, if
* location $7A contains a zero, this routine will
* create space for a new DS$ and read in and assign
* it to DS$.
*
* Note: To use this routine in machine language,
* place the address of where you wish the error
* message to be located in RAM BANK 1 in
* locations $7B, $7C and JSR $A795. This
* routine will read in the error message and
* place a delimiter of zero after after the
* error message.
*
* Note: This routine does not actually close the
* command channel (secondary address of $6F
* or $60 + CMD address of $OF) down in the
* disk drive but, it uses the method of
* setting the carry flag and JSR CLOSE which
* will close the I/O channel internally in
* the C-128. This is done this way to enable
* you to leave any channels that are currently
* open to remain opened. This is so because
* when you close the command channel to the
* disk drive, it will automatically close all
* of its internal channels which would cause
* various errors to occur. But, the errors
* will not be caught by DS$ because, as far as
* the disk drive is concerned, everything is
* okay. However, if you check ST, it will
* either equal $40 or $42 which indicates the
* drive produced an 'end of file' or an
* 'End of file' and a 'Time out read' error.
*
*****

```

```

A778: LDA   $7A           ;If DS$ has already been allocated,
A77A: BNE   $A795        ;Then branch past obtaining a new DS$
A77C: LDA   #$28         ;Place the length to be allocated for
A77E: STA   $7A         ;DS$ in $7A, then allocate the space for

```

```

A780: JSR    $9299    ;DS$ + 2 bytes for the descriptor's address
A783: STX    $7B      ;Save the address of the space created
A785: STY    $7C      ;For DS$ in $7B, $7C
A787: LDY    #$28     ;Get the length of DS$
A789: STA    $FF04    ;Enable Bank 15 with RAM BANK 1 enabled
A78C: LDA    #$7A     ;Save the address of
A78E: STA    ($7B),Y  ;DS$ variable descriptor
A790: INY                    ;After the string
A791: LDA    #$00     ;Which is always
A793: STA    ($7B),Y  ;Located at $007A
A795: LDX    $011C    ;Get the device # (unit number) to get DS$ from
A798: BNE    $A79F    ;And if not zero, then branch to process it
A79A: LDX    #$08     ;If device address was zero, make it the default
A79C: STX    $011C    ;Address of eight
A79F: LDA    #$00     ;Then call SETCFS to set the logical file
A7A1: LDY    #$6F     ;Number to 0 and the device address that's in
                    ;The accumulator
A7A3: JSR    $9257    ;And the secondary address of 15
A7A6: LDA    #$00     ;Then set the length of the filename
A7A8: JSR    $925D    ;To zero
A7AB: JSR    $90D8    ;Then call OPEN to finish opening zero,
                    ;Accumulator, 15
A7AE: LDX    #$00     ;Open an input channel to the logical
A7B0: JSR    $FFC6    ;File number of zero and if an error occurs
A7B3: BCS    $A7D5    ;During the opening process, then branch with
                    ;The error number in the accumulator
A7B5: LDY    #$FF     ;Get this value so the next instruction sets the
                    ;Index value to zero
A7B7: INY                    ;Increment the index register
A7B8: JSR    $9263    ;Get a character from the disk drive
A7BB: STA    $FF04    ;Then Enable Bank 15 with RAM BANK 1 Enabled
A7BE: CMP    #$0D     ;And see if it is a carriage return denoting the
A7C0: BEQ    $A7C8    ;End of the error message. If it is a carriage
                    ;Return, then branch
A7C2: STA    ($7B),Y  ;If not a carriage return, then place the error
                    ;Message in DS$'s address
A7C4: CPY    #$28     ;Continue reading in DS$ until a carriage return
A7C6: BCC    $A7B7    ;Or length of the string reaches 40 characters
A7C8: LDA    #$00     ;Place a delimiter of zero
A7CA: STA    ($7B),Y  ;After DS$ then
A7CC: JSR    $926F    ;Close the input file down
A7CF: LDA    #$00     ;Close the input channel down in the
A7D1: SEC                    ;C-128, but NOT in the disk drive to
A7D2: JMP    $9275    ;Prevent the drive from closing other channels

```

```

*****
*
* This routine will soft close the input channel and
* place a delimiter after DS$. The routine will then
* generate the error message.
*
*****

```

```

A7D5: PHA                ;Save the error number onto the stack
A7D6: JSR   $A7C8        ;Soft-close input channel,place delimiter
                          ;After DS$
A7D9: JSR   $A80D        ;Deallocate DS$
A7DC: PLA                ;Get the error number back off the stack
A7DD: TAX                ;Place it in the X register
A7DE: JMP   $4D3C        ;Then jump to the error message handler

```

```

*****
*
* This routine checks to see if we are in the PROGRAM
* mode and if we are, then the question 'ARE YOU SURE'
* is not asked. Instead, the command is executed.
*
*****

```

```

A7E1: BIT   $7F          ;Check if we are in the PROGRAM mode
A7E3: BMI   $A80A        ;If we are, then branch
A7E5: JSR   $9281        ;PRIMM, output the following text
A7E8: TXT   'are you sure?'
A7F5: .BYTE $00          ;Delimiter byte
A7F6: JSR   $926F        ;Reset system to the default I/O configuration
A7F9: JSR   $9263        ;Input the first drive character
A7FC: PHA                ;And save it onto the stack
A7FD: CMP   #13          ;Is it a carriage return?
A7FF: BEQ   $A806        ;If it is, then exit
A801: JSR   $9263        ;If it is not, then keep inputting characters
A804: BNE   $A7FD        ;Until a carriage return is processed
A806: PLA                ;Get the first character back
A807: CMP   #'y'         ;Compare it to a 'Y' to condition the Z flag
A809: RTS                ;And exit

```

```
*****
*
*   If we are in the PROGRAM mode, then set the flag   *
*   for the BEQ branch.  This routine is used if the   *
*   SCRATCH, HEADER, or BACKUP commands are used in the *
*   PROGRAM mode.  So the comparison to 'Y' in the     *
*   'ARE YOU SURE' question is always true.           *
*
*****
```

```
A80A: LDA    #0           ;Set the zero flag
A80C: RTS                      ;And exit
A80D: TYA                      ;Preserve the index register onto the
A80E: PHA                      ;Stack
A80F: LDA    $7A          ;If DS$ has not been updated
A811: BEQ    $A820        ;Since the last disk operation, then branch
                                ;To prevent the deallocation of the old DS$
A813: LDY    #$28         ;Place the # of characters to deallocate in the
                                ;Accumulator and the Y register

A815: TYA
A816: STA    $FF04        ;Enable Bank 15 with RAM BANK 1 Enabled
A819: STA    ($7B),Y      ;Place the # of characters to deallocate in the
A81B: INY                      ;Two bytes following string (Descriptor address)
A81C: LDA    #$FF        ;So that the next string operation or garbage
A81E: STA    ($7B),Y      ;Collection will deallocate and use this address
A820: LDA    #$00        ;Flag to indicate a disk operation occurred
                                ;Since the last time DS$ was processed
A822: STA    $FF03        ;Enable Bank 14
A825: STA    $7A         ;Set flag to get a new DS$ from the disk drive
A827: PLA                      ;Get the Y register off of the stack
A828: TAY                      ;And restore it in the Y register
A829: RTS                      ;Exit the routine
```

```
*****
*
*   This routine is used by the 'KEY' command to output *
*   the word 'KEY' plus the key number.                 *
*
*****
```

```
A82A: .BYTE $00           ;Delimiter byte
A82B: TXT ' YEK'         ;Text for the KEY command
```

```

*****
*
* This routine is used to output the ASCII value in
* accumulator. The Y register is preserved.
*
*****

A830: TAX                ;Save the value to be outputted
A831: TYA                ;Save the Y register
A832: PHA                ;Onto the stack
A833: LDA    #$00        ;Set the MSB of the value to zero
A835: JSR    $8E32       ;Output the value in X,A (LSB,MSB) in ASCII
A838: PLA                ;Restore the Y register
A839: TAY
A83A: RTS                ;And exit

*****
*
* This routine is used by the GOSUB and DO commands to
* check if the line number specified on the pseudo stack
* is too large. If the line number is too large, this
* routine sets $7F to zero, which puts the system into
* DIRECT mode and exits. This means that if somehow you
* are able to slip a line number past the operating
* system that is too large, this routine will stop your
* BASIC program without displaying an error!
*
*****

A83B: STA    $3C         ;Save MSB of the line number
A83D: DEY                ;Decrement the index pointer by one to point
                        ;To the LSB of the line number
A83E: TAX                ;Move the MSB into the X register
A83F: INX                ;Add one to it
A840: BNE    $A844       ;If it was not $FF then exit
A842: STX    $7F         ;Place the zero into mode to stop the program
                        ;After this BASIC statement is executed
A844: RTS                ;Exit this routine

*****
*
*
* BASIC BANK 15
*
* This routine switches the computer to BANK 15.
* The value in the accumulator is preserved.
*
*****

```

```

A845: PHA                ;Save the accumulator onto the stack
A846: LDA    #0          ;Set the memory configuration
A848: STA    $FF00       ;To BASIC BANK 15
A84B: PLA                ;Restore the accumulator
A84C: RTS                ;And exit

*****
*
*          BASIC IRQ ROUTINE
*
*****

A84D: LDA    $12FD       ;If the BASIC IRQ
A850: BEQ    $A853       ;Is already in progress,
A852: RTS                ;Then exit the routine
A853: INC    $12FD       ;Block the BASIC IRQ
A856: LDX    #$10        ;Copy each of the eight sprites X and Y location
A858: LDA    $11D6,X     ;(Position) into their respective VIC Register
A85B: STA    $D000,X     ;(8 sprites X 2 registers=16 bytes to be copied)
A85E: DEX
A85F: BPL    $A858
A861: LDY    #$07        ;Check each one of the eight sprites
A863: LDA    $D015       ;To see if they are Enabled (=1) and if
A866: AND    $6CB3,Y     ;They are not, then
A869: BEQ    $A8A3       ;Branch to check the next sprite
A86B: LDX    $6DD9,Y     ;If the sprite # that is in the Y register is
                        ;Enabled, then get the offset to its speed value
A86E: LDA    $117E,X     ;Get the speed value of the sprite
A871: BEQ    $A8A3       ;If the speed is 0, then continue to next sprite
A873: STA    $117F,X     ;Save the speed value
A876: TYA                ;Transfer the sprite number to the accumulator
A877: ASL                ;Multiply it by two
A878: TAY                ;Save the result in the Y register
A879: LDA    $1180,X     ;Get the angle of movement divided by 90
A87C: SEC
A87D: SBC    #$01        ;Subtract 1 from it
A87F: INX                ;Move the X index to the proper
A880: INX                ;Timer value
A881: INY                ;And the Y index to the proper sprite coordinate
A882: JSR    $A9F4       ;Update the Y coordinate and timers
A885: DEX                ;Restore the indexes
A886: DEX                ;To point to the X
A887: DEY                ;Coordinate of the sprite and its corresponding
                        ;Timer
A888: LDA    $1180,X     ;Get the angle of movement divided by 90
A88B: JSR    $A9F4       ;And update the X coordinate and timers
A88E: PHP                ;Save the status register onto the stack

```



```

A88F: TYA                ;Move the sprite number to the accumulator
A890: LSR                ;Divide it by 2
A891: TAY                ;And move it back to the accumulator
A892: PLP                ;Get the status register back
A893: BCC $A89E          ;If sprite did not cross the seam, then branch
A895: LDA $11E6          ;If the sprite crossed the seam,
A898: EOR $6CB3,Y       ;Set the MSB bit of the sprite
A89B: STA $11E6          ;To indicate that the sprite is on the seam
A89E: DEC $117F,X       ;Decrement the speed factor by 1
A8A1: BNE $A876         ;Update coordinates until speed value equals 0
A8A3: DEY                ;Move to the next sprite register
A8A4: BPL $A863         ;Branch until all eight are done
A8A6: LDA $D019         ;Get the status of the VIC IRQ
A8A9: STA $D019         ;Erase the register
A8AC: AND #$0E          ;Mask bits for light pen and sprite collision
A8AE: BEQ $A8F4         ;If no IRQ triggered, then branch
A8B0: LSR                ;Shift all bits right one bit
A8B1: LDY #$01          ;Set index to the sprite to Background
                        ;Collision Register
A8B3: LSR                ;Check for sprite to Background Collision
A8B4: BCC $A8D6         ;If none, then branch
A8B6: PHA                ;Save the IRQ condition to the stack
A8B7: LDA $D01E,Y       ;Get the sprite to sprite collision register
A8BA: ORA $11E7,Y       ;Set bits for sprites involved in the last
A8BD: STA $11E7,Y       ;Collision and save
A8C0: LDA #$00          ;Clear the appropriate
A8C2: STA $D01E,Y       ;Sprite collision register
A8C5: LDA $127F         ;Get value of what type of interrupt check is
                        ;Requested
A8C8: CPY #$00          ;Was last register we read the sprite to sprite
A8CA: BEQ $A8CD         ;Collision register? If so then shift only 1 bit
A8CC: LSR                ;If entered here, shift the bit for sprite to
                        ;Background collision into the carry
A8CD: LSR                ;If entered here, shift bit for sprite to sprite
                        ;Collision into the carry
A8CE: BCC $A8D5         ;If there was no collision, then branch to
                        ;Continue to the next register
A8D0: LDA #$FF          ;Store a $FF in the correct location to indicate
                        ;$1277 = Sprite to Background, $1276 = Sprite to
                        ;Sprite Collision
A8D2: STA $1276,Y       ;What type of interrupt took place
A8D5: PLA                ;Get the VIC IRQ condition off of the stack
A8D6: DEY                ;Move to the next sprite collision register
A8D7: BPL $A8B3         ;Loop until both collision registers are done
A8D9: LSR                ;Check for a light pen interrupt
A8DA: BCC $A8F4         ;If there is none, then branch
A8DC: LDA $D013         ;Save the light pen

```

```

A8DF: STA $11E9 ;X coordinate
A8E2: LDA $D014 ;And the
A8E5: STA $11EA ;Y coordinate
A8E8: LDA $127F ;Get the interrupt value
A8EB: AND #$04 ;And see if light pen interrupt was requested
A8ED: BEQ $A8F4 ;If not, then branch
A8EF: LDA #$FF ;Store a $FF in LPWT to
A8F1: STA $1278 ;Indicate a light pen interrupt has occurred
A8F4: LDX #$00 ;Clear an index
A8F6: LDA $1224,X ;Check if the voice is active ($FF = inactive)
A8F9: BMI $A922 ;If the voice is inactive, then branch to check
;The next voice
A8FB: LDA $1223,X ;Get the timer value for the active voice
A8FE: SEC ;Subtract the current
A8FF: SBC $1222 ;Tempo rate from the timer value
A902: STA $1223,X ;And save it back
A905: BCS $A922 ;If timer has not counted down yet, then branch
A907: LDA $1224,X ;If the timer has counted down,
A90A: SBC #$00 ;Then subtract one from the voice flag
A90C: STA $1224,X ;And save it back
A90F: BCS $A922 ;If the flag was reset to inactive ($FF), do the
;Next voice
A911: TXA ;Divide the index by 2
A912: LSR ;To get the voice number
A913: TAY ;And transfer it to the Y register as an index
A914: LDA $1230,Y ;Get the waveform for the selected voice
A917: AND #$FE ;Drop the gate bit
A919: PHA ;And save the waveform onto the stack
A91A: LDA $7039,Y ;Get the offset to the control register of the
;Selected voice
A91D: TAY ;Use the offset as an index
A91E: PLA ;Get the waveform off the stack
A91F: STA $D404,Y ;And store it in the control register
A922: INX ;Increment the index by 2
A923: INX ;To get the next voice register
A924: CPX #$06 ;Have we done all 3 yet?
A926: BNE $A8F6 ;If not, then branch
A928: LDY #$02 ;Set an index to the sound time MSB for voice 3
A92A: LDA $1285,Y ;Get the sound time MSB
A92D: BPL $A935 ;Branch if in use
A92F: DEY ;If not in use, then move to the next register
A930: BPL $A92A ;And continue
A932: JMP $A9F0 ;Exit IRQ
A935: CLC ;Clear the carry for addition
A936: LDA $129D,Y ;Add the LSB of the Frequency Sweep Step
A939: ADC $1297,Y ;To the sound frequency LSB
A93C: STA $129D,Y ;And save as the new frequency LSB

```

```

A93F: LDA $12A0,Y ;Add the MSB of the Frequency Sweep Step
A942: ADC $129A,Y ;To the sound frequency MSB
A945: STA $12A0,Y ;And save as the new frequency MSB
A948: LDA $1294,Y ;Get the frequency sweep direction
A94B: TAX ;Save it in the X register
A94C: AND #$01 ;Check if the sweep direction is down
A94E: BEQ $A97E ;If not, then branch
A950: BCC $A961 ;If it is not down, then check if the sweep
;Direction is up
A952: SEC ;Set the carry for subtraction
A953: LDA $129D,Y ;Check if the current
A956: SBC $128E,Y ;Sound frequency is
A959: LDA $12A0,Y ;Less than or equal to the
A95C: SBC $1291,Y ;Specified minimum frequency
A95F: BCS $A9AE ;If current frequency less than minimum, branch
A961: CPX #$02 ;Is the direction of the frequency up?
A963: BCC $A96F ;If less, then branch
A965: JSR $A9DA ;Invert the sweep step value
A968: LDA #$02 ;Set the sweep direction
A96A: STA $1294,Y ;To 2 for oscillating
A96D: BNE $A9A2 ;And set up frequency and update the counters
A96F: LDA $1288,Y ;Set the sound
A972: STA $129D,Y ;Frequency to the maximum
A975: LDA $128B,Y ;Value specified
A978: STA $12A0,Y
A97B: JMP $A9AE ;And update the SID registers
A97E: BCS $A994 ;Branch if sweep value is for oscillating sweep
A980: LDA $12A0,Y ;Compare the current sound frequency MSB
A983: CMP $128B,Y ;To the sound frequency MAX MSB
A986: BCC $A9AE ;If the current value is less, then branch
A988: BNE $A994 ;If it is not the same, then branch
A98A: LDA $129D,Y ;Compare the current sound frequency LSB
A98D: CMP $1288,Y ;To the sound frequency MAX MSB
A990: BCC $A9AE ;If the current value is less
A992: BEQ $A9AE ;Or if the value is equal, then branch
A994: CPX #$02 ;Is the sweep direction oscillating?
A996: BCC $A9A2 ;No, then branch
A998: JSR $A9DA ;Invert the sweep step value
A99B: LDA #$03 ;Set the flag for an
A99D: STA $1294,Y ;Oscillating sweep value
A9A0: BNE $A96F ;And set up the sound registers
A9A2: LDA $128E,Y ;Get the minimum frequency sweep LSB
A9A5: STA $129D,Y ;Save as the LSB of the sound's frequency
A9A8: LDA $1291,Y ;Get the minimum frequency sweep MSB
A9AB: STA $12A0,Y ;Save as the MSB of the sound's frequency
A9AE: LDX $7039,Y ;Get offset to the control register of the voice
A9B1: LDA $129D,Y ;Set up the current

```

```

A9B4: STA  $D400,X    ;Voice's frequency
A9B7: LDA  $12A0,Y    ;In the SID chip
A9BA: STA  $D401,X
A9BD: TYA                      ;Transfer the voice
A9BE: TAX                      ;Number to the X register
A9BF: LDA  $1282,X    ;If the LSB of the sound timer
A9C2: BNE  $A9C7      ;Is not zero, then branch
A9C4: DEC  $1285,X    ;Decrement the timer's MSB
A9C7: DEC  $1282,X    ;And the timer's LSB
A9CA: LDA  $1285,X    ;Check the timer's MSB value
A9CD: BPL  $A9D7      ;And if no overflow has occurred, then continue
                          ;To the next register
A9CF: LDA  #$08        ;Value to turn off voice
A9D1: LDX  $7039,Y    ;Get the offset to
A9D4: STA  $D404,X    ;Turn off the current voice
A9D7: JMP  $A92F      ;Continue to the next register

```

```

*****
*
* This routine is used to invert the frequency sweep
* step value. Ex: from $2000 - $6000
*
*****

```

```

A9DA: LDA  $1297,Y    ;Get the LSB of the step value
A9DD: EOR  #$FF       ;Invert it
A9DF: CLC                      ;Clear the carry for addition
A9E0: ADC  #$01       ;Add one to the inverted LSB
A9E2: STA  $1297,Y    ;And save it back
A9E5: LDA  $129A,Y    ;Get the MSB of the step value
A9E8: EOR  #$FF       ;Invert it
A9EA: ADC  #$00       ;And carry the flag to it
A9EC: STA  $129A,Y    ;And save it back
A9EF: RTS                      ;Exit

```

```

*****
*
* This routine is called to exit the BASIC IRQ by
* clearing the flag at $12FD which when set to
* anything but zero will block any IRQ call to the
* BASIC IRQ. This is an easy way to disable the BASIC
* IRQ routine. Just set $12FD to any value but zero.
*
*****

```

```

A9F0: DEC  $12FD      ;Clear the IRQ in progress flag
A9F3: RTS                      ;And exit

```

```

*****
*
* This routine is used to update the sprite
* coordinates. On entry, the accumulator holds the
* angle of movement divided by 90. Ex: Angle =
* 180 degree, then the accumulator = 2. An increment
* value is added to two sixteen bit count down timers
* and if the timer overflows, i.e. flips from $FFFF to
* $0000, the sprite coordinates are updated. The
* X register must hold the offset to the timer
* selected and the Y register must hold the current
* sprite number (0-7).
*
*****

```

```

A9F4: PHA           ;Save the angle of movement value onto the stack
A9F5: CLC           ;Clear the carry for addition
A9F6: LDA $1181,X  ;Add the LSB of the increment value
A9F9: ADC $1185,X  ;To the timer value
A9FC: STA $1185,X  ;And save it as the new timer LSB
A9FF: LDA $1182,X  ;Get the MSB of the increment value
AA02: ADC $1186,X  ;And add it to the MSB of the timer
AA05: STA $1186,X  ;Save the results as the new timer MSB
AA08: PLA           ;Get the angle of movement value back
AA09: BCC $AA1E    ;If the timer did not overflow, then exit
AA0B: LSR           ;Divide the angle of movement by 2
AA0C: LSR           ;And 2 again (divide by 4)
AA0D: LDA $11D6,Y  ;Get the proper coordinate
AA10: BCS $AA17    ;If the angle was up or left, then branch
AA12: ADC #$01     ;Add one to the selected coordinate
AA14: JMP $AA1B    ;And JMP to update the sprites coordinate
AA17: SBC #$01     ;If the angle was down or right, subtract 1 from
AA19: CMP $FF     ;The coordinate and check if it underflowed
AA1B: STA $11D6,Y  ;Save the new sprite coordinate
AA1E: RTS          ;And exit

```

```

*****
*
* BASIC STASH COMMAND
*
* Command syntax: STASH #bytes, insta, expa, expb
*
* NOTE: #bytes = the number of bytes to be sent
*        insta = INTERNAL Starting Address
*              of the C-128 RAM to be transferred
*        expb = EXPansion RAM Bank number (0-15)
*        expa = EXPansion RAM Starting Address
*

```

```

* This command is used to transfer the contents of the *
* C-128 RAM into the EXPANSION RAM. *
* *
*****

```

```

AA1F: LDA #132 ;Set the bit pattern to transfer C-128 RAM
AA21: JMP $AA2B ;To the RAM Disk and Jump to process

```

```

*****
*
* BASIC FETCH COMMAND *
*
* Command syntax:  FETCH #bytes, insta, expsa, expb *
*
* NOTE: #bytes = the number of bytes to be received *
*        intsa = INTernal Starting Address *
*              of the C-128 RAM to store the bytes *
*        expb = EXPansion RAM Bank number (0-15) *
*        expsa = EXPansion RAM Starting Address *
*
* This command is used to transfer the contents of the *
* EXPANSION RAM to the C-128 RAM. *
*
*****

```

```

AA24: LDA #$10000101 ;Set the bit value for executing a
AA26: JMP $AA2B ;Transfer from the RAM DISK to the C-128

```

```

*****
*
* BASIC SWAP COMMAND *
*
* Command syntax:  SWAP #bytes, insta, expsa, expb *
*
* NOTE: #bytes = the number of bytes to be swapped *
*        intsa = INTernal Starting Address *
*              of the C-128 RAM to be swapped *
*        expb = EXPansion RAM Bank number (0-15) *
*        expsa = EXPansion RAM Starting Address *
*
* This command is used to swap the contents of the *
* EXPANSION RAM with the contents of the C-128 RAM. *
*
*****

```

```

AA29: LDA #134 ;Set the bit pattern to swap memory between
AA2B: PHA ;The RAM disk and the 128 and save it on stack

```

```

AA2C: JSR $8812 ;Change number of bytes from ASCII to integer
;In Y,A
AA2F: JSR $A845 ;Enable Bank 15
AA32: STY $DF07 ;Save the number of bytes to be
AA35: STA $DF08 ;Transferred in DMADAC and DMADAH
AA38: JSR $880F ;Convert the internal starting address after the
;Comma into integer format in Y,A and if comma
;Is not found, then generate a 'SYNTAX' error
AA3B: JSR $A845 ;Enable Bank 15
AA3E: STY $DF02 ;Save the address of the C-128
AA41: STA $DF03 ;Internal RAM address to be accessed
AA44: JSR $880F ;Convert external starting address after comma
;Into integer format in A,Y and if comma is not
;Found, then generate a 'SYNTAX' error
AA47: JSR $A845 ;Enable Bank 15
AA4A: STY $DF04 ;Save the address of the expansion RAM
AA4D: STA $DF05 ;To be accessed
AA50: JSR $8809 ;Get the external RAM BANK number in X register
;(Ram Disk)
AA53: CPX #16 ;Get the Bank number to be accessed in the
AA55: BCS $AA65 ;C-128 and if it is 16 or greater, then branch
AA57: JSR $A845 ;Enable Bank 15
AA5A: STX $DF06 ;Save the Bank number to be accessed
AA5D: PLA ;Get the type of operation to perform
AA5E: TAY ;And place it into the Y register
AA5F: LDX $03D5 ;Get the Internal Bank number from/to where the
;Operation is to occur. (Inside the 128)
AA62: JMP $FF50 ;Perform the function (Direct Memory Access-DMA)

```

```

*****
*
*      Output 'ILLEGAL QUANTITY' error message
*
*****

```

```

AA65: JMP $7D28

```

```

*****
*
* This routine rounds the Floating Point Number in
* FAC1 and converts it to integer format in $64,$65.
*
*****

```

```

AA68: JSR $8C47
AA6B: JMP $8CC7

```

```

*****
*
*           Unused bytes from $AA6E to $AE62
*
*****

```

```

AA6E: .BYTE $FF,$FF,$FF
AE61: .BYTE $FF,$FF

```

```

*****
*
* This is the encrypted message that appears when you
* enter SYS 32800,123,45,6
*
*****

```

```

AE63: .BYTE $7B,$E9,$77,$6A,$5F,$5E,$5D,$BE
AE6B: .BYTE $21,$3D,$24,$37,$3F,$22,$55,$20
AE73: .BYTE $24,$4A,$30,$27,$3A,$4E,$2F,$35
AE7B: .BYTE $4D,$4C,$4F,$40,$47,$46,$68,$69
AE83: .BYTE $88,$15,$1F,$0C,$08,$1F,$0F,$19
AE8B: .BYTE $69,$5F,$71,$96,$05,$13,$11,$74
AE93: .BYTE $89,$05,$1E,$0D,$01,$43,$6D,$98
AE9B: .BYTE $06,$10,$13,$19,$67,$94,$1C,$05
AEA3: .BYTE $75,$37,$19,$EE,$70,$70,$1D,$F9
AEB3: .BYTE $E3,$6F,$7B,$6C,$78,$6F,$7F,$69
AEBB: .BYTE $19,$2F,$01,$E2,$6E,$6A,$05,$EC
AEC3: .BYTE $5E,$48,$5D,$15,$3F,$DA,$5C,$4A
AECB: .BYTE $56,$32,$09,$51,$4E,$58,$5C,$51
AED3: .BYTE $06,$2A,$CF,$5A,$4E,$40,$46,$23
AEDB: .BYTE $D3,$43,$4D,$41,$4E,$47,$08,$09
AEE3: .BYTE $E9,$36,$B0,$B6,$B4,$DE,$BC,$AE
AEEB: .BYTE $BE,$A1,$DD,$B4,$B8,$B8,$D2,$A0
AEF3: .BYTE $CB,$A7,$A8,$A3,$AA,$CE,$B9,$A4
AEFB: .BYTE $A6,$AF,$CF,$ED,$E7

```

```

*****
*
*           JUMP TABLE FOR FORMAT CONVERSIONS
*
*****

```

```

AF00: JMP $84B4 ;AYINT Convert a Floating Point number to an integer
AF03: JMP $793C ;GIVAYF Convert an integer to Floating Point format
AF06: JMP $8E42 ;FOUT Convert Floating Point to an ASCII string
AF09: JMP $8052 ;VAL1 Convert an ASCII string to Floating Point
AF0C: JMP $8815 ;GETADR Convert Floating Point to LSB/MSB address

```



```
*****
*
*           JUMP TABLE FOR MATH FUNCTIONS
*
*****
```

```
AF0F: JMP  $8C75 ;FLOATC Convert an address to Floating Point format
AF12: JMP  $882E ;FSUB   Subtract FAC1 from a value in memory
AF15: JMP  $8831 ;FSUBT  FAC2 - FAC1
AF18: JMP  $8845 ;FADD   FAC1 = FAC1 + MEM
AF1B: JMP  $8848 ;FADDT  FAC2 + FAC1
AF1E: JMP  $8A24 ;FMULT  Multiply a value in memory by FAC1
AF21: JMP  $8A27 ;FMULTT Multiply FAC1 by FAC2
AF24: JMP  $8B49 ;FDIV   Divide a value in memory by FAC1
AF27: JMP  $8B4C ;FDIVT  Divide FAC2 by FAC1
AF2A: JMP  $89CA ;LOG    Calculate the natural log of FAC1
AF2D: JMP  $8CFB ;INT    Take the integer portion of FAC1
AF30: JMP  $8FB7 ;SQR    Calculate the square root of FAC1
AF33: JMP  $8FFA ;NEGOP  Negate FAC1
AF36: JMP  $8FBE ;FPWR   Raise FAC2 to power of a value in memory
AF39: JMP  $8FC1 ;FPWRT  Raise FAC2 to the FAC1 power
AF3C: JMP  $9033 ;EXP    Calculate the EXP of FAC1
AF3F: JMP  $9409 ;COS    Calculate the cosine of FAC1
AF42: JMP  $9410 ;SIN    Calculate the sine of FAC1
AF45: JMP  $9459 ;TAN    Calculate the tangent of FAC1
AF48: JMP  $94B3 ;ATN    Calculate the arctangent of FAC1
AF4B: JMP  $8C47 ;ROUND  Round off the value in FAC1
AF4E: JMP  $8C84 ;ABS    Take the absolute value of FAC1
AF51: JMP  $8C57 ;SIGN   Test the sign of FAC1
AF54: JMP  $8C87 ;FCOMP  Compare FAC1 with a value in memory
AF57: JMP  $8437 ;RND 0  Generate a random Floating Point number
```

```
*****
*
*           JUMP TABLE FOR MEMORY MOVEMENT
*
*****
```

```
AF5A: JMP  $8AB4 ;CONUPK Transfer a value in RAM to FAC2
AF5D: JMP  $8A89 ;ROMUPK Transfer a value in ROM to FAC2
AF60: JMP  $7A85 ;MOVFRM Transfer a value in RAM to FAC1
AF63: JMP  $8BD4 ;MOVFM  Transfer a value in ROM to FAC1
AF66: JMP  $8C00 ;MOVMF  Transfer FAC1 to memory
AF69: JMP  $8C28 ;MOVFA  Transfer FAC2 to FAC1
AF6C: JMP  $8C38 ;MOVAF  Transfer FAC1 to FAC2
AF6F: JMP  $4828 ;OPTAB  Math operator vector table
AF72: JMP  $9B30 ;DRAWLN Draw a straight line
```

```

AF75: JMP $9BFB ;GPLOT Plot a single point
AF78: JMP $6750 ;CIRSUB Draw a circle or polygon
AF7B: JMP $5A9B ;RUN Perform RUN
AF7E: JMP $51F3 ;RUNC Reset the current text character pointer to
; the beginning of the program
AF81: JMP $51F8 ;CLEAR Perform CLR
AF84: JMP $51D6 ;NEW Perform NEW
AF87: JMP $4F4F ;LNKPRG Relink the program lines
AF8A: JMP $430A ;CRUNCH Tokenize the line in the input buffer
AF8D: JMP $5064 ;FNDLIN Search for a line number
AF90: JMP $4AF6 ;NEWSTT Set up the next program line for execution
AF93: JMP $78D7 ;EVAL Convert a number from ASCII text to a
; Floating Point number
AF96: JMP $77EF ;FRMEVL Evaluate the expression
AF99: JMP $5AA6 ;RUN A PROGRAM
AF9C: JMP $5A81 ;SETEXC Set the system to the PROGRAM mode
AF9F: JMP $50A0 ;LINGET Convert no. from ASCII to 2-byte line no.
AFA2: JMP $92EA ;GARBA2 Perform string garbage collection
AFA5: JMP $4DCD ;EXECUTE A LINE

```

```

*****
*
*           More unused bytes from $AFA8 to $AFFF
*
*
*****

```

```

AFA8: .BYTE $FF,$FF,$FF,.....

```

## **Appendices**



## APPENDIX A

COMMODORE 64 TO COMMODORE 128  
CROSS REFERENCE GUIDE

C64	ROUTINE COMMENTS	LABEL	C128
A000	COLD START IN THE C-128 THIS IS THE JUMP ADDRESS	CLDSTRT	4000
A00C	BASIC COMMAND ADDRESSES - 1	ADDRLST	46FC
A052	BASIC FUNCTION ADDRESSES	NFUN	47D8
A080	TABLE OF PRIORITY VALUES AND BASIC OPERATOR ADDRESSES	OPTAB	4828
A09E	LIST OF BASIC COMMAND/STATEMENTS (TEXT)	RESLST	4417
A19E	LIST OF BASIC ERROR MESSAGES	ERRTAB	484B
A38A	FIND FOR GOSUB SEARCH STACK	FNDFOR	4FAA
A3B8	CREATE A SPACE IN RAM FOR VARIABLE OR PROGRAM LINE	BLTU	7C66
A3FB	TEST STACK DEPTH (CHKSTK)	GETSTK	4FFE
A408	ENSURE THERE IS ENOUGH RAM FOR NEW LINE OR VARIABLE	REASON	5017
A437	ERROR MESSAGE GENERATOR WHERE X = THE ERROR NUMBER	ERROR	4D3C
A469	PRINT 'ERROR' AND THE LINE NUMBER OF THE ERROR IF ANY	PRTErr	4DA5
A474	PRINT 'READY.' AND SETS SYSTEM STATE TO DIRECT MODE	READY	4D37
A480	MAIN BASIC INPUT LOOP	MAIN	4DC3
A49C	ADD OR REPLACE A BASIC PROGRAM TEXT	MAIN1	4DE2
A533	RELINK THE LINES OF BASIC TEXT (LINE LINKS)	LINKPRG	4F4F
A560	INPUT A LINE TO THE INPUT BUFFER	INLIN	4F93
A579	TOKENIZE BASIC LINE	CRNCH	430A
A613	SEARCH BASIC TEXT FOR THE SPECIFIED LINE NUMBER	FNDLIN	5064
A642	BASIC NEW COMMAND	SCRATCH	51D6
A65E	BASIC CLR COMMAND	CLEAR	51F8
A68E	SET TXTPTR TO TXTTAB-1	RUNC	5254
A69C	BASIC LIST COMMAND	LIST	50E2
A717	PRINT BASIC TOKENS AS ASCII CHARACTERS	QPLOP	514E
A742	BASIC FOR COMMAND	FOR	5DF9
A7AE	SET UP NEXT STATEMENT	NEWSTT	4AF6
A7E4	EXECUTE NEXT BASIC STATEMENT	GONE	4A9F
A7ED	EXECUTE CURRENT BASIC STATEMENT (TOKEN IN THE ACC)	noLabel	4B74
A81D	BASIC RESTORE COMMAND	RESTOR	5ACA
A82C	TEST THE STOP KEY AND IF DEPRESSED THEN 'BREAK' ERROR	CHKSTP	4BC1
A82F	BASIC STOP COMMAND	STOP	4BCB
A831	BASIC END COMMAND	END	4BCD
A857	BASIC CONT COMMAND	CONT	5A60
A871	BASIC RUN COMMAND	RUN	5A9B
A883	BASIC GOSUB COMMAND	GOSUB	59CF
A8A0	BASIC GOTO COMMAND	GOTO	59DB

C64	ROUTINE COMMENTS	LABEL	C128
A8D2	BASIC RETURN COMMAND	RETURN	5262
A8EB	BASIC DATA COMMAND	DATA	528F
A906	SEARCH FOR THE NEXT STATEMENT FLAG ':' colon	DATAN	52A2
A909	SEARCH FOR END OF LINE FLAG (#\$00)	DATAL	52A5
A928	BASIC IF COMMAND	IF	52C5
A93B	BASIC REM COMMAND	REM	529D
A94B	BASIC ON COMMAND	ONGOTO	53A3
A96B	CONVERT ASCII LINE NUMBER TO INTEGER	LINGET	50A0
A9A5	BASIC LET COMMAND	LET	53C6
AA80	BASIC PRINT# COMMAND	PRINTN	553A
AA86	BASIC CMD COMMAND	CMD	5540
AAA0	BASIC PRINT COMMAND	PRINT	555A
AB1E	PRINT STRING FROM MEMORY	STROUT	55E2
AB4D	ERROR HANDLER FOR THE INPUT ROUTINE	DOAGIN	574D
AB7B	BASIC GET COMMAND	GET	5612
ABA5	BASIC INPUT# COMMAND	INPUTN	5648
ABBF	BASIC INPUT COMMAND	INPUT	5662
ABF9	PRINT THE INPUT PROMPT AND INPUT THE DATA	KEYBRD	569C
AC06	BASIC READ COMMAND	READ	56A9
AD1E	BASIC NEXT COMMAND	NEXT	57F4
AD8A	EVALUATE NUMERIC EXPRESSION	FRMNUM	77D7
AD8D	CHECK FOR DATA TYPE OF NUMERIC	CHKNUM	77DA
AD8F	CHECK FOR DATA TYPE OF STRING	CHKSTR	77DD
AD9E	EVALUATE EXPRESSION	FRMEVL	77EF
AE83	CONVERT SINGLE NUMERIC TERM FROM ASCII TO F.P.N.	EVAL	78D7
AEA8	CONSTANT-PI	PIVAL	78FE
AED4	BASIC NOT COMMAND	NOT	7930
AEF1	EVALUATE WITHIN PARENTHESES	PARCHK	7950
AEF7	CHECK FOR ")"	CHKCLS	7956
AEFA	CHECK FOR "("	CHKOPN	7959
AEFD	CHECK FOR ','	CHKCOM	795C
AEFF	CHECK FOR THE CHARACTER THAT'S IN THE ACCUMULATOR	CHKACC	795E
AF08	SYNTAX ERROR	SNERR	796C
AF2B	GET VALUE OF A VARIABLE	SVAR	7978
AFA7	SET UP REFERENCES	SFUN	4BF7
AFE6	BASIC OR COMMAND	OROP	4C86
AFE9	BASIC AND COMMAND	ANDOP	4C86
B016	BASIC <=>COMPARISONS FOR FLOATING POINT #S & STRINGS	DORE1	4CB6
B081	BASIC DIM COMMAND	DIM	587B
B08B	SEARCH FOR VARIABLE DESCRIPTOR;IF NOT FOUND CREATE IT	PTRGET	7AAF
B11D	CREATE A NEW 7 BYTE DESCRIPTOR	NOTFNS	7B46
B185	RETURN THE ADDRESS OF A VARIABLE CREATED/FOUND	FINPTR	7B4A
B1A5	FLOATING POINT CONSTANT -32768	N32768	849A
B1AA	CONVERT FPN TO A SIGNED INTEGER IN Y,A (LSB/MSB)	FPNINT	849F
B1B2	INPUT AND CONVERT A FPN TO A POSITIVE INTEGER	INTIDX	84A7

C64	ROUTINE COMMENTS	LABEL	C128
B1BF	FAC 1 INTEGER	AYINT	84B4
B1D1	SEARCH FOR AN ARRAY OR CREATE IT	ISARY	7CAB
B245	GENERATE 'BAD SUBSCRIPT' ERROR MESSAGE	BSERR	7D25
B248	GENERATE AN 'ILLEGAL QUANTITY' ERROR MESSAGE	FCERR	7D28
B37D	BASIC FRE FUNCTION	FREFN	8000
B391	INTEGER TO FAC 1 (GIVAYF)	GIVAYF	8C70
B39E	BASIC POS COMMAND	POS	84D0
B3A6	CHECK FOR DIRECT MODE (ERR)	ERRDIR	84D9
B3B3	BASIC DEF COMMAND	DEF	847A
B3E1	CHECKS FOR PROPER 'FN' SYNTAX	GETFNM	8528
B3F4	BASIC FN FUNCTION	FNDOER	853B
B465	BASIC STR\$ FUNCTION	STRD	85AE
B487	SET UP A STRING IN MEMORY	STRLIT	869A
B4F4	ALLOCATE SPACE IN MEMORY FOR A STRING (LENGTH = ACC.)	GETSPA	9299
B526	DISCARD UNUSED STRING BY SHIFTING GOOD STRING OVER IT	GARBAG	92EA
B5BD	IF THE CURRENT STRING IS THE HIGHEST IN MEMORY	GARBAG1	9383
B63D	CONCATENATE TWO STRINGS TOGETHER	CAT	870D
B67A	TRANSFER A STRING IN MEMORY	MOVINS	874E
B6A3	DISCARD UNWANTED STRING	FRESTR	877E
B6DB	DELETE A STRING FROM THE TEMPORARY STRING STACK	FRETMS	87E0
B6EC	BASIC CHR\$ FUNCTION	CHRD	85BF
B700	BASIC LEFT\$ FUNCTION	LFTD	85D6
B72C	BASIC RIGHT\$ FUNCTION	RIGHTD	860A
B737	BASIC MID\$ FUNCTION	MIDD	861C
B761	PULL STRING PARAMETERS OFF THE STACK	PREAM	864D
B77C	BASIC LEN FUNCTION	LEN	8668
B782	GET/SET STRING PARAMETERS	-----	866E
B78B	BASIC ASC FUNCTION	ASCFN	8677
B79B	CONVERT ASCII NUMBER TO A # VALUE 0-255 IN X REG.	GETBYTC	87F1
B7AD	BASIC VAL FUNCTION	VAL	804A
B7EB	GET POKE VALUES	GETADR	8803
B80D	BASIC PEEK FUNCTION	PEEK	80C5
B824	BASIC POKE COMMAND	POKE	80ED
B82D	BASIC WAIT COMMAND	FUWAIT	6C2D
B849	ADD 0.5 to FAC1	FADDH	8A0E
B850	SUBTRACT FAC1 FROM A NUMBER IN MEMORY FSUB	FSUB	882E
B853	BASIC SUBTRACTION COMMAND	FUSUBT	8831
B867	ADD FAC1 TO A NUMBER IN MEMORY	FADD	8A45
B86A	BASIC ADDITION COMMAND (+)	FADDT	8848
B947	COMPLEMENT FAC1	NEGFAC	8926
B97E	GENERATE AN 'OVERFLOW' ERROR	VERR	895D
B983	SINGLE BYTE MULTIPLY	MULTSHF	8962
B9BC	FLOATING POINT CONSTANT FOR LOG	FONE	899C
B9EA	BASIC LOG FUNCTION	LOG	89CA
BA28	FAC1=FAC1*MEMORY	FMULT	8A08

C64	ROUTINE COMMENTS	LABEL	C128
BA30	MULT FAC2*FAC1	FMULT1	8A0B
BA8C	TRANSFER MEMORY TO FAC2 (ARG)	CONUPK	8A89
BAB7	ADD THE EXPONENT OF FAC1 TO THE EXPONENT OF FAC2 (ARG)	MULTDIV	8AEC
BAE2	MULTIPLY FAC1 BY 10	MUL10	8B17
BAF9	CONSTANT 10 IN FLOATING POINT NUMBER FORMAT	TENC	8B2E
BAFE	DIVIDE BY 10 FAC1 = FAC1 / 10	DIV10	8B38
BB07	DIVIDE FAC 2/MEMORY	FDIV1	8B3F
BBOF	DIVIDE MEMORY/FAC 1	FDIV2	8B49
BB12	DIVIDE FAC 2/FAC 1	FDIVT	8B4C
BB8A	GENERATE 'DIVISION BY ZERO' ERROR MESSAGE	DZERR	8B33
BBA2	MEMORY TO FAC 1	MOVFM	8BD4
BBC7	MOVE A FPN FROM FAC1 TO TEMP1	-----	8BF9
BBCA	MOVE A FPN FROM FAC1 TO TEMP2	-----	8BFC
BBD0	MOVE FAC1 INTO A VARIABLE'S DESCRIPTOR	-----	8C00
BBFC	FAC 2 TO FAC 1	MOVFA	8C28
BC0C	ROUND NUMBER & MOVE FROM FAC1 TO FAC2	MOVAF	8C38
BC0F	FAC 1 TO FAC 2	MOVEF	8C3B
BC1B	ROUND OFF FAC 1	ROUND	8C47
BC2B	GET SIGN	SIGN	8C57
BC39	BASIC SGN FUNCTION	SGN	8C65
BC58	BASIC ABS FUNCTION	ABS	8C84
BC5B	COMPARE FAC 1 TO MEMORY	FCOMP	8C87
BC9B	FAC 1 TO INTEGER	QINT	8CC7
BCCC	BASIC INT FUNCTION	INT	8CFB
BCF3	ASCII TO FAC 1	FIN	8D22
BD7E	GET NEW ASCII DIGIT	FINLOG	8DB4
BDB3	CONSTANTS	NO999	8E17
BDC0	PRINT 'IN' FOLLOWED BY LINE NUMBER	INPRT	8E26
BDCD	OUTPUT NUMBER IN ASCII DEC. DIGITS	LINPRT	8E32
BDDD	FAC 1 TO ASCII	FOUT	8E42
BF11	MORE CONSTANTS	FHALF	8F76
BF1C	POWERS OF - 10 CONSTANTS TABLE	FOUTBL	8F81
BF3A	TABLE OF CONSTANTS FOR TI\$ CONVERSION	FDCEND	8FDF
BF71	BASIC SQR FUNCTION	SQR	8FB7
BF7B	BASIC EXPONENTIATION FUNCTION '^'	FPWRT	8FC1
BFB4	BASIC NEGATION FUNCTION	NEGOP	8FFA
BFBF	MORE CONSTANTS	EXPCON	9005
BFED	BASIC EXP FUNCTION	EXP	9033
E043	FUNCTION SERIES EVALUATION SUBROUTINE NUMBER 1	POLY1	9086
E059	FUNCTION SERIES EVALUATION SUBROUTINE NUMBER 2	POLY2	909C
E08D	MULTIPLICATIVE CONSTANT FOR RND	RMULC	8490
E092	ADDITIVE CONSTANT FOR RND	RADDC	8495
E10C	BSOUT - OUTPUT A CHAR. TO THE CURRENT OUTPUT DEVICE	BSOUT	90DF
E112	BASIN - INPUT A CHAR. FROM THE CURRENT INPUT DEVICE	BASIN	90E5
E118	CHKOUT - SET FILE TO OUTPUT	CHKOUT	90EB



C64	ROUTINE COMMENTS	LABEL	C128
E11E	CHKIN - SET FILE TO INPUT	CHKIN	90FD
E124	GETIN - GET A CHARACTER FROM THE CURRENT INPUT DEVICE	GETIN	9109
E12A	BASIC SYS COMMAND	SYS	5885
E156	BASIC SAVE COMMAND	SAVE	9112
E165	BASIC VERIFY COMMAND	VERIFY	9129
E168	BASIC LOAD COMMAND	LOAD	912C
E1BE	BASIC OPEN COMMAND	OPEN	918D
E1C7	BASIC CLOSE COMMAND	CLOSE	919A
E1D4	SET PARAMETERS FOR LOAD,VERIFY,SAVE	SETPAR	91AE
E200	SKIP COMMA & GET INTEGER IN X	SKPCOM	91DD
E206	GET CURRENT CHARACTER & CHECK FOR END OF LINE	ENDTRM	91E3
E20E	CHECK FOR COMMA AND ENSURE THAT A CHARACTER FOLLOWS	CHKCOM1	91EB
E219	SET PARAMETERS FOR OPEN & CLOSE	SETPAR1	91F6
E264	BASIC COS FUNCTION	COS	9409
E26B	BASIC SIN FUNCTION	SIN	9410
E2B4	BASIC TAN FUNCTION	TAN	9459
E2E0	CONSTANTS FOR TRIG FUNCTIONS	PI2	9485
E2E5	5 BYTE FLOATING POINT REP. OF CONSTANT 2*PI	TWOPI	948A
E2EA	5 BYTE FLOATING POINT REP. OF CONSTANT 1/4	FR4	948F
E2EF	TABLE OF CONSTANTS FOR EVAL. OF SIN,COS,TAN	SINCON	9494
E30E	PERFORM ATN	ATN	94B3
E33E	CONSTANTS FOR ATN	ATNCON	94E3
E37B	BASIC NMI JUMP IN ADDRESS	NMI	4009
E38B	ERROR MESSAGE HANDLER	ERROR	4D3F
E334	COLD START BASIC	CLDSTRT	4D23
E3A2	COPY OF CHRGET ROUTINE	INITAT	4279
E3BF	INITIALIZE BASIC	INIT	4045
E447	TABLE OF BASIC VECTORS	SYSVEC	4267
E453	TRANSFER BASIC VECTORS FROM ROM TO RAM	VECRAM	4251



## APPENDIX B

## BASIC RESERVED WORD TABLES WITH TOKENS AND COMMAND ADDRESSES

### BASIC Commands and Functions

RESERVED KEYWORD	ROUTINE ADDRESS	TOKEN	D/P/B	RESERVED KEYWORD	ROUTINE ADDRESS	TOKEN	D/P/B
ABS	\$8C84	-- \$B6	B	APPEND	\$A134	\$FE \$0E	B
ASC	\$8677	-- \$C6	B	ATN	\$94B3	-- \$C1	B
AUTO	\$5975	-- \$DC	D	BACKUP	\$A37C	-- \$F6	B
BANK	\$6BC9	\$FE \$02	B	BEGIN	\$796C	\$FE \$18	B
BEND	\$528F	\$FE \$19	B	BLOAD	\$A218	\$FE \$11	B
BOOT	\$7335	\$FE \$1B	B	BOX	\$62B7	-- \$E1	B
BSAVE	\$A1C8	\$FE \$10	B	BUMP	\$837C	\$CE \$03	B
CATALOG	\$A07E	\$FE \$0C	B	CHAR	\$67D7	-- \$E0	B
CHR\$	\$85BF	-- \$C7	B	CIRCLE	\$668E	-- \$E2	B
CLOSE	\$919A	-- \$A0	B	CLR	\$51F8	-- \$9C	B
CMD	\$55F0	-- \$9D	B	COLLECT	\$A32F	-- \$F3	B
COLLISION	\$7164	\$FE \$17	B	COLOR	\$69E2	-- \$E7	B
CONCAT	\$A362	\$FE \$13	B	CONT	\$5A60	-- \$9A	D
COPY	\$A346	-- \$F4	B	COS	\$9409	-- \$BE	B
DATA	\$528F	-- \$83	P	DCLEAR	\$A322	\$FE \$15	B
DCLOSE	\$A16F	\$FE \$0F	B	DEC	\$8076	-- \$D1	B
DEF	\$84FA	-- \$96	P	DELETE	\$5E87	-- \$F7	D
DIM	\$587B	-- \$86	B	DIRECTORY	\$A07E	-- \$EE	B
DLOAD	\$A1A7	-- \$F0	B	DO	\$5FE0	-- \$EB	B
DOPEN	\$A11D	\$FE \$0D	B	DRAW	\$6797	-- \$E5	B
DS	----	-- --	B	DS\$	----	-- --	B
DSAVE	\$A18C	-- \$EF	B	DVERIFY	\$A1A4	\$FE \$14	B
EL	----	-- --	B	ELSE/BEND	\$5391	-- \$D5	B
END	\$4BCD	-- \$80	B	ENVELOPE	\$70C1	\$FE \$0A	B
ER	----	-- --	B	ERR\$	\$80F6	-- \$D3	B
EXIT	\$6039	-- \$ED	B	EXP	\$9033	-- \$BD	B
FAST	\$77B3	\$FE \$25	B	FETCH	\$AA24	\$FE \$21	B
FILTER	\$7046	\$FE \$03	B	FN	\$853B	-- \$A5	B
FOR	\$5DF9	-- \$81	B	FRE	\$8000	-- \$B8	B
GET	\$5612	-- \$A1	P	GETKEY	\$561F	-- --	P
GET#	\$5625	-- --	P	GO	\$5A3D	-- \$CB	B
GO64	\$5A54	-- --	B	GOSUB	\$59CF	-- \$8D	B
GOTO	\$59DB	-- \$89	B	GRAPHIC	\$6B5A	-- \$DE	B

RESERVED	ROUTINE				RESERVED	ROUTINE			
KEYWORD	ADDRESS	TOKEN	D/P/B		KEYWORD	ADDRESS	TOKEN	D/P/B	
GSHAPE	\$658D	-- \$E3	B		HEADER	\$A267	-- \$F1	B	
HELP	\$5986	-- \$EA	B		HEX\$	\$8142	-- \$D2	B	
IF	\$52C5	-- \$8B	B		INPUT	\$5662	-- \$85	P	
INPUT#	\$5648	-- \$84	P		INSTR	\$99C1	-- \$D4	B	
INT	\$8CFB	-- \$B5	B		JOY	\$99C1	-- \$CF	B	
KEY	\$610A	-- \$F9	B		LEFT\$	\$85D6	-- \$C8	B	
LEN	\$8668	-- \$C3	B		LET	\$53C6	-- \$88	B	
LIST	\$50E2	-- \$9B	B		LOAD	\$912C	-- \$93	B	
LOCATE	\$6955	-- \$E6	B		LOG	\$89CA	-- \$BC	B	
LOOP	\$608A	-- \$EC	B		MID\$	\$861C	-- \$CA	B	
MONITOR	\$B000	-- \$FA	B		MOVSPR	\$6CC6	FE \$06	B	
NEW	\$51D6	-- \$A2	B		NEXT	\$57F4	-- \$82	B	
OFF	\$4845	\$FE \$24	N/I		ON	\$53A3	-- \$91	B	
OPEN	\$918D	-- \$9F	B		PAINT	\$61A8	-- \$DF	B	
PEEK	\$80C5	-- \$C2	B		PEN	\$82AE	\$CE \$04	B	
PI	----	-- \$FF	B		PLAY	\$6DE1	\$FE \$04	B	
POINTER	\$82FA	\$CE \$0A	B		POKE	\$80E5	-- \$97	B	
POS	\$84D0	-- \$B9	B		POT	\$824D	\$CE \$02	B	
PRINT	\$555A	-- \$99	B		PRINT#	\$553A	-- \$98	B	
PRINT USING	\$9520	-- \$FB	B		PUDEF	\$5F34	-- \$DD	B	
QUIT	\$4845	\$FE \$1E	N/I		RCLR	\$819B	-- \$CD	B	
RDOT	\$9B0C	-- \$D0	B		READ	\$56A9	-- \$87	B	
RECORD	\$A2D7	\$FE \$12	B		REM	\$529D	-- \$8F	B	
RENAME	\$A36E	-- \$F5	B		RENUMBER	\$5AF8	-- \$F8	D	
RESTORE	\$5ACA	-- \$8C	B		RESUME	\$5F62	-- \$D6	P	
RETURN	\$5262	-- \$8E	P		RGR	\$8182	-- \$CC	B	
RIGHT\$	\$860A	-- \$C9	B		RND	\$8434	-- \$BB	B	
RREG	\$58BD	\$FE \$09	B		RSPCOLOR	\$8361	\$CE \$07	B	
RSPPPOS	\$8397	\$CE \$05	B		RSPRITE	\$831E	\$CE \$06	B	
RUN	\$5A9B	-- \$8A	B		RWINDOW	\$8407	\$CE \$09	B	
SAVE	\$9112	-- \$94	B		SCALE	\$6960	-- \$E9	B	
SCNCLR	\$6A79	-- \$E8	B		SCRATCH	\$A2A1	-- \$F2	B	
SGN	\$8C65	-- \$B4	B		SIN	\$9410	-- \$BF	B	
SLEEP	\$6BD7	\$FE \$0B	B		SLOW	\$77C4	\$FE \$26	B	
SOUND	\$71EC	-- \$DA	B		SPC (	\$55B9	-- \$A6	B	
SPRCOLOR	\$7190	\$FE \$08	B		SPRDEF	\$7372	\$FE \$1D	B	
SPRITE	\$6C4F	\$FE \$07	B		SPRSAV	\$76EC	\$FE \$16	B	
SQR	\$8FB7	-- \$BA	B		SSHAPE	\$642B	-- \$E4	B	
ST	----	N/A	B		STASH	\$AA1F	\$FE \$1F	B	
STEP	----	-- \$A9	B		STOP	\$4BCB	-- \$90	B	
STR\$	\$85AE	-- \$C4	B		SWAP	\$AA29	\$FE \$23	B	
SYS	\$5885	-- \$9E	B		TAB (	\$55B9	-- \$A3	B	
TAN	\$9459	-- \$C0	B		TEMPO	\$6FD7	\$FE \$05	B	
THEN	----	-- \$A7	B		TI	----	N/A	B	
TI\$	----	N/A	B		TO	----	-- \$A4	B	

RESERVED	ROUTINE				RESERVED	ROUTINE			
KEYWORD	ADDRESS	TOKEN	D/P/B		KEYWORD	ADDRESS	TOKEN	D/P/B	
TRAP	\$5F4D	-- \$D7	B		TROFF	\$58B7	-- \$D9	B	
TRON	\$58B4	-- \$D8	B		UNTIL	\$600B	-- \$FC	B	
USING	\$9520	-- \$FB	B		USR	\$1218	-- \$B7	B	
VAL	\$804A	-- \$C5	B		VERIFY	\$9129	-- \$95	B	
VOL	\$71C5	-- \$DB	B		WAIT	\$6C2D	-- \$92	B	
WHILE	\$60BC	-- \$FD	B		WIDTH	\$71B6	\$FE \$1C	B	
WINDOW	\$72CC	\$FE \$1A	B		XOR	\$83E1	\$CE \$08	B	

### BASIC Operators

	OPERATOR	ADDRESS	TOKEN	PRIORITY CODE	D/P/B
	+ (ADDITION)	\$8848	-- \$AA	\$79	B
	- (SUBTRACTION)	\$8831	-- \$AB	\$79	B
	* (MULTIPLICATION)	\$8A27	-- \$AC	\$7B	B
	/ (DIVISION)	\$8B4C	-- \$AD	\$7B	B
	^ (EXPONENTIAL)	\$8FC1	-- \$AE	\$7F	B
	AND	\$4C89	-- \$AF	\$50	B
	OR	\$4C86	-- \$B0	\$46	B
	NOT	\$7930	-- \$A8	\$5A	B
	- (UNARY MINUS) (NEGATIVE )	\$8FFA	-----	\$7D	B
RELATIONAL OPERATORS	< (LESS THAN)	\$4CB6	-- \$B3	\$64	B
	> (GREATER THAN)		-- \$B1		B
	= (EQUAL TO)		-- \$B2		B

D = DIRECT MODE  
 P = PROGRAM MODE  
 B = BOTH DIRECT AND PROGRAM MODES  
 N/I = NOT IMPLEMENTED  
 N/A = NOT APPLICABLE



## APPENDIX C

### Hexadecimal/Decimal Conversion Table

The following table can be used as a tool by both the BASIC and machine language programmer alike. The BASIC programmer will appreciate the columns that deal with the character sets and the machine language programmer will appreciate the hexadecimal to decimal conversion columns as well as the machine code instruction column.

For those of you who have a hard time converting a one byte or two byte hexadecimal number to decimal, it is an easy matter when using this table. In order to convert a one byte HEX number, find the HEX number that you wish to convert in the HEX column and look across the page to the LSB column to find the decimal equivalent. For example, to convert the HEX value of \$20, find that value in the HEX column and look across the page to the LSB column. In that column will be the decimal equivalent of the HEX value \$20 which is 32.

In order to convert a two byte number or address, first divide the number or address into two one byte numbers. The first number will be the MSB (Most Significant Byte) and the second number will be the LSB (Least Significant Byte). Now find the HEX value of the MSB in the HEX column and get the decimal equivalent. Then find the HEX value of the LSB in the HEX column and get the decimal equivalent. In order to finish the conversion, add the decimal value of the MSB to the decimal value of the LSB. The sum of the two values will be the decimal equivalent of the two byte address or number that you wanted to convert.

For example, if the value of the number was \$FF0A, after dividing the number into two one byte numbers, you would have the HEX value of \$FF for the MSB and the HEX value of \$0A for the LSB. Now find the HEX value of the MSB in the HEX column and get the decimal equivalent from the MSB column. The equivalent decimal value of \$FF as the MSB of a number is 65280. Next, find the HEX value of the LSB in the HEX column and get its decimal equivalent from the LSB column. This time, the equivalent decimal value of \$0A as the LSB of a number is 10. In order to complete the conversion, add the decimal equivalent of the MSB (65280) to the decimal equivalent of the LSB (10) to obtain the value of 65290. Therefore the decimal equivalent of \$FF0A is 65290.

To convert a decimal number to its hexadecimal equivalent, first find the number in the MSB column that comes the closest to (but not larger than) the number you wish to convert. Look in the HEX column and write down that value; this is the MSB of the final number. Then subtract that value from your original number and find the result in the HEX column, this is the LSB of the number.

For example, to convert the number 63248 to its hex equivalent, find the number closest to it without going over it in the HEX column. This would be 63232 or F7—this is the MSB. Subtract that value from the original number (63248-63232), which would give you 16 or \$10 for the LSB. Combine the two numbers, and you have the result: \$F710.



HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
00	0	0	00000000	@	@		BRK	
01	1	256	00000001	A	a		ORA(I,X)	indirect,X indexed
02	2	512	00000010	B	b		---	
03	3	768	00000011	C	c		---	
04	4	1024	00000100	D	d		---	
05	5	1280	00000101	E	e	WHITE	ORA	zero page
06	6	1536	00000110	F	f		ASL	zero page
07	7	1792	00000111	G	g		---	
08	8	2048	00001000	H	h		PHP	
09	9	2304	00001001	I	i		ORA #	immediate
0A	10	2560	00001010	J	j		ASL A	accumulator
0B	11	2816	00001011	K	k		---	
0C	12	3072	00001100	L	l		---	
0D	13	3328	00001101	M	m	RETURN	ORA	absolute
0E	14	3584	00001110	N	n	lower	ASL	absolute
0F	15	3840	00001111	O	o		---	
10	16	4096	00010000	P	p		BPL	
11	17	4352	00010001	Q	q	CSR Dn	ORA(I),Y	indirect,Y indexed
12	18	4608	00010010	R	r	RVSON	---	
13	19	4864	00010011	S	s	HOME	---	
14	20	5120	00010100	T	t	DELETE	---	
15	21	5376	00010101	U	u		ORA Z,X	zero page,X indexed
16	22	5632	00010110	V	v		ASL Z,X	zero page,X indexed
17	23	5888	00010111	W	w		---	
18	24	6144	00011000	X	x		CLC	
19	25	6400	00011001	Y	y		ORA Y	absolute,Y indexed
1A	26	6656	00011010	Z	z			

HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
1B	27	9984	00100111	'	'	'	---	
1C	28	7168	00011100	£	£	RED	---	
1D	29	7424	00011101	]	]	CSR Rt	ORA X	absolute,X indexed
1E	30	7680	00011110	^	^	GRN	ASL X	absolute,X indexed
1F	31	7936	00011111	<-	<-	BLUE	---	
20	32	8192	00100000	SPACE	SPACE	SPACE	JSR	
21	33	8448	00100001	!	!	!	AND (I,X)	indirect,X indexed
22	34	8704	00100010	"	"	"	---	
23	35	8960	00100011	#	#	#	---	
24	36	9216	00100100	\$	\$	\$	BIT Z	zero page
25	37	9472	00100101	%	%	%	AND Z	zero page
26	38	9728	00100110	&	&	&	ROL Z	zero page
27	39	9984	00100111	'	'	'	---	
28	40	10240	00101000	(	(	(	PLP	
29	41	10496	00101001	)	)	)	AND#	immediate
2A	42	10752	00101010	*	*	*	ROL A	accumulator
2B	43	11008	00101011	+	+	+	---	
2C	44	11264	00101100	,	,	,	BIT	absolute
2D	45	11520	00101101	-	-	-	AND	absolute
2E	46	11776	00101110	.	.	.	ROL	absolute
2F	47	12032	00101111	/	/	/	---	
30	48	12288	00110000	0	0	0	BMI	
31	49	12544	00110001	1	1	1	AND (I),Y	indirect,Y indexed
32	50	12800	00110010	2	2	2	---	
33	51	13056	00110011	3	3	3	---	
34	52	13312	00110100	4	4	4	---	























HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
35	53	13568	00110101	5	5	5	AND Z,X	zero page,X indexed
36	54	13824	00110110	6	6	6	ROL Z,X	zero page,X indexed
37	55	14080	00110111	7	7	7	---	
38	56	14336	00111000	8	8	8	SEC	
39	57	14592	00111001	9	9	9	AND Y	absolute,Y indexed
3A	58	14848	00111010	:	:	:	---	
3B	59	15104	00111011	;	;	;	---	
3C	60	15360	00111100	<	<	<	---	
3D	61	15616	00111101	=	=	=	AND X	absolute,X indexed
3E	62	15872	00111110	>	>	>	ROL X	absolute,X indexed
3F	63	16128	00111111	?	?	?	---	
40	64	16384	01000000	<input type="checkbox"/>	<input type="checkbox"/>		RTI	
41	65	16640	01000001	<input type="checkbox"/>	A	A	EOR(I,X)	indirect,X indexed
42	66	16896	01000010	<input type="checkbox"/>	B	B	---	
43	67	17152	01000011	<input type="checkbox"/>	C	C	---	
44	68	17408	01000100	<input type="checkbox"/>	D	D	---	
45	69	17664	01000101	<input type="checkbox"/>	E	E	EOR Z	zero page
46	70	17920	01000110	<input type="checkbox"/>	F	F	LSR Z	zero page
47	71	18176	01000111	<input type="checkbox"/>	G	G	---	
48	72	18432	01001000	<input type="checkbox"/>	H	H	PHA	
49	73	18688	01001001	<input type="checkbox"/>	I	I	EOR #	immediate
4A	74	18944	01001010	<input type="checkbox"/>	J	J	LSR A	accumulator
4B	75	19200	01001011	<input type="checkbox"/>	K	K	---	
4C	76	19456	01001100	<input type="checkbox"/>	L	L	JMP	absolute
4D	77	19712	01001101	<input checked="" type="checkbox"/>	M	M	EOR	absolute
4E	78	19968	01001110	<input checked="" type="checkbox"/>	N	N	LSR	absolute

HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
4F	79	20224	01001111			O	O	---
50	80	20480	01010000			P	P	BVC
51	81	20736	01010001			Q	Q	EOR(I),Y (indirect),Y indexed
52	82	20992	01010010			R	R	---
53	83	21248	01010011			S	S	---
54	84	21504	01010100			T	T	---
55	85	21760	01010101			U	U	EOR Z,X zero page,X indexed
56	86	22016	01010110			V	V	LSR Z,X zero page,X indexed
57	87	22272	01010111			W	W	---
58	88	22528	01011000			X	X	CLI
59	89	22784	01011001			Y	Y	EOR Y absolute,Y indexed
5A	90	23040	01011010			Z	Z	---
5B	91	23296	01011011			[	[	---
5C	92	23552	01011100			£	£	---
5D	93	23808	01011101			]	]	EOR X absolute,X indexed
5E	94	24064	01011110			↑	↑	LSR X absolute,X indexed
5F	95	24320	01011111			↑	↑	---
60	96	24576	01100000					RTS
61	97	24832	01100001					ADC(I,X) indirect,X indexed
62	98	25088	01100010					---
63	99	25344	01100011					---
64	100	25600	01100100					---
65	101	25856	01100101					ADC Z zero page
66	102	26112	01100110					ROR Z zero page
67	103	26368	01100111					---
68	104	26624	01101000					PLA

HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
69	105	26880	01101001				ADC #	immediate
6A	106	27136	01101010				ROR A	accumulator
6B	107	27392	01101011				---	
6C	108	27648	01101100				JMP (I)	indirect
6D	109	27904	01101101				ADC	absolute
6E	110	28160	01101110				ROR	absolute
6F	111	28416	01101111				---	
70	112	28672	01110000				BVS	
71	113	28928	01110001				ADC (I), Y	indirect, Y indexed
72	114	29184	01110010				---	
73	115	29440	01110011				---	
74	116	29696	01110100				---	
75	117	29952	01110101				ADC Z, X	zero page, X indexed
76	118	30208	01110110				ROR Z, X	zero page, X indexed
77	119	30464	01110111				---	
78	120	30720	01111000				SEI	
79	121	30976	01111001				ADC Y	absolute, Y indexed
7A	122	31232	01111010				---	
7B	123	31488	01111011				---	
7C	124	31744	01111100				---	
7D	125	32000	01111101				ADC X	absolute, X indexed
7E	126	32256	01111110				ROR X	absolute, X indexed
7F	127	32512	01111111				---	
80	128	32768	10000000				---	
81	129	33024	10000001			ORNG	STA (I, X)	indirect, X indexed
82	130	33280	10000010				---	

HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
83	131	33536	10000011				---	
84	132	33792	10000100				STY Z	zero page
85	133	34048	10000101			f1	STA Z	zero page
86	134	34304	10000110			f3	STX Z	zero page
87	135	34560	10000111			f5	---	
88	136	34816	10001000			f7	DEY	
89	137	35072	10001001			f2	---	
8A	138	35328	10001010			f4	TXA	
8B	139	35584	10001011			f6	---	
8C	140	35840	10001100			f8	STY	absolute
8D	141	36096	10001101				STA	absolute
8E	142	36352	10001110			UPPER	STX	absolute
8F	143	36608	10001111				---	
90	144	36864	10010000			BLACK	BCC	
91	145	37120	10010001			CSR UP	STA(I),Y	indirect,Y indexed
92	146	37376	10010010			RVS OFF	---	
93	147	37632	10010011			CLR	---	
94	148	37888	10010100			INS	STY Z,X	zero page,X indexed
95	149	38144	10010101			BROWN	STA Z,X	zero page,X indexed
96	150	38400	10010110			LT RED	STX Z,Y	zero page,Y indexed
97	151	38656	10010111			GRAY1	---	
98	152	38912	10011000			GRAY2	TYA	
99	153	39168	10011001			LT GRN	STA Y	absolute,Y indexed
9A	154	39424	10011010			LT BLUE	TXS	
9B	155	39680	10011011			GRAY3	---	
9C	156	39936	10011100			PURPLE	---	

NOTE: CHARUP and CHARDWN \$80 to \$FF (128 to 255) are reverse-video versions of CHARUP and CHARDN \$00 to \$7F.

HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
9D	157	40192	10011101	NOTE: CHARUP and CHARDWN \$80 to \$FF (128 to 255) are reverse-video versions of CHARUP and CHARDN \$00 to \$7F.		CSR LFT	STA X	absolute,X indexed
9E	158	40448	10011110			YEL	---	
9F	159	40704	10011111			CYAN	---	
A0	160	40960	10100000			SPACE	LDY #	immediate
A1	161	41216	10100001				LDA (I,X)	indirect,X indexed
A2	162	41472	10100010				LDX #	immediate
A3	163	41728	10100011				---	
A4	164	41984	10100100				LDY Z	zero page
A5	165	42240	10100101				LDA Z	zero page
A6	166	42496	10100110				LDX Z	zero page
A7	167	42752	10100111				---	
A8	168	43008	10101000				TAY	
A9	169	43264	10101001				LDA #	immediate
AA	170	43520	10101010				TAX	
AB	171	43776	10101011				---	
AC	172	44032	10101100				LDY	absolute
AD	173	44288	10101101				LDA	absolute
AE	174	44544	10101110				LDX	absolute
AF	175	44800	10101111			---		
B0	176	45056	10110000			BCS		
B1	177	45312	10110001			LDA (I),Y	indirect,Y indexed	
B2	178	45568	10110010			---		
B3	179	45824	10110011			---		
B4	180	46080	10110100			LDY Z,X	zero page,X indexed	
B5	181	46336	10110101			LDA Z,X	zero page,X indexed	
B6	182	46592	10110110			LDX Z,Y	zero page,Y indexed	

HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
B7	183	46848	10110111				---	
B8	184	47104	10111000				CLV	
B9	185	47360	10111001				LDA Y	absolute, Y indexed
BA	186	47616	10111010				TSX	
BB	187	47872	10111011				---	
BC	188	48128	10111100				LDY X	absolute, X indexed
BD	189	48384	10111101				LDA X	absolute, X indexed
BE	190	48640	10111110				LDX Y	absolute, Y indexed
BF	191	48896	10111111				---	
C0	192	49152	11000000				CPY #	immediate
C1	193	49408	11000001				CMP (I), X	indirect, X indexed
C2	194	49664	11000010				---	
C3	195	49920	11000011				---	
C4	196	50176	11000100				CPY Z	zero page
C5	197	50432	11000101				CMP Z	zero page
C6	198	50688	11000110				DEC Z	zero page
C7	199	50944	11000111				---	
C8	200	51200	11001000				INY	
C9	201	51456	11001001				CMP #	immediate
CA	202	51712	11001010				DEX	
CB	203	51968	11001011				---	
CC	204	52224	11001100				CPY	absolute
CD	205	52480	11001101				CMP	absolute
CE	206	52736	11001110				DEC	absolute
CF	207	52992	11001111				---	
D0	208	53248	11010000				BNE	

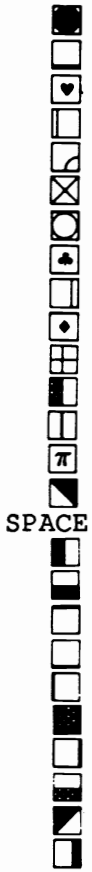
NOTE: CHARUP and CHARDWN \$80 to \$FF (128 to 255) are reverse-video versions of CHARUP and CHARDN \$00 to \$7F.





HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
D1	209	53504	11010001			█	CMP (I),Y	indirect,Y indexed
D2	210	53760	11010010			▣	---	
D3	211	54016	11010011			▣	---	
D4	212	54272	11010100			▣	---	
D5	213	54528	11010101			▣	CMP Z,X	zero page,X indexed
D6	214	54784	11010110			▣	DEC Z,X	zero page,X indexed
D7	215	55040	11010111			▣	---	
D8	216	55296	11011000			▣	CLD	
D9	217	55552	11011001			▣	CMP Y	absolute,Y indexed
DA	218	55808	11011010			▣	---	
DB	219	56064	11011011			▣	---	
DC	220	56320	11011100			▣	---	
DD	221	56576	11011101			▣	CMP X	absolute,X indexed
DE	222	56832	11011110			▣	DEC X	absolute,X indexed
DF	223	57088	11011111			▣	---	
E0	224	57344	11100000			▣	CPX #	immediate
E1	225	57600	11100001			▣	SBC (I),X	indirect,Y indexed
E2	226	57856	11100010			▣	---	
E3	227	58112	11100011			▣	---	
E4	228	58368	11100100			▣	CPX Z	zero page
E5	229	58624	11100101			▣	SBC Z	zero page
E6	230	58880	11100110			▣	INC Z	zero page
E7	231	59136	11100111			▣	---	
E8	232	59392	11101000			▣	INX	
E9	233	59648	11101001			▣	SBC #	immediate
EA	234	59904	11101010			▣	NOP	

NOTE: CHARUP and CHARDWN \$80 to \$FF (128 to 255) are reverse-video versions of CHARUP and CHARDN \$00 to \$7F.



HEX	LSB	MSB	BINARY	CHARUP	CHARDWN	ASCII	INS	ADDRESSING MODE
EB	235	60160	11101011				---	
EC	236	60416	11101100				CPX	absolute
ED	237	60672	11101101				SBC	absolute
EE	238	60928	11101110				INC	absolute
EF	239	61184	11101111				---	
F0	240	61440	11110000				BEQ	
F1	241	61696	11110001				SBC(I),Y	indirect,Y indexed
F2	242	61952	11110010				---	
F3	243	62208	11110011				---	
F4	244	62464	11110100				---	
F5	245	62720	11110101				SBC Z,X	zero page,X indexed
F6	246	62976	11110110				INC Z,X	zero page,X indexed
F7	247	63232	11110111				---	
F8	248	63488	11111000				SED	
F9	249	63744	11111001				SBC Y	absolute,Y indexed
FA	250	64000	11111010				---	
FB	251	64256	11111011				---	
FC	252	64512	11111100				---	
FD	253	64768	11111101				SBC X	absolute,X indexed
FE	254	65024	11111110				INC X	absolute,X indexed
FF	255	65280	11111111				---	

NOTE: CHARUP and CHARDWN \$80 to \$FF (128 to 255) are reverse-video versions of CHARUP and CHARDN \$00 to \$7F.



## APPENDIX D

## MMU Values

VALUE	IN	MMU-CRL	0400	4000	8000	D000	C000
HEX		BINARY	3FFF	7FFF	BFFF	DFFF	FFFF
0000		00000000	RAM 0	BASIC	BASIC	I/O	KERN
0001		00000001	RAM 0	BASIC	BASIC	CHAR	KERN
0002		00000010	RAM 0	RAM 0	BASIC	I/O	KERN
0003		00000011	RAM 0	RAM 0	BASIC	CHAR	KERN
0004		00000100	RAM 0	BASIC	INT	I/O	KERN
0005		00000101	RAM 0	BASIC	INT	CHAR	KERN
0006		00000110	RAM 0	RAM 0	INT	I/O	KERN
0007		00000111	RAM 0	RAM 0	INT	CHAR	KERN
0008		00001000	RAM 0	BASIC	EXT	I/O	KERN
0009		00001001	RAM 0	BASIC	EXT	CHAR	KERN
000A		00001010	RAM 0	RAM 0	EXT	I/O	KERN
000B		00001011	RAM 0	RAM 0	EXT	CHAR	KERN
000C		00001100	RAM 0	BASIC	RAM 0	I/O	KERN
000D		00001101	RAM 0	BASIC	RAM 0	CHAR	KERN
000E		00001110	RAM 0	RAM 0	RAM 0	I/O	KERN
000F		00001111	RAM 0	RAM 0	RAM 0	CHAR	KERN
0010		00010000	RAM 0	BASIC	BASIC	I/O	INT
0011		00010001	RAM 0	BASIC	BASIC	RAM 0	INT
0012		00010010	RAM 0	RAM 0	BASIC	I/O	INT
0013		00010011	RAM 0	RAM 0	BASIC	RAM 0	INT
0014		00010100	RAM 0	BASIC	INT	I/O	INT
0015		00010101	RAM 0	BASIC	INT	RAM 0	INT
0016		00010110	RAM 0	RAM 0	INT	I/O	INT
0017		00010111	RAM 0	RAM 0	INT	RAM 0	INT
0018		00011000	RAM 0	BASIC	EXT	I/O	INT
0019		00011001	RAM 0	BASIC	EXT	RAM 0	INT
001A		00011010	RAM 0	RAM 0	EXT	I/O	INT
001B		00011011	RAM 0	RAM 0	EXT	RAM 0	INT
001C		00011100	RAM 0	BASIC	RAM 0	I/O	INT
001D		00011101	RAM 0	BASIC	RAM 0	RAM 0	INT
001E		00011110	RAM 0	RAM 0	RAM 0	I/O	INT
001F		00011111	RAM 0	RAM 0	RAM 0	RAM 0	INT
0020		00100000	RAM 0	BASIC	BASIC	I/O	EXT
0021		00100001	RAM 0	BASIC	BASIC	RAM 0	EXT
0022		00100010	RAM 0	RAM 0	BASIC	I/O	EXT
0023		00100011	RAM 0	RAM 0	BASIC	RAM 0	EXT
0024		00100100	RAM 0	BASIC	INT	I/O	EXT
0025		00100101	RAM 0	BASIC	INT	RAM 0	EXT
0026		00100110	RAM 0	RAM 0	INT	I/O	EXT
0027		00100111	RAM 0	RAM 0	INT	RAM 0	EXT

VALUE	IN MMU-CRL	0400	4000	8000	D000	C000
HEX	BINARY	3FFF	7FFF	BFFF	DEFF	FFFF
0028	00101000	RAM 0	BASIC	EXT	I/O	EXT
0029	00101001	RAM 0	BASIC	EXT	RAM 0	EXT
002A	00101010	RAM 0	RAM 0	EXT	I/O	EXT
002B	00101011	RAM 0	RAM 0	EXT	RAM 0	EXT
002C	00101100	RAM 0	BASIC	RAM 0	I/O	EXT
002D	00101101	RAM 0	BASIC	RAM 0	RAM 0	EXT
002E	00101110	RAM 0	RAM 0	RAM 0	I/O	EXT
002F	00101111	RAM 0	RAM 0	RAM 0	RAM 0	EXT
0030	00110000	RAM 0	BASIC	BASIC	I/O	RAM 0
0031	00110001	RAM 0	BASIC	BASIC	RAM 0	RAM 0
0032	00110010	RAM 0	RAM 0	BASIC	I/O	RAM 0
0033	00110011	RAM 0	RAM 0	BASIC	RAM 0	RAM 0
0034	00110100	RAM 0	BASIC	INT	I/O	RAM 0
0035	00110101	RAM 0	BASIC	INT	RAM 0	RAM 0
0036	00110110	RAM 0	RAM 0	INT	I/O	RAM 0
0037	00110111	RAM 0	RAM 0	INT	RAM 0	RAM 0
0038	00111000	RAM 0	BASIC	EXT	I/O	RAM 0
0039	00111001	RAM 0	BASIC	EXT	RAM 0	RAM 0
003A	00111010	RAM 0	RAM 0	EXT	I/O	RAM 0
003B	00111011	RAM 0	RAM 0	EXT	RAM 0	RAM 0
003C	00111100	RAM 0	BASIC	RAM 0	I/O	RAM 0
003D	00111101	RAM 0	BASIC	RAM 0	RAM 0	RAM 0
003E	00111110	RAM 0	RAM 0	RAM 0	I/O	RAM 0
003F	00111111	RAM 0	RAM 0	RAM 0	RAM 0	RAM 0
0040	01000000	RAM 1	BASIC	BASIC	I/O	KERN
0041	01000001	RAM 1	BASIC	BASIC	CHAR	KERN
0042	01000010	RAM 1	RAM 1	BASIC	I/O	KERN
0043	01000011	RAM 1	RAM 1	BASIC	CHAR	KERN
0044	01000100	RAM 1	BASIC	INT	I/O	KERN
0045	01000101	RAM 1	BASIC	INT	CHAR	KERN
0046	01000110	RAM 1	RAM 1	INT	I/O	KERN
0047	01000111	RAM 1	RAM 1	INT	CHAR	KERN
0048	01001000	RAM 1	BASIC	EXT	I/O	KERN
0049	01001001	RAM 1	BASIC	EXT	CHAR	KERN
004A	01001010	RAM 1	RAM 1	EXT	I/O	KERN
004B	01001011	RAM 1	RAM 1	EXT	CHAR	KERN
004C	01001100	RAM 1	BASIC	RAM 1	I/O	KERN
004D	01001101	RAM 1	BASIC	RAM 1	CHAR	KERN
004E	01001110	RAM 1	RAM 1	RAM 1	I/O	KERN
004F	01001111	RAM 1	RAM 1	RAM 1	CHAR	KERN
0050	01010000	RAM 1	BASIC	BASIC	I/O	INT
0051	01010001	RAM 1	BASIC	BASIC	RAM 1	INT
0052	01010010	RAM 1	RAM 1	BASIC	I/O	INT

VALUE	IN MMU-CRL	0400	4000	8000	D000	C000
HEX	BINARY	3FFF	7FFF	BFFF	DFFF	FFFF
0053	01010011	RAM 1	RAM 1	BASIC	RAM 1	INT
0054	01010100	RAM 1	BASIC	INT	I/O	INT
0055	01010101	RAM 1	BASIC	INT	RAM 1	INT
0056	01010110	RAM 1	RAM 1	INT	I/O	INT
0057	01010111	RAM 1	RAM 1	INT	RAM 1	INT
0058	01011000	RAM 1	BASIC	EXT	I/O	INT
0059	01011001	RAM 1	BASIC	EXT	RAM 1	INT
005A	01011010	RAM 1	RAM 1	EXT	I/O	INT
005B	01011011	RAM 1	RAM 1	EXT	RAM 1	INT
005C	01011100	RAM 1	BASIC	RAM 1	I/O	INT
005D	01011101	RAM 1	BASIC	RAM 1	RAM 1	INT
005E	01011110	RAM 1	RAM 1	RAM 1	I/O	INT
005F	01011111	RAM 1	RAM 1	RAM 1	RAM 1	INT
0060	01100000	RAM 1	BASIC	BASIC	I/O	EXT
0061	01100001	RAM 1	BASIC	BASIC	RAM 1	EXT
0062	01100010	RAM 1	RAM 1	BASIC	I/O	EXT
0063	01100011	RAM 1	RAM 1	BASIC	RAM 1	EXT
0064	01100100	RAM 1	BASIC	INT	I/O	EXT
0065	01100101	RAM 1	BASIC	INT	RAM 1	EXT
0066	01100110	RAM 1	RAM 1	INT	I/O	EXT
0067	01100111	RAM 1	RAM 1	INT	RAM 1	EXT
0068	01101000	RAM 1	BASIC	EXT	I/O	EXT
0069	01101001	RAM 1	BASIC	EXT	RAM 1	EXT
006A	01101010	RAM 1	RAM 1	EXT	I/O	EXT
006B	01101011	RAM 1	RAM 1	EXT	RAM 1	EXT
006C	01101100	RAM 1	BASIC	RAM 1	I/O	EXT
006D	01101101	RAM 1	BASIC	RAM 1	RAM 1	EXT
006E	01101110	RAM 1	RAM 1	RAM 1	I/O	EXT
006F	01101111	RAM 1	RAM 1	RAM 1	RAM 1	EXT
0070	01110000	RAM 1	BASIC	BASIC	I/O	RAM 1
0071	01110001	RAM 1	BASIC	BASIC	RAM 1	RAM 1
0072	01110010	RAM 1	RAM 1	BASIC	I/O	RAM 1
0073	01110011	RAM 1	RAM 1	BASIC	RAM 1	RAM 1
0074	01110100	RAM 1	BASIC	INT	I/O	RAM 1
0075	01110101	RAM 1	BASIC	INT	RAM 1	RAM 1
0076	01110110	RAM 1	RAM 1	INT	I/O	RAM 1
0077	01110111	RAM 1	RAM 1	INT	RAM 1	RAM 1
0078	01111000	RAM 1	BASIC	EXT	I/O	RAM 1
0079	01111001	RAM 1	BASIC	EXT	RAM 1	RAM 1
007A	01111010	RAM 1	RAM 1	EXT	I/O	RAM 1
007B	01111011	RAM 1	RAM 1	EXT	RAM 1	RAM 1
007C	01111100	RAM 1	BASIC	RAM 1	I/O	RAM 1
007D	01111101	RAM 1	BASIC	RAM 1	RAM 1	RAM 1

VALUE	IN MMU-CRL	0400	4000	8000	D000	C000
HEX	BINARY	3FFF	7FFF	BFFF	DFFF	FFFF
007E	01111110	RAM 1	RAM 1	RAM 1	I/O	RAM 1
007F	01111111	RAM 1	RAM 1	RAM 1	RAM 1	RAM 1
0080	10000000	RAM 2	BASIC	BASIC	I/O	KERN
0081	10000001	RAM 2	BASIC	BASIC	CHAR	KERN
0082	10000010	RAM 2	RAM 2	BASIC	I/O	KERN
0083	10000011	RAM 2	RAM 2	BASIC	CHAR	KERN
0084	10000100	RAM 2	BASIC	INT	I/O	KERN
0085	10000101	RAM 2	BASIC	INT	CHAR	KERN
0086	10000110	RAM 2	RAM 2	INT	I/O	KERN
0087	10000111	RAM 2	RAM 2	INT	CHAR	KERN
0088	10001000	RAM 2	BASIC	EXT	I/O	KERN
0089	10001001	RAM 2	BASIC	EXT	CHAR	KERN
008A	10001010	RAM 2	RAM 2	EXT	I/O	KERN
008B	10001011	RAM 2	RAM 2	EXT	CHAR	KERN
008C	10001100	RAM 2	BASIC	RAM 2	I/O	KERN
008D	10001101	RAM 2	BASIC	RAM 2	CHAR	KERN
008E	10001110	RAM 2	RAM 2	RAM 2	I/O	KERN
008F	10001111	RAM 2	RAM 2	RAM 2	CHAR	KERN
0090	10010000	RAM 2	BASIC	BASIC	I/O	INT
0091	10010001	RAM 2	BASIC	BASIC	RAM 2	INT
0092	10010010	RAM 2	RAM 2	BASIC	I/O	INT
0093	10010011	RAM 2	RAM 2	BASIC	RAM 2	INT
0094	10010100	RAM 2	BASIC	INT	I/O	INT
0095	10010101	RAM 2	BASIC	INT	RAM 2	INT
0096	10010110	RAM 2	RAM 2	INT	I/O	INT
0097	10010111	RAM 2	RAM 2	INT	RAM 2	INT
0098	10011000	RAM 2	BASIC	EXT	I/O	INT
0099	10011001	RAM 2	BASIC	EXT	RAM 2	INT
009A	10011010	RAM 2	RAM 2	EXT	I/O	INT
009B	10011011	RAM 2	RAM 2	EXT	RAM 2	INT
009C	10011100	RAM 2	BASIC	RAM 2	I/O	INT
009D	10011101	RAM 2	BASIC	RAM 2	RAM 2	INT
009E	10011110	RAM 2	RAM 2	RAM 2	I/O	INT
009F	10011111	RAM 2	RAM 2	RAM 2	RAM 2	INT
00A0	10100000	RAM 2	BASIC	BASIC	I/O	EXT
00A1	10100001	RAM 2	BASIC	BASIC	RAM 2	EXT
00A2	10100010	RAM 2	RAM 2	BASIC	I/O	EXT
00A3	10100011	RAM 2	RAM 2	BASIC	RAM 2	EXT
00A4	10100100	RAM 2	BASIC	INT	I/O	EXT
00A5	10100101	RAM 2	BASIC	INT	RAM 2	EXT
00A6	10100110	RAM 2	RAM 2	INT	I/O	EXT
00A7	10100111	RAM 2	RAM 2	INT	RAM 2	EXT
00A8	10101000	RAM 2	BASIC	EXT	I/O	EXT

VALUE	IN MMU-CRL	0400	4000	8000	D000	C000
HEX	BINARY	3FFF	7FFF	BFFF	DFFF	FFFF
00A9	10101001	RAM 2	BASIC	EXT	RAM 2	EXT
00AA	10101010	RAM 2	RAM 2	EXT	I/O	EXT
00AB	10101011	RAM 2	RAM 2	EXT	RAM 2	EXT
00AC	10101100	RAM 2	BASIC	RAM 2	I/O	EXT
00AD	10101101	RAM 2	BASIC	RAM 2	RAM 2	EXT
00AE	10101110	RAM 2	RAM 2	RAM 2	I/O	EXT
00AF	10101111	RAM 2	RAM 2	RAM 2	RAM 2	EXT
00B0	10110000	RAM 2	BASIC	BASIC	I/O	RAM 2
00B1	10110001	RAM 2	BASIC	BASIC	RAM 2	RAM 2
00B2	10110010	RAM 2	RAM 2	BASIC	I/O	RAM 2
00B3	10110011	RAM 2	RAM 2	BASIC	RAM 2	RAM 2
00B4	10110100	RAM 2	BASIC	INT	I/O	RAM 2
00B5	10110101	RAM 2	BASIC	INT	RAM 2	RAM 2
00B6	10110110	RAM 2	RAM 2	INT	I/O	RAM 2
00B7	10110111	RAM 2	RAM 2	INT	RAM 2	RAM 2
00B8	10111000	RAM 2	BASIC	EXT	I/O	RAM 2
00B9	10111001	RAM 2	BASIC	EXT	RAM 2	RAM 2
00BA	10111010	RAM 2	RAM 2	EXT	I/O	RAM 2
00BB	10111011	RAM 2	RAM 2	EXT	RAM 2	RAM 2
00BC	10111100	RAM 2	BASIC	RAM 2	I/O	RAM 2
00BD	10111101	RAM 2	BASIC	RAM 2	RAM 2	RAM 2
00BE	10111110	RAM 2	RAM 2	RAM 2	I/O	RAM 2
00BF	10111111	RAM 2	RAM 2	RAM 2	RAM 2	RAM 2
00C0	11000000	RAM 3	BASIC	BASIC	I/O	KERN
00C1	11000001	RAM 3	BASIC	BASIC	CHAR	KERN
00C2	11000010	RAM 3	RAM 3	BASIC	I/O	KERN
00C3	11000011	RAM 3	RAM 3	BASIC	CHAR	KERN
00C4	11000100	RAM 3	BASIC	INT	I/O	KERN
00C5	11000101	RAM 3	BASIC	INT	CHAR	KERN
00C6	11000110	RAM 3	RAM 3	INT	I/O	KERN
00C7	11000111	RAM 3	RAM 3	INT	CHAR	KERN
00C8	11001000	RAM 3	BASIC	EXT	I/O	KERN
00C9	11001001	RAM 3	BASIC	EXT	CHAR	KERN
00CA	11001010	RAM 3	RAM 3	EXT	I/O	KERN
00CB	11001011	RAM 3	RAM 3	EXT	CHAR	KERN
00CC	11001100	RAM 3	BASIC	RAM 3	I/O	KERN
00CD	11001101	RAM 3	BASIC	RAM 3	CHAR	KERN
00CE	11001110	RAM 3	RAM 3	RAM 3	I/O	KERN
00CF	11001111	RAM 3	RAM 3	RAM 3	CHAR	KERN
00D0	11010000	RAM 3	BASIC	BASIC	I/O	INT
00D1	11010001	RAM 3	BASIC	BASIC	RAM 3	INT
00D2	11010010	RAM 3	RAM 3	BASIC	I/O	INT
00D3	11010011	RAM 3	RAM 3	BASIC	RAM 3	INT

VALUE	IN MMU-CRL	0400	4000	8000	D000	C000
HEX	BINARY	3FFF	7FFF	BFFF	DEFF	FFFF
00D4	11010100	RAM 3	BASIC	INT	I/O	INT
00D5	11010101	RAM 3	BASIC	INT	RAM 3	INT
00D6	11010110	RAM 3	RAM 3	INT	I/O	INT
00D7	11010111	RAM 3	RAM 3	INT	RAM 3	INT
00D8	11011000	RAM 3	BASIC	EXT	I/O	INT
00D9	11011001	RAM 3	BASIC	EXT	RAM 3	INT
00DA	11011010	RAM 3	RAM 3	EXT	I/O	INT
00DB	11011011	RAM 3	RAM 3	EXT	RAM 3	INT
00DC	11011100	RAM 3	BASIC	RAM 3	I/O	INT
00DD	11011101	RAM 3	BASIC	RAM 3	RAM 3	INT
00DE	11011110	RAM 3	RAM 3	RAM 3	I/O	INT
00DF	11011111	RAM 3	RAM 3	RAM 3	RAM 3	INT
00E0	11100000	RAM 3	BASIC	BASIC	I/O	EXT
00E1	11100001	RAM 3	BASIC	BASIC	RAM 3	EXT
00E2	11100010	RAM 3	RAM 3	BASIC	I/O	EXT
00E3	11100011	RAM 3	RAM 3	BASIC	RAM 3	EXT
00E4	11100100	RAM 3	BASIC	INT	I/O	EXT
00E5	11100101	RAM 3	BASIC	INT	RAM 3	EXT
00E6	11100110	RAM 3	RAM 3	INT	I/O	EXT
00E7	11100111	RAM 3	RAM 3	INT	RAM 3	EXT
00E8	11101000	RAM 3	BASIC	EXT	I/O	EXT
00E9	11101001	RAM 3	BASIC	EXT	RAM 3	EXT
00EA	11101010	RAM 3	RAM 3	EXT	I/O	EXT
00EB	11101011	RAM 3	RAM 3	EXT	RAM 3	EXT
00EC	11101100	RAM 3	BASIC	RAM 3	I/O	EXT
00ED	11101101	RAM 3	BASIC	RAM 3	RAM 3	EXT
00EE	11101110	RAM 3	RAM 3	RAM 3	I/O	EXT
00EF	11101111	RAM 3	RAM 3	RAM 3	RAM 3	EXT
00F0	11110000	RAM 3	BASIC	BASIC	I/O	RAM 3
00F1	11110001	RAM 3	BASIC	BASIC	RAM 3	RAM 3
00F2	11110010	RAM 3	RAM 3	BASIC	I/O	RAM 3
00F3	11110011	RAM 3	RAM 3	BASIC	RAM 3	RAM 3
00F4	11110100	RAM 3	BASIC	INT	I/O	RAM 3
00F5	11110101	RAM 3	BASIC	INT	RAM 3	RAM 3
00F6	11110110	RAM 3	RAM 3	INT	I/O	RAM 3
00F7	11110111	RAM 3	RAM 3	INT	RAM 3	RAM 3
00F8	11111000	RAM 3	BASIC	EXT	I/O	RAM 3
00F9	11111001	RAM 3	BASIC	EXT	RAM 3	RAM 3
00FA	11111010	RAM 3	RAM 3	EXT	I/O	RAM 3
00FB	11111011	RAM 3	RAM 3	EXT	RAM 3	RAM 3
00FC	11111100	RAM 3	BASIC	RAM 3	I/O	RAM 3
00FD	11111101	RAM 3	BASIC	RAM 3	RAM 3	RAM 3
00FE	11111110	RAM 3	RAM 3	RAM 3	I/O	RAM 3
00FF	11111111	RAM 3	RAM 3	RAM 3	RAM 3	RAM 3



BASIC = BASIC rom.  
EXT = EXTERNAL rom (plug in CARTRIDGE).  
INT = INTERNAL rom (This is a spare socket inside the computer which is not used at this time.)  
KERN = KERNal rom.  
RAM = RAM bank x (NOTE: RAM 3 and RAM 4 are currently UNUSED and not available on the current model of the C-128.)



**Index**



ABS	457
AND	133-134
APPEND	542-543
Array	
descriptor	27-29
floating point	30
function	32-33
integer	31
string	31-32
ASC	425-426
ATN	498-499
AUTO	203
BACKUP	553-554
BANK	71-80,281-282
BASIC IRQ routine	584-588
BASIC jump table	99-100
BASIC statement execute	123-124
Bitmap	71-80
BLOAD	547-548
BOOT	320-321
BOX	79-80,246-250
BSAVE	546
BUMP	403-404
CATALOG	539-541
CHAR	262-267
Character memory	73-76
CHKIN	476
CHKOUT	476
CHKSPRN	286
CHR\$	419
CHRGET	59-68,95
CHRGOT	59-68,95
CIRCLE	258-261
CLOSE	481
CLR	160-161
CMD	178-190
COLLECT	551-552
COLLISION	312-313
COLOR	271-274

<b>Color memory</b>	76-77
CONCAT	552-553
CONT	208-209
COPY	552
COS	496
DATA	163
DCLEAR	551
DCLOSE	543-544
DEC	385-387
DEF FN	414-418
DELETE	227-229
DIM	197-198
DIRECTORY	539-541
<b>DIRECT mode</b>	3
<b>Direct Memory Access (DMA)</b>	43-52
<b>Division (/) function</b>	451-453
DLOAD	545-546
DO	8-9,234
DOPEN	541-542
<b>DOS initialization data</b>	554-564
DRAW	79,261-262
DSAVE	544
<b>Dual token commands and functions</b>	110-111
DVERIFY	545
ELSE BEGIN	168-169
END	129-130
ERR\$	388-389
<b>Execute Direct Memory Access (EDMA)</b>	49-52
EXP	470-473
FAST	339-340
FETCH	590
FILL	276-277
FN	415-418
FOR	22,225-227
FRE	382-384
FROMTO	229
<b>Functions</b>	128-131

GET	184-186
GET#	186
GETKEY	185
GOSUB	6-7,205-206
GOTO	206-208
GSHAPE	255-258
GRAPHIC	279-281
Graphic CLR	278-279
Graphics	71-80
HEADER	548-549
HELP	204-205
HEX\$	390-391
High resolution graphics	71-80
IF	165-168
INPUT	186-188
INPUT#	187
INSTR	513-517
INT	459-460
Interrupt ReQuest (IRQ)	3-4,82,83,584-589
JOY	394-396
Jump tables	592-594
KEY	240-243
KEYOFF	55,57-64
KEYON	55,57-64
Least Significant Bit (LSB)	13,23
LEFT\$	419-421
LEN	424-425
LESS THAN (<) function	134-135
LET	170
Line link address	13
LIST	154-155
LOAD	478-480
LOCATE	268
LOG	443-444
LOOP	8-9,237-240
LOOP/WHILE	234-235

Memory expansion	40-53
MID\$	422-423
MONITOR	12,23
Most Significant Bit (MSB)	13,23
MOVSPR	287-290
NEW	158-159
NEXT	195-197
NORMAL	417-418
NOT	347
ON	169
OPEN	480-481
OR	133
PAINT	76-77,243-246
PEEK	387
PEN	398-399
PLAY	87-89,292-312
POINTER	400
POKE	388
POS	412
POT	396-397
PRINT	179-182
PRINT#	178
PRINT USING	161,500-513
Pseudo stack	5-11
PUDEF	230
RAM Expansion Module (REM)	41-42
RAM Expansion Controller (REC)	41
Random Access Memory (RAM)	41-53
RCLR	392-394
RDOT	517-521
READ	188-191
Read Only Memory (ROM)	54-56
READY	136
RECORD	550-551
REM	164
RENAME	553



RENUMBER	214-225
<b>Reserved words</b>	104-116,603-605
RESTORE	212-213
RESUME	231-234
RESUME LINE NUMBER	233-234
RESUME NEXT	232
RETURN	6-7,162-163
RGR	391-392
RIGHT\$	421-422
RND	407-408
RREG	200-201
RSPCOLOR	402-403
RSPPOS	404-405
RSPRITE	401-402
RUN	210-212
RWINDOW	406-407
SAVE	477
SAVE with REPLACE	564-566
SCALE	268-271
SCNCLR	77,274-276
SCRATCH	549-550
SGN	456
<b>SID</b>	87-89,292-312
SIN	496-497
SLEEP	282-283
SLOW	340
SOUND	315-318
SPC	182
SPRCOLOR	313-314
SPRDEF	322-336
SPRITE	284-286
SPRSAV	336-339
SQR	470
SSHAPE	251-255
<b>Stack</b>	5-11
<b>Stack pointer</b>	5
STASH	589-590
STOP	128-129
STR\$	418-419

SWAP	590-591
SYS	198-199
System "crashes"	16
TAB	182
TAN	497-498
Tokens	102-119
Top Of Stack (TOS)	6-11
TROFF	199
TRON	199
TRAP	127,230-231
unNEWing BASIC programs	15-16, 158-159
UNTIL	235-236
VAL	384-385
Variable descriptor	20,23,24-25
Variable Dump program	36-39
Variable pointers	33-35
Variables	
complex— <i>see</i> Arrays	
floating point	17-18,20
function	17-18
integer	17-18,23,170-171
reserved	17-18
simple	19
string	17-18,24,171-178
VERIFY	477-478
VIC chip	67-69,71-72
Video bank	68-72,74
VOLUME	306-307
WAIT	283-284
WIDTH	314
WINDOW	318-320
XOR	405-406

## Optional Diskette



For your convenience, the program listings contained in this book are available on an 1541 formatted floppy disk. You should order the diskette if you want to use the programs, but don't want to type them in from the listings in the book.

All programs on the diskette have been fully tested. You can change the programs for your particular needs. The diskette is available for \$14.95 plus \$2.00 (\$5.00 foreign) for postage and handling.

When ordering, please give your name and shipping address. Enclose a check, money order or credit card information. Mail your order to:

Abacus Software  
P.O. Box 7219  
Grand Rapids, MI 49510

Or for *fast* service, call (616) 241-5510.

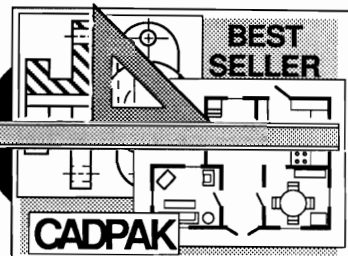


# '128™ and C-64™ PROVEN PERFORMANCE



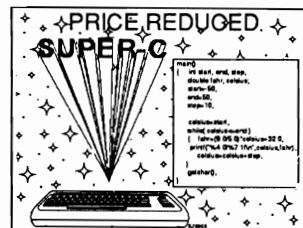
The complete compiler and development package. Speed up your programs 5x to 35x. Many options: flexible memory management; choice of compiling to machine code, compact p-code or both. '128 version: 40 or 80 column monitor output and FAST-mode operation. '128 Compiler's extensive 80-page programmer's guide covers compiler directives and options, two levels of optimization, memory usage, I/O handling, 80 column hi-res graphics, faster, higher precision math functions, speed and space saving tips, more. A great package that no software library should be without.

128 Compiler \$59.95  
64 Compiler \$39.95



Remarkably easy-to-use interactive drawing package for accurate graphic designs. New dimensioning features to create exact scaled output to all major dot-matrix printers. Enhanced version allows you to input via keyboard or high quality lightpen. Two graphic screens for COPYING from one to the other. DRAW, LINE, BOX, CIRCLE, ARC, ELLIPSE available. FILL objects with preselected PAT-TERNS; add TEXT; SAVE and RECALL designs to/from disk. Define your own library of symbols/objects with the easy-to-use OBJECT MANAGEMENT SYSTEM—store up to 104 separate objects.

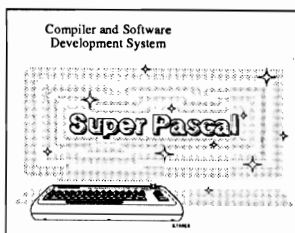
C-128 \$59.95  
C-64 \$39.95



For school or software development. Learn C on your Commodore with our in-depth tutorial. Compile C programs into fast machine language. C-128 version has added features: Unix™-like operating system; 60K RAM disk for fast editing and compiling Linker combines up to 10 modules; Combine M/L and C using CALL; 51K available for object code;

Fast loading (8 sec. 1571, 18 sec. 1541); Two standard I/O libraries plus two additional libraries—math functions (sin, cos, sqrt, etc.) & 20+ graphic commands (line, fill, dot, etc.).

C-128 \$59.95  
C-64 \$59.95



Not just a compiler, but a complete system for developing applications in Pascal with graphics and sound features. Extensive editor with search, replace, auto, renumber, etc. Standard J & W compiler that generates fast machine code. If you want to learn Pascal or to develop software using the best tools available—SUPER Pascal is your first choice.

C-128 \$59.95  
C-64 \$59.95

## OTHER TITLES AVAILABLE:

### COBOL Compiler

Now you can learn COBOL, the most widely used commercial programming language, and use COBOL on your 64. COBOL is easy to learn because its easy to read. COBOL Compiler package comes complete with Editor, Compiler, Interpreter and Symbolic Debugger.

C-64 \$39.95

### Personal Portfolio Manager

Complete portfolio management system for the individual or professional investor. Easily manage your portfolios, obtain up-to-the-minute quotes and news, and perform selected analysis. Enter quotes manually or automatically through Warner Computer Systems.

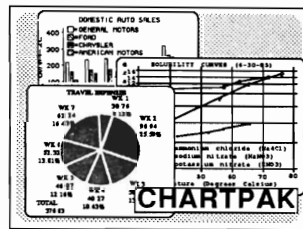
C-64 \$39.95

### Xper

XPER is the first "expert system" for the C-128 and C-64. While ordinary data base systems are good for reproducing facts, XPER can derive knowledge from a mountain of facts and help you make expert decisions. Large capacity. Complete with editing and reporting.

C-64 \$59.95

C-128 and C-64 are trademarks of Commodore Business Machines Inc. Unix is a trademark of Bell Laboratories



Easily create professional high quality charts and graphs without programming. You can immediately change the scaling, labeling, axis, bar filling, etc. to suit your needs. Accepts data from CalcResult and MultiPlan. C-128 version has 3X the resolution of the '64 version. Outputs to most printers.

C-128 \$39.95  
C-64 \$39.95

### PowerPlan

One of the most powerful spreadsheets with integrated graphics. Includes menu or keyword selections, online help screens, field protection, windowing, trig functions and more. PowerGraph, the graphics package, is included to create integrated graphs and charts.

C-64 \$39.95

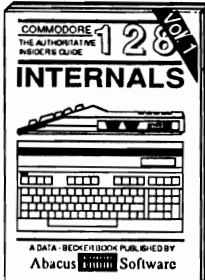
Technical Analysis System for the C-64 \$59.95  
Ada Compiler for the C-64 \$39.95  
VideoBasic Language for the C-64 \$39.95

# Abacus Software

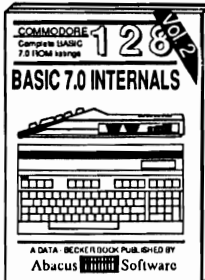
P.O. Box 7219 Dept. H8 Grand Rapids, MI 49510 - Telex 709-101 - Phone (616) 241-5510  
Call now for the name of your nearest dealer. Or to order directly by credit card, MC, AMEX or VISA call (616) 241-5510. Other software and books are available—Call and ask for your free catalog. Add \$4.00 for shipping per order. Foreign orders add \$12.00 per item. Dealer inquiries welcome—1400+ nationwide.



# C-128™ AUTHORITY™ and C-64™ BOOKS



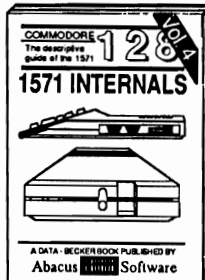
Detailed guide presents the 128's operating system, explains graphic chips, Memory Management Unit, 80 column graphics and commented ROM listings. **500pp \$19.95**



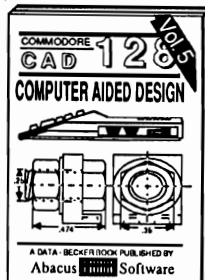
Get all the inside information on BASIC 7.0. This exhaustive handbook is complete with commented BASIC 7.0 ROM listings. Coming Summer '86. **\$19.95**



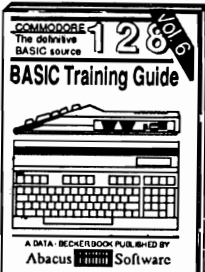
Filled with info for everyone. Covers 80 column hi-res graphics, windowing, memory layout, Kernal routines, sprites, software protection, autostarting. **300pp \$19.95**



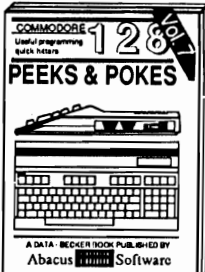
Insiders' guide for novice & advanced users. Covers sequential & relative files, & direct access commands. Describes DOS routines. Commented listings. **\$19.95**



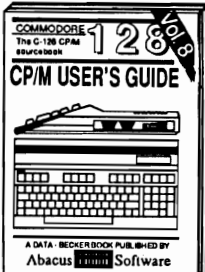
Learn fundamentals of CAD while developing your own system. Design objects on your screen to dump to a printer. Includes listings for '64 with Simon's Basic. **300pp \$19.95**



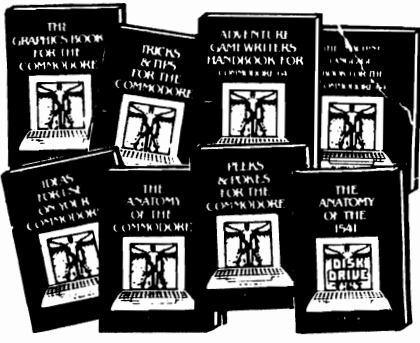
Introduction to programming, problem analysis; thorough description of all BASIC commands with hundreds of examples; monitor commands; utilities; much more. **\$16.95**



Presents dozens of programming quick-hitters. Easy and useful techniques on the operating system, stacks, zero-page, pointers, the BASIC interpreter and more. **\$16.95**



Essential guide for everyone interested in CP/M on the 128. Simple explanation of the operating system, memory usage, CP/M utility programs, submit files and more. **\$19.95**



**ANATOMY OF C-64** Insider's guide to the '64 internals. Graphics, sound, I/O, kernal, memory maps, more. Complete commented ROM listings. **300pp \$19.95**

**TRICKS & TIPS FOR C-64** Collection of easy-to-use techniques: advanced graphics, improved data input, enhanced BASIC, CP/M, more. **275pp \$19.95**

**SCIENCE/ENGINEERING ON C-64** In depth intro to computers in science. Topics: chemistry, physics, biology, astronomy, electronics, others. **350pp \$19.95**

**Adventure Gamewriters Handbook** Step-by-step guide to designing and writing your own adventure games. With automated adventure game generator. **200pp \$14.95**

**ANATOMY OF 1541 DRIVE** Best handbook on floppy drives all. Many examples and listings. Fully commented 1541 ROM listings. **500pp \$19.95**

**1541 REPAIR & MAINTENANCE** Handbook describes the disk drive hardware. Includes schematics and techniques to keep 1541 running. **200pp \$19.95**

**CASSETTE BOOK C-64/VIC-20** Comprehensive guide; many sample programs. High speed operating system fast file loading and saving. **225pp \$14.95**

**PEEKs & POKEs FOR THE C-64** Includes in-depth explanations of PEEK, POKE, USR, and other BASIC commands. Learn the "inside" tricks to get the most out of your '64. **200pp \$14.95**

**MACHINE LANGUAGE C-64** Learn 6510 code write fast programs. Many samples and listings for complete assembler, monitor, & simulator. **200pp \$14.95**

**ADVANCED MACHINE LANGUAGE** Not covered elsewhere: - video controller, interrupts, timers, clocks, I/O, real time, extended BASIC, more. **210pp \$14.95**

**IDEAS FOR USE ON C-64** Themes: auto expenses, calculator, recipe file, stock lists, diet planner, window advertising, others. Includes listings. **200pp \$12.95**

**Optional Diskettes for books** For your convenience, the programs contained in each of our books are available on diskette to save you time entering them from your keyboard. Specify name of book when ordering. **\$14.95 each**

**GRAPHICS BOOK C-64** - best reference covers basic and advanced graphics. Sprites, animation, hires, Multicolor, lighten, 3D-graphics, IRO, CAD, projections, curves, more. **350pp \$19.95**

**PRINTER BOOK C-64/VIC-20** Understand Commodore, Epson-compatible printers and 1520 plotter. Packed: utilities; graphics dump; 3D-plot; commented MP5801 ROM listings, more. **330pp \$19.95**

**COMPILER BOOK C-64/C-128** All you need to know about compilers: how they work; designing and writing your own; generating machine code. With working example compiler. **300pp \$19.95**

C-128 and C-64 are trademarks of Commodore Business Machines Inc.

# Abacus Software

P.O. Box 7219 Dept. H8 Grand Rapids, MI 49510 - Telex 709-101 - Phone (616) 241-5510

Optional diskettes available for all book titles - \$14.95 each. Other books & software also available. Call for the name of your nearest dealer. Or order directly from ABACUS using your MC, Visa or Amex card. Add \$4.00 per order for shipping. Foreign orders add \$10.00 per book. Call now or write for your free catalog. Dealer inquiries welcome--over 1400 dealers nationwide.











COMMODORE®

128™

BASIC 7.0

INTERNALS

**BASIC 7.0 Internals** is the most comprehensive book about the BASIC interpreter built into the C-128. It begins with an extensive explanation of how the BASIC interpreter works. The main section of **BASIC 7.0 Internals** contains the complete, fully-commented BASIC 7.0 ROM listings. It thoroughly explains how each routine functions and what registers are used. With a thorough knowledge of BASIC 7.0, you can create BASIC extensions or use BASIC 7.0 routines to your best advantage. Some of the subjects include:

- how variables are used (simple, floating, integer, string, arrays)
- ROM/RAM memory expansion on the C-128 using the DMA controller
- how to "wedge" into BASIC
- managing graphic memory
- moving the bit-map screen
- over 450 pages of fully-commented BASIC 7.0 ROM listings

ISBN 0-916439-71-2

Commodore 128™ is a trademark of Commodore Electronics, Ltd.

you can count on  
**Abacus**

