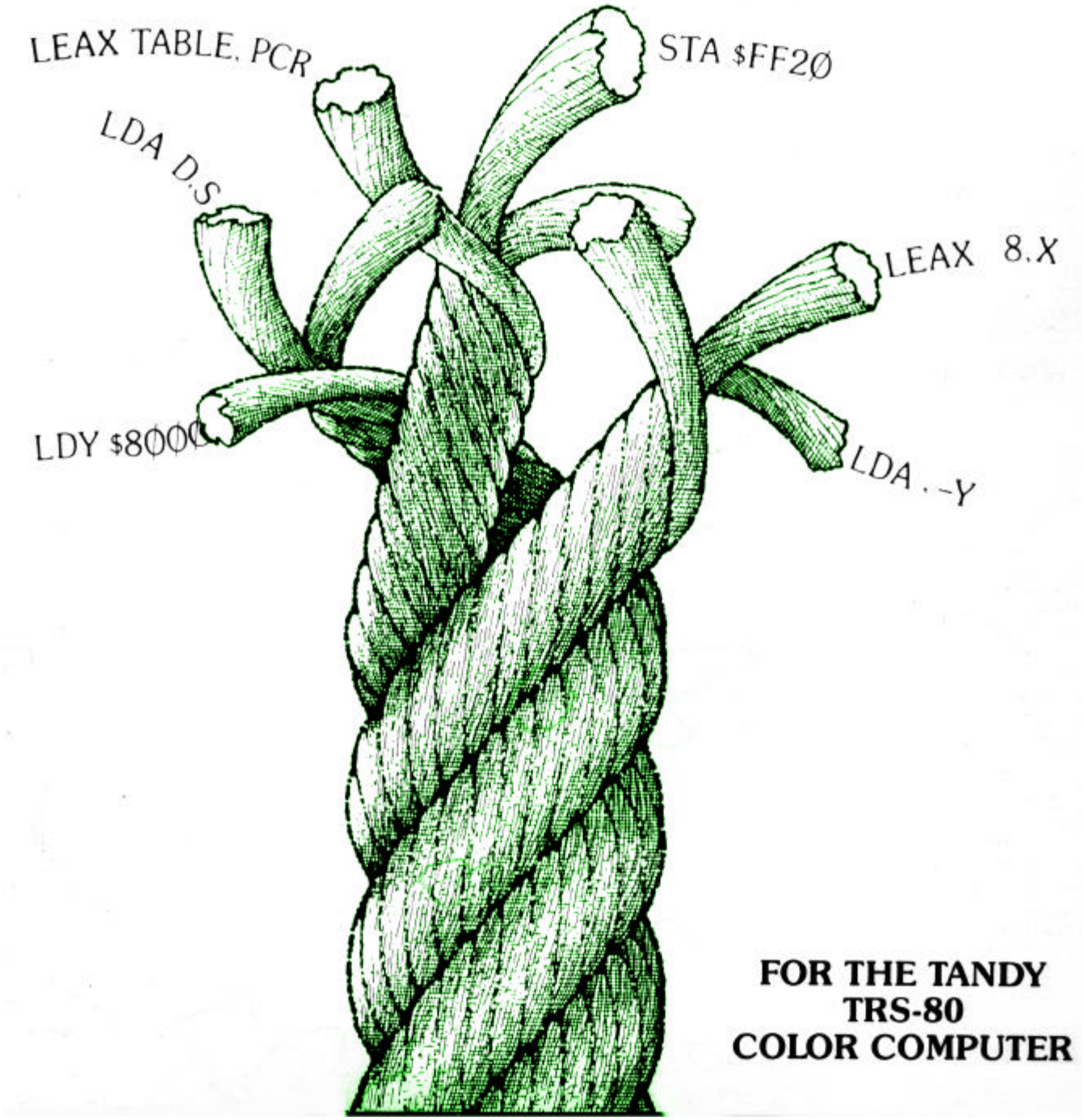
COLOR BASIC UNRAVELLED II



FOR THE TANDY TRS-80 COLOR COMPUTER

TABLE OF CONTENTS

APPENDICES

FOREWORD

Due to the many requests for the Unravelled Series produced by Spectral Associates, and the fact that these books are rare and no longer in production, I have taken it upon myself to reproduce them in electronic .PDF (Adobe Acrobat®) format.

I have re-disassembled the ROMs listed in this book, and added all the comments from the Original Extended Basic Unravelled Book. Some changes were made to make the book a little easier to read.

1. The comments have been cleaned up some. In cases where a comments continued onto the next line, a * is placed in the Labels column, as well as a * at the beginning of each line of the comment. In cases where the previous comment used this format, a = was used. This was done in the original, but not all comments stuck to this format.

2. I have renumbered all the linenumbers. Each Appendix starts at Line 0001.

3. Some spell checking, and context checking was done to verify accuracy.

4. I used the Letter Gothic MT Bold Font. This allows for display of Slashed Zeros. I thought it important to be able to distinguish between 0 and O.

5. All the Hex code now shows the Opcodes.

There were other minor changes that were made to make viewing a little better. If any discrepancies arise, please let me know so that I may correct the errors. I can be contacted at: mailto:wzydhek@internetcds.com

About Me

My name is Walter K. Zydhek. I've been a Computer Hobbyist since 1984 when I received my 1st Tandy Color Computer 2 for Christmas. It had 32K of ram, Cassette, and one Cartridge. I quickly learned to program in Basic and then moved into Assembly.

Over the next few years, I saved to purchase the Multi-Pak Interface, Disk Drives, Modem, OS-9, and various Odds and Ends.

I moved to Tampa Florida and in the move, My CoCo was damaged. I then replaced it with the CoCo 3. WOW what a difference. I added the 512K Ram Upgrade, A CM-8 color monitor, and joined a CoCo Club. Can anyone from Tampa, Florida tell me the name?

I had a couple of close friends that helped me explore the world of CoCo and by this time, I knew that my CoCo would be my friend forever. I give special thanks to Steve Cohn, who helped me get started with ADOS. Two other people who's names I can't remember were very beneficial to my mastering of the CoCo.

Shortly after getting my CoCo 3, I started BBS'ing.. Wow, a whole new world.. My knowledge just kept growing..

A few years later, I moved to Oregon, then to Phoenix, Arizona to attend school. I studied Electronics Technology at Phoenix Institute of Technology. In the second year, we studied Micro-processor Theory. For our labs, we just happen to use the Tandy Color Computer 3 (for studying 6809 Processors). I had it made. In this class I added an EPROM programmer/reader to my list of hardware. My favorite instructor, Gary Angle & I spent many hours sharing information on the CoCo. At one time, we shared a joint project to disassemble ROMs from an industrial machinery which used the 6809 Processor. Using the CoCo to read the ROMs to work with.

I even had a BBS running under OS-9 at one time. RiBBS I think it was. Very similar to QuickBBS and RemoteAccess BBS for the PC.

In 1991, I finally converted over to PC, but never forgetting my CoCo. About 5 years ago, My CoCo and all related material was stolen from me. And the CoCo world was just a memory.

In the last 2 Years, my love for the CoCo has re-kindled. I have been partially content to use a CoCo Emulator for my PC. I tried the CoCo 2 Emulator by Jeff Vavasour. This was OK, but a lot was left out. I then purchased the CoCo 3 Emulator. Much better, but would not use “Double Sided Disks”. Although it did have a Virtual Hard Drive for use in OS-9.

I then wanted to ‘better’ the CoCo Emulator, add use of PC hardware, Add Double Sided Disk functionality, and even make it Windows Native, instead of a Dos Box. Unfortunately the I could not get the source code for the CoCo 3 Emulator.

I then turned to Paul Burgin’s Dragon 2/Coco 2 Emulator. This had source code available and with a small $20.00 donation, was able to get the source code to additional portions of his program. I have tinkered with it, but came to understand that I needed more info on the CoCo. I have looked all over the net and found quite a lot of useful information, but what I really needed was the Unravelled Series.

I was able to find someone that had Extended Basic Unravelled and Disk Basic Unravelled (He sent them to me for free). And a friend of mine had Super Extended Basic Unravelled (A copy I gave him years ago). Unfortunately, the books are not in the best of shape, and the type is hard to read, and with so many people looking for the books, I decided to re-do them in Electronic format.

I ask everyone that obtains copies of this electronic document to PLEASE give freely. These books are for educational/informational use only. These books are no longer in publication and Spectral Associates no longer in business. Do not use these books for financial gain, as that would most certainly abuse the Copyright Laws that I have already bruised by re-producing them.

Other than that, enjoy the books!! I’ll add more information to them as I get it. I plan on adding more Memory Map information, as well as hardware info in the coming months.. But for now, take advantage of this fine resource.

Walter K. Zydhek

INTRODUCTION

BASIC Unravelled is a book that has specifically been written in order to provide the Color Computer user with a detailed, commented source listing of Color BASIC. Many entry points and useful routines for doing functions, which are required in machine language, have been detailed. Information contained in the book is extremely valuable and useful for anyone attempting to use BASIC integrated with machine language routines of their own. If there are some functions or facets of BASIC, which are too slow or awkward for the user, he needs to have a good idea as to what is going on in Color BASIC in order to be able to merge his routine with the routines, which are in color BASIC. There are many times when a person wants to know exactly what is going on in BASIC for a certain function such as clearing the screen or outputting a character to the screen. The information provided in BASIC Unravelled will allow the user to determine exactly what BASIC is doing under these circumstances. He will get an extremely good, in depth, basic knowledge of BASIC and be able to use that in any application he has in mind.

This book will not explain how to make the BASIC interpreter or give a detailed in depth knowledge of how a BASIC interpreter works. It assumes that the user is an experienced machine language programmer, understands 6809 assembly language inside and out, and will understand the nuances and programming terminology which is used in the comments included with each BASIC program line. Do not attempt to use BASIC Unravelled as a textbook in order to teach yourself how to write a BASIC interpreter. If you are a very good programmer you will be able to write your own BASIC interpreter following the in depth reading of the assembly listings and the comments included in this book, but it should not be taken as a text on how to write a BASIC interpreter. The book is primarily designed to explain Color BASIC so that somebody who has a fair knowledge of how an interpreter works will be able to determine exactly how Color BASIC works. BASIC Unravelled will explain major operating formats of the most useful routines in BASIC and will identify the tricks, which Microsoft has used in programming Color BASIC. If the reader has any questions concerning the hardware of the Color computer, he's referred to the FACTS book, published by Spectral Associates. This book contains detailed descriptions of the hardware of the Color Computer and how one uses software in order to enable or disable the various hardware functions of the computer.

BASIC Unravelled will deal specifically with Color BASIC version 1.2 which is the version of Color BASIC released by Radio Shock, as of October 1983. The two earlier versions, version 1.0 and version 1.1, have only minor differences in relation to version 1.2. These differences are described in detail in the appendices, and if the reader has any questions in regard to version 1.1 and version 1.0 is referred to those appendices. Extended BASIC and Disk BASIC are covered in the two final books of the BASIC Unravelled sequence published by Spectral Associates. Any questions that regard explicitly to Extended BASIC and Disk BASIC will be covered in those books.

COLOR BASIC – An Interpreter

Color BASIC is a computer program, which is written in machine language, is very complex, and is extremely difficult to understand without some kind of helpful information. The idea behind writing a program, such as BASIC, is that BASIC is very easy to understand for the beginning user. Machine language, unfortunately, is very difficult to use and takes considerable amount of practice in order to get familiar with it. Therefore, BASIC is the language, which is provided with most computers when they are sold to the general public. As the user gets more and more familiar with BASIC, more and more questions generally arise as to how BASIC functions. That is one of the main purposes of the book -- to explain to the user exactly how Color BASIC, the Interpreter, works.

It is assumed that the reader is familiar with the manner in which the Interpreter functions. He at least knows the basic overall method of how an interpreter works in that the lines must be numbered, the interpreter executes these lines one after the other, and transfers control with GOTO, GOSUB and other similar statements. BASIC is an interpretive language related to the direct commands we are executing. BASIC executes a command by taking the last line typed to it and analyzing the line working from left to right looking for keywords and expressions, which it recognizes. Every time it encounters a keyword such as PRINT (or ? which is the abbreviation for PRINT), it interprets this word into a command, which means something to BASIC. Command words are stored in memory with bit 8 set to tell BASIC that it is a command word, or keyword (token). As a program line is entered into RAM memory through the use of the enter key, BASIC takes the line number and searches through memory, until it finds the same number, or the number just greater. If it is the same line number, then the entire line in memory is deleted and a new line is inserted into memory. In the preinterpreted state all the keywords are replaced with the single character token of the keyword. This allows the interpreter to store commands in the most memory efficient form. The only data stored is the data typed in by the programmer such as strings, pointers to the variables, and the keywords. PRINT, even though it takes five characters to type, only takes one character in memory.

BASIC is called an interpreter because the actual execution of the instructions is done by analyzing the keyword that needs to be executed in the program line, then executing that keyword under the control of a series of subroutines. This is a trade-off, which results in very memory-efficient storage programs but longer execution times that would be true of a machine language program. Because Color BASIC uses tokens in memory and stores them on I/O devices whenever a program is loaded and saved, the actual coding of data on tape or in memory is not transferable to other Machines. It is generally not possible to use BASIC instructions typed in from other machines. It is not assumed that the reader is very familiar with all the weaknesses and strengths of the BASIC interpreter as opposed to a compiled language. No effort will be made to explain the differences between compilation and interpretation except to make note of the fact that many of the weaknesses of the BASIC interpreter stem from the fact that it is not compiled; that is, that the program is not converted into machine language and executed in one pass after it is converted into machine language. Each time a statement has to be interpreted with the BASIC interpreter, the interpreter must look up the functions that need to be interpreted, find out what they are, calculate any numerical results that are necessary as a result of the interpretation, print things to the screen and so forth, and then continue to the next statement. This is one of the main weaknesses of an interpreter-it is slow. Every time a statement has to be interpreted the some slow process has to take place. A perfect example of this is the determination of the value of a variable. BASIC stores its variables in

a large table directly after the BASIC program. These tables have the variable tagged by its name, that is the one or two character ASCII sequence which is defined in the program such as: AA, Al, X, Y, etc. These variables are listed one after the other in the variable table and every time the BASIC program makes reference to a variable, BASIC must start at the beginning of the variable table and search through the entire length of table until it determines where that variable is. If a program were compiled, the program would know exactly where the variable is and wouldn't have to go searching through the table in order to find it. Obviously if the variable was located near the end of a very long variable table, a substantial amount of time will be consumed every time the BASIC program makes reference to this variable. This is one reason why it’s convenient to put the variables which are the most often used in a BASIC program, at the very beginning of the program. Another example of the slowness of an interpreter is every time you make program control transfers such as GOTO or GOSUB, the program has to search through the entire length of the BASIC program in order to find where the destination line number is. If the line number happens to be just before the point where the BASIC program starts searching, the entire program will have to be searched through in order to determine where the program line is and then transfer control there. As you can see, this will waste a lot of time.

Why then, you would ask, do we use BASIC programs in the first place? The primary reason is because of the fact that BASIC is user friendly. It is simple to learn and it is simple to program. And even if it’s slow, it still provides a very powerful tool for the user. It is very easy to develop and test programs and takes only a fraction of the time involved in what it would take to develop a comparable length machine language program. The penalty, of course, is the speed and the size of the final program.

INTERPRETER MECHANICS

The Interpreter has various statements, commands, and functions, which are used in order to process, manipulate or otherwise use data. The overall goal of any computer program is the manipulation and movement of data in the memory of the computer and the transference of that data to an input/output device such as the screen, a disk file, or a cassette tape. Commands will tell the Color Computer to do something with the program. Sample commands would be LIST, RUN, STOP, and CONTINUE. Statements are used to operate on the data or program, which is in the Color Computer at the time. Examples of statements are CLEAR, DATA, DIM, END, GOSUB, INPUT, and so forth. Functions provide another manner that BASIC statements can be used to control the Color Computer. Intrinsic functions provided by BASIC are used to operate on string or numeric data and produce a result, which will be useful. Many of these functions are mathematical functions or string manipulation functions which will form substrings based upon certain subsets of the string in question. The arguments of these functions are always enclosed in parentheses. The argument of any function is the value which is being manipulated by the function and sometimes there may be more than one argument in any particular function, such as MID$, which may have three arguments. Often arguments may be left out and BASIC will supply default values. These default values can be found by looking at the routine in BASIC which controls that particular function and you can see whether or not a default value is allowed for. Sometimes a default value is not allowed, and if a value is not given BASIC will generate an error. Some examples of functions are ABS, ASC, SIN, COS, RIGHT$, LEFT$, etc.

The commands and functions of BASIC must be identified to the computer in a way that they can be understandable by the machine. The machine cannot understand PRINT, LIST, and RUN, it can only understand numbers. Therefore, there must be some way of identifying the commands and functions to the computer so that it knows that these are commands and functions. The method that most BASIC interpreters use in order to identify commands and functions is to identify them with a number from 128 to 255. The user will notice that these numbers are the equivalent of an 8-bit byte with bit 7 set. This is a very convenient way of identifying and abbreviating the commands and functions, because the numbers from 0 to 127 are the normal ASCII symbols used by BASIC. The numbers from 128 to 255 represent graphic symbols, which are rarely used in ASCII strings. These abbreviations for the BASIC commands and functions are called tokens. You will find in the BASIC listing a table of tokens and the respective addresses where command of BASIC is transferred when a certain token is encountered in an interpreted line. These are referred to as the dictionary of commands and the dictionary of dispatch jump addresses. Color BASIC created a problem when BASIC was written because of the fact that so many commands were required because of the graphics routines and the disk routines that 128 different commands would not suffice. Therefore, a novel method of expanding the number of tokens available by using the token $FF as a special pre-token marker. There are two sets of tokens in Color BASIC, primary and secondary tokens. Primary tokens will have a value from 128 ($80) to 254 ($FE). If an $FF token is encountered in an input line it signifies that the byte immediately following the $FF is the secondary token in question. This can be confusing if you are not careful so you must be aware of the fact that secondary tokens require two bytes, an $FF to identify it as a secondary token and then a number from 128 to 254 immediately after it which identifies the actual token number. Secondary tokens are used to keep track of the intrinsic functions, whereas the primary tokens are used to keep track of commands and statements.

When you type LIST, the computer lists your program; you see the words PRINT, LIST, NEW, LEFT, etc., spelled out on the screen for you. When the program statements are stored in the computer's memory these words are not spelled out. The

tokenized value of these words is what is stored in the memory of the computer. There are two routines in BASIC called crunch and uncrunch which will tokenize or detokenize the BASIC input line. When you list the line to the screen, uncrunch is called and the tokens are detokenized and converted into ASCII strings which are representations of the tokens. When you type a program line into the computer from the keyboard and then hit ENTER to store it into the computer's memory, crunch is called and it tokenizes the line. PRINT, LIST, etc., are crunched down from ASCII representations of those words into tokens. This explains why when you use a monitor to look at an actual BASIC program which is stored in the memory of the computer you will see ASCII strings and PRINT literals or the prompts for input statements spelled out as ASCII words, but you will see graphics blocks interlaced throughout your program. These graphics blocks are the tokens, which have been crunched by BASIC and stored in the computer. Later on you will see in the disk or in the cassette versions of input/output routines either crunched methods of saving the program or ASCII saves. The crunched method of saving the program is the normal method where the BASIC program is merely taken directly from the memory of the computer and stored onto the tape or disk. The ASCII save is where the program is taken from the memory of the computer, uncrunched and then saved on the tape. Generally, when you want to transfer BASIC programs from different computers, i.e., Radio Shack to an Apple, Atari or vice versa, you will have to move the files back and forth in ASCII format because all of the tokens for the different machines will have different values, not to mention different functions.

BASIC program lines are stored in RAM according to the following rules:

1. Start of text is a zero byte.
2. Each program line is preceded by a two-byte RAM link containing the address of the first byte of the next line and a two-byte line number.
3. The compressed (crunched) text is followed by a zero byte.
4. End of text is stored as two zero link bytes.

Now its time to investigate the process, which is used by BASIC in order to interpret the program line. There is a main program loop in BASIC, which is referred to as the command interpretation loop. In this loop commands and statements are evaluated. The token is decoded to determine where the routine is which must be jumped to in order to perform the particular function that needs to be evaluated. The input into the command interpretation loop is taken from console in. Therefore, a clever machine language programmer would be able to have the input to BASIC taken not from the memory of the machine but potentially from a disk file or a cassette file or some other input device, if desired. One of the benefits, if you want to look at it that way, of Color BASIC is that various modules can be added on to BASIC very easily. That is why it is possible to buy extended BASIC, plug it into the computer, turn it on and have it work with no hardware modification. Disk BASIC may be added and provision has even been given for a user add-on. The primary and secondary tokens have a dictionary table, which is the ASCII representation of the various commands and functions spelled out in the memory of the computer. The last byte of the command or function, such as PRINT or LIST, has bit 7 set. This is done so that BASIC can look through the lists of commands and functions and know when a particular command or function spelling is done. In this way an input command in a BASIC program is matched against the command, which is stored in the computer's memory. The computer knows if Extended BASIC, Disk BASIC or a user command table has been implemented. Therefore, it will search through the list of BASIC commands and if it does not find the command that you have typed in the list of BASIC commands, it will then go to Extended BASIC's command table. If it doesn't find it there, it will go to Disk BASIC's command table. If it doesn't find it there it will go to the User Supplied Command Table, if one has been given. When a command is found, there is an associated address in the dispatch table, which tells BASIC where control has to be transferred in order

to perform the various functions, which have been called from the BASIC program. These linkable command tables are explained in the memory map of the computer and are located at $120 (COMVEC). When the first byte of a command table is equal to zero, it is an indication to BASIC that there are no further valid command tables following.

BASIC determines where it is at any one particular time in RAM through the use of the BASIC input pointer. This is an address maintained in the direct page at address $A5. When BASIC wants to get another character from a BASIC input line in order to determine what function or command needs to be interpreted or to get data or anything else from a BASIC program, it gets this data from the BASIC input pointer. This is done by executing the statement JSR $9F. This is a small routine which is moved into the direct page from ROM when BASIC is initialized, and when you call this routine it will increment the BASIC input pointer by one, fetch a character from the address pointed to by the BASIC input pointer, load it into accumulator A and then jump back into the main BASIC ROM. The point at which it jumps back into the main BASIC ROM will determine whether the character, which has been fetched from the BASIC program, is numeric. If the character is numeric, the carry flag will be set. Most people are familiar with the Extended BASIC PCLEAR bug, which was caused when a program was written which would PCLEAR memory during the execution of the program. The PCLEAR statement will cause the BASIC program to be moved up or down in RAM if more or fewer pages of graphic RAM are PCLEARED as a result of the statement. Unfortunately, the BASIC program was moved up or down in RAM, but the BASIC input pointer was never moved, therefore the program would be moved to a new place, but the BASIC input pointer would not be moved a corresponding amount. Therefore, BASIC would begin interpreting new program lines from garbage and you would usually get syntax errors.

The stack is used primarily for normal 6809 functions in the Color Computer It does, however, have several auxiliary functions, which it must provide in order to support Color BASIC. For example, when you use a FOR/NEXT loop, 18 bytes of data are stored on the stack so that you can keep track of the index pointer, step value, the initial value that you started the loop at, and the terminal value at which the FOR/NEXT loop will be stopped. All GOSUB return addresses, which are comprised of 5 bytes, are stored on the stack. The expression evaluation routine uses the stack to store many different bytes and floating point numbers on it during the course of the evaluation of an expression. There are several routines in this BASIC and Extended BASIC such as PAINT, BACKUP, COPY, etc., which also use the stack for temporary storage. The experienced machine language programmer will realize that there can be problems with using the stack for temporary storage and variables. If stack storage is used in recursive loops and there are no controls placed upon the limit of the recursive loops, the stack can grow in an uncontrolled manner and will eventually crash into your program and destroy it. For this reason there is a special routine provided in BASIC, which determines if there is enough free RAM to store the amount of data, which you want to store on the stack. This is required so that if you keep storing data on the stack YOU can merely check to verify that there is enough free RAM left in the machine in order to store the data on the stack. If there is not enough free RAM, an OM error results and you exit from the program in a controlled manner. There are, at this time, at least two bugs known to the author, which will create problems during the execution of a program. Both of these bugs are in the Extended BASIC ROM and neither one has been fixed by version 1.1 of Extended BASIC. The first bug is in the PAINT routine, which uses 6 bytes of stack to remember to paint a particular weird angle, which has been left in an odd shaped paint figure. The paint routine only checks to see if there are 4 bytes of free RAM each time it stores 6 bytes of data on the stack. As such, problems can occur in some very weird shaped paint pictures and the stack could actually creep down and either destroy some variables or parts of the BASIC program

and the user would never know what hit him. The second bug is in the PCOPY statement, which will allow the user to PCOPY from page one to page five if only four pages have been PCLEARED. This is interesting and causes a really good explosion that may completely destroy the BASIC program.

VARIABLES

Variables are used by BASIC to keep track of quantities that may take on different values or change during the course of the execution of the program. COLOR BASIC uses only one type of numeric variable, SINGLE PRECISION. Many other BASICs use different types of variables including integer type and double precision. When COLOR BASIC was first developed, the decision was made not to allow double precision or integer type variables because it would take up too much room in the ROM in order to support these variables types. As a result, we have a smaller and more compact ROM but you lose the efficiency and power that is afforded the use of double precision and integer variable types. No matter which type of variable you have, either single precision or string variable, five bytes are allocated for the storage of this variable in the memory of the computer. All variables are identified by a two-character ASCII string, which is the variable name. Variables may have more than two characters in their name but the characters following the second character will always be ignored. This may lead to some confusion when you're writing a BASIC program, but limiting the number of the characters in the variable name to two makes it much more compact and simple to store the variable 'in the memory of the computer. Any useful program has to deal with alphanumeric data. BASIC has a set of functions to deal with these data. Also, all alphanumeric data may be expressed as a continuous connection of characters, which is viewed by BASIC as the value of a single variable.

Color BASIC has a $ notation which is used to express variables which are strings of alphanumeric data. All of the rules, which apply to normal variables, apply to the string variable.

The limitation on the number of characters that can be stored in a string is 255. The accumulation of characters from an I/O device and the construction of data is accomplished by the concatenation of strings. The operator that is used is +.

Space is allocated for variables only as they are encountered. It is not possible to allocate an array on the basis of 2 single elements; hence, the reason to execute DIM statement before array references. Seven bytes are allocated for each simple variable whether it is a string, number, or user defined function. Each string variable is defined by a five-byte descriptor. This descriptor has five bytes only so that it may be the same length as the single precision floating point variable. Only three of the bytes in the descriptor are actually used to define a string. The other two bytes are wasted but are necessary in order to maintain the some length of descriptor as the floating-point descriptor. Byte zero represents the length of the string and may be any number from 0 to 255. If the length of the string is equal to 0, it indicates a null string. Bytes two and three of the string descriptor are a pointer to the absolute RAM address of the start of the string. When one uses the instruction in BASIC, the access which is returned is the address of the descriptor and this is how you address the string, The absolute RAM address of the string may be anywhere in RAM so that the string may be located in the BASIC program itself, in the string space, or it may even be located in the random buffer file if you have a disk system. If you've been using BASIC for any length of time you may have become familiar with the time delays that occur whenever the BASIC program does what is called housekeeping or garbage collection. What the computer does is to sort all of string space and throws away all of the null strings. This can very often be a long, time-consuming process, which causes the computer to sit and do what appears to be nothing or to be in a hung up state for an extended period of time.

Variables are stored in the variable table, which is immediately following the BASIC program in the memory of the computer. Each variable requires seven bytes to hold its space in the variable table. The first two bytes are the variable name,

the next five bytes are the actual value of the variable if it's a floating point number or it's the five byte descriptor if it's a string variable. The variable names as described above contain two ASCII characters. The method that is used to determine whether the number is a floating point variable or a string variable is the condition of the first variable letter. If this first letter has bit seven set, the variable is a string. If bit seven of the first letter of the variable name is not set, that variable is a floating point variable. This is how BASIC determines the type of variable as it searches for a variable in the variable tables. This searching method should be kept in mind when writing BASIC programs so that you can get maximum efficiency and speed out of your program. Variables that are used most often should be located near the top of the table since BASIC starts at the top and works its way to the bottom when it's searching for a variable name. Looking in the BASIC disassembly of expression evaluation, you will find the method that is used in order to find a variable name (LB357). BASIC gets the variable name, which is found by stripping it off of the BASIC line. It then points itself to the beginning of the table and looks all the way through the variable table until it finds a match. If no match is found, then it inserts a variable in the variable table and a value of 0 or a null string if its a string variable is assigned to it. As you can see, this method can be very slow and cumbersome if you have variables that are very often used and are located at the bottom of a very large variable table. Therefore, if at all possible, define the variables, which are to be used most often at the beginning of your program, and this will cause an increase in speed of your BASIC program.

Floating Point Numbers. Single precision variables are stored in the computer as floating point numbers, which are comprised of an exponent, a four-byte mantissa and the sign of the mantissa. In this way, numbers in the approximate range 1E-39 < X < 1E+39 may be saved. A fifth mantissa byte, the sub byte, (FPSBYT), is used in calculations to achieve 9 significant digits of accuracy.

Floating point numbers are always stored with the mantissa "normalized", that is the mantissa is shifted to the left until a "1" is in the high order bit (7) of the most significant byte. When the mantissa sign in not placed into the high order bit of the most significant mantissa byte, the number is "unpacked".

The exponent is computed such that the mantissa 0 = 1 x 1. It is stored as a signed 8 bit binary plus a bias of $80. Negative exponents are not stored 2's complement. Maximum exponent is 10^{38} and minimum exponent is 10^{-39}, which is stored as $00. A zero exponent is used to flag the number as zero.

Exponent	Approximate Value
FF	10^{38}
A2	10^{10}
7F	10^{-1}
02	10^{-36}
00	10^{-39}

Since the exponent is really a power of 2, it should best be described as the number of left shifts (EXP > $80) or right shifts (EXP < = $80) to be performed on the normalized mantissa to create the actual binary representation of the value.

Example of Floating Point Numbers

Exponent		MS	MANTISSA		LS	Sign
1E38	FF	96	76	99	52	00
4E10	A4	95	02	F9	00	00
2E10	A3	95	02	F9	00	00
1E10	A2	95	02	F9	00	00
1	81	80	00	00	00	00
.5	80	80	00	00	00	00
.25	7F	80	00	00	00	00
1E-4	73	D1	B7	59	59	00
1E-37	06	88	1C	14	14	00
1E-38	02	D9	C7	EE	EE	00
1E-39	00	A0	00	00	00	00
0	00	00	00	00	00	XX
-1	81	80	00	00	00	FF
-10	84	A0	00	00	00	FF

Actual floating point BASIC variables are stored in 5 bytes, rather than 6 bytes as in the floating accumulator. Upon examination, one will note that the most significant byte of the mantissa is always set. If we always assure the number will be in this format, we can use that bit to indicate the sign of the mantissa -- thus, freeing the byte used for sign. This is referred to as "packed" format.

The contents of the floating accumulator may be converted to a double byte integer by calling a subroutine INTCNV which is located at $B3ED The integer is returned in ACCD. An integer can be converted back to floating by loading the two most significant bytes ACCD then calling GIVABF at $B4F4.

Array Variables. Array variables need not be declared with a DIM statement if they have only one dimension and contain fewer than 10 elements. Each element in an array requires 5 bytes of storage and the format of the 5-byte block is the same as simple variables. Arrays are stored in the array table and each array is preceded by a header block of 5+2*N bytes where N = number of dimensions in the array. The first two bytes contain the name of the array, the next two bytes contain the total length of array items and header block, the fifth byte contains the number of dimensions and, finally, 2 bytes per each dimension contain the length of the dimension.

If large arrays are defined and initialized first before simple variables are assigned, much execution time can be lost moving the arrays each time a simple variable is defined. The best strategy to follow in this case is to assign a value to all known simple variables before assigning arrays. This will optimize execution speed.

CONSOLE INPUT/OUTPUT

Console input and console output are the data channels that are used when transferring information into and out of the computer. There are various different methods that are used by different computer manufacturers in order to control the transmission or the flow of data into and out of the computer. It has become useful in most of the jargon to refer to the process of transferring data into or out of the computer as console input or console output. By using a method such as this, one can merely call the console input function if one wants to get a character in Accumulator A, for instance, and then call the console output device and that character will be placed in the appropriate output device, be it cassette, disk, printer or even the screen. Obviously, something else has to be defined when using console in or console out, such as where we are going to send the character to, or from where we will get the character. The Color computer uses the concept of a device number (DEVNUM) in order to define from where the character is coming or where it is going.

The Color Computer has five different device types associated with it: device number 0 is the screen; -1 is the cassette; -2 is the Line Printer; -3 is the DLOAD (RS232 Download) option and device numbers 1 through 15 represent Disk files. Device number 16 is not accessible to the user because it is used by the system as a temporary scratch disk input/output file. The typical method that one uses to access the console in or console out function is to initially define the device number and then jump to the console input or console output and either get the character back into Accumulator A if you are using console in or to transmit the character to the appropriate output device in Accumulator A if using console out.

CONSOLE INPUT - Get a character from an input buffer, which has been defined somewhere in the computer by the routine, which is being used. Generally speaking, one has to OPEN an input channel with the open command. This is not necessary if one is using the screen because the input from the screen comes from the keyboard, which is always an open channel, and it is not necessary to either open or close it. However, if one is trying to use a cassette file, when the cassette file is open for input, there is a buffer established in the memory of the computer which will allow a block of data to be read from the cassette tape. When the user wants a byte of data out of that buffer, he simply calls console in and the byte is returned from the buffer. In this method the buffer is systematically emptied until, when the last character is taken from the buffer, the computer automatically attempts to read another block of data from the cassette file. If further blocks of data are available, then the buffer is refilled and console in can get more data. If there is no longer any further data in the cassette file, then the EOF flag is set and the user is told that there is no longer any data to be gotten from that device. The same type of method is used with DLOAD and, of course, can't be used from the Line Printer, because the Line Printer is only an output device. The method used to transfer data into and out of Disk files will be explained in the Disk BASIC Unravelled.

CONSOLE OUTPUT - Used to transmit data from the computer to an output device. All of the output devices as defined above may be used for outputting data. The method is very similar to the method used for inputting data from files. For example, if one is using a cassette file, an output buffer is established in the computer's memory. Characters are continuously placed in this output buffer until the buffer is filled with 255 characters. At that time the buffer is flushed, that is, the contents of the buffer are written to tape and further input to the buffer is prohibited until the data block is written to the selected device. Upon completion of the data block transfer, the character buffer is reset to a 0 value,

meaning it is empty, and further data may be input into the buffer. This is the same method that is used by Disk and that method of outputting data to a disk file will be explained in Disk BASIC Unravelled.

It should also be noted that you can not open a DLOAD file for output. That feature has not been implemented in the Color Computer -- DLOAD, can only be used to input data.

```
0001        8000        EXBAS       EQU    $8000
0002        A000        BASIC       EQU    $A000
0003        C000        ROMPAK      EQU    $C000
0004
0005        0008        BS          EQU    8                 BACKSPACE
0006        000D        CR          EQU    $D                ENTER KEY
0007        001B        ESC         EQU    $1B               ESCAPE CODE
0008        000A        LF          EQU    $A                LINE FEED
0009        000C        FORMF       EQU    $C                FORM FEED
0010        0020        SPACE       EQU    $20               SPACE (BLANK)
0011
0012        003A        STKBUF      EQU    58                STACK BUFFER ROOM
0013        045E        DEBDEL      EQU    $45E              DEBOUNCE DELAY
0014        00FA        LBUFMX      EQU    250               MAX NUMBER OF CHARS IN A BASIC LINE
0015        00FA        MAXLIN      EQU    $FA               MAXIMUM MS BYTE OF LINE NUMBER
0016
0017        2600        DOSBUF      EQU    $2600             RAM LOAD LOCATION FOR THE DOS COMMAND
0018        0020        DIRLEN      EQU    32                NUMBER OF BYTES IN DIRECTORY ENTRY
0019        0100        SECLEN      EQU    256               LENGTH OF SECTOR IN BYTES
0020        0012        SECMAX      EQU    18                MAXIMUM NUMBER OF SECTORS PER TRACK
0021        1200        TRKLEN      EQU    SECMAX*SECLEN     LENGTH OF TRACK IN BYTES
0022        0023        TRKMAX      EQU    35                MAX NUMBER OF TRACKS
0023        004A        FATLEN      EQU    6+(TRKMAX-1)*2    FILE ALLOCATION TABLE LENGTH
0024        0044        GRANMX      EQU    (TRKMAX-1)*2      MAXIMUM NUMBER OF GRANULES
0025        0119        FCBLEN      EQU    SECLEN+25         FILE CONTROL BLOCK LENGTH
0026        0010        INPFIL      EQU    $10               INPUT FILE TYPE
0027        0020        OUTFIL      EQU    $20               OUTPUT FILE TYPE
0028        0040        RANFIL      EQU    $40               RANDOM/DIRECT FILE TYPE
0029
0030                    * PSEUDO PSEUDO OPS
0031        0021        SKP1        EQU    $21               OP CODE OF BRN   SKIP ONE BYTE
0032        008C        SKP2        EQU    $8C               OP CODE OF CMPX # - SKIP TWO BYTES
0033        0086        SKP1LD      EQU    $86               OP CODE OF LDA # - SKIP THE NEXT BYTE
0034                    *                                    AND LOAD THE VALUE OF THAT BYTE INTO ACCA   THIS
0035                    *                                    IS USUALLY USED TO LOAD ACCA WITH A NON ZERO VALUE
0036
0037                    * REGISTER ADDRESSES
0038        FF00        PIA0        EQU    $FF00             PERIPHERAL INPUT ADAPTER #0
0039        FF20        PIA1        EQU    $FF20             PERIPHERAL INPUT ADAPTER #1
0040        FF20        DA          EQU    PIA1+0            DIGITAL/ANALOG CONVERTER
0041        FF40        DSKREG      EQU    $FF40             DISK CONTROL REGISTER
0042        FF48        FDCREG      EQU    $FF48             1793 CONTROL REGISTER
0043        FFC0        SAMREG      EQU    $FFC0             SAM CONTROL REGISTER
0044
0045  0000                          ORG    0
0046        0000        SETDP       0
0047
0048  0000              ENDFLG      RMB    1                 STOP/END FLAG: POSITIVE=STOP, NEG=END
0049  0001              CHARAC      RMB    1                 TERMINATOR FLAG 1
0050  0002              ENDCUR      RMB    1                 TERMINATOR FLAG 2
0051  0003              TMPLOC      RMB    1                 SCRATCH VARIABLE
0052  0004              IFCTR       RMB    1                 IF COUNTER - HOW MANY IF STATEMENTS IN A LINE
0053  0005              DIMFLG      RMB    1                 *DV* ARRAY FLAG 0=EVALUATE, 1=DIMENSIONING
0054  0006              VALTYP      RMB    1                 *DV* *PV TYPE FLAG: 0=NUMERIC, $FF=STRING
0055  0007              GARBFL      RMB    1                 *TV STRING SPACE HOUSEKEEPING FLAG
0056  0008              ARYDIS      RMB    1                 DISABLE ARRAY SEARCH: 00=ALLOW SEARCH
0057  0009              INPFLG      RMB    1                 *TV INPUT FLAG: READ=0, INPUT<>0
0058  000A              RELFLG      RMB    1                 *TV RELATIONAL OPERATOR FLAG
0059  000B              TEMPPT      RMB    2                 *PV TEMPORARY STRING STACK POINTER
0060  000D              LASTPT      RMB    2                 *PV ADDR OF LAST USED STRING STACK ADDRESS
0061  000F              TEMPTR      RMB    2                 TEMPORARY POINTER
0062  0011              TMPTR1      RMB    2                 TEMPORARY DESCRIPTOR STORAGE (STACK SEARCH)
0063                    ** FLOATING POINT ACCUMULATOR #2 (MANTISSA ONLY)
0064  0013              FPA2        RMB    4                 FLOATING POINT ACCUMULATOR #2 MANTISSA
0065  0017              BOTSTK      RMB    2                 BOTTOM OF STACK AT LAST CHECK
0066  0019              TXTTAB      RMB    2                 *PV BEGINNING OF BASIC PROGRAM
0067  001B              VARTAB      RMB    2                 *PV START OF VARIABLES
0068  001D              ARYTAB      RMB    2                 *PV START OF ARRAYS
0069  001F              ARYEND      RMB    2                 *PV END OF ARRAYS (+1)
0070  0021              FRETOP      RMB    2                 *PV START OF STRING STORAGE (TOP OF FREE RAM)
0071  0023              STRTAB      RMB    2                 *PV START OF STRING VARIABLES
0072  0025              FRESPC      RMB    2                 UTILITY STRING POINTER
0073  0027              MEMSIZ      RMB    2                 *PV TOP OF STRING SPACE
0074  0029              OLDTXT      RMB    2                 SAVED LINE NUMBER DURING A "STOP"
0075  002B              BINVAL      RMB    2                 BINARY VALUE OF A CONVERTED LINE NUMBER
0076  002D              OLDPTR      RMB    2                 SAVED INPUT PTR DURING A "STOP"
0077  002F              TINPTR      RMB    2                 TEMPORARY INPUT POINTER STORAGE
0078  0031              DATTXT      RMB    2                 *PV 'DATA' STATEMENT LINE NUMBER POINTER
0079  0033              DATPTR      RMB    2                 *PV 'DATA' STATEMENT ADDRESS POINTER
0080  0035              DATTMP      RMB    2                 DATA POINTER FOR 'INPUT' & 'READ'
0081  0037              VARNAM      RMB    2                 *TV TEMP STORAGE FOR A VARIABLE NAME
0082  0039              VARPTR      RMB    2                 *TV POINTER TO A VARIABLE DESCRIPTOR
0083  003B              VARDES      RMB    2                 TEMP POINTER TO A VARIABLE DESCRIPTOR
0084  003D              RELPTR      RMB    2                 POINTER TO RELATIONAL OPERATOR PROCESSING ROUTINE
0085  003F              TRELFL      RMB    1                 TEMPORARY RELATIONAL OPERATOR FLAG BYTE
0086
0087                    * FLOATING POINT ACCUMULATORS #3,4 & 5 ARE MOSTLY
```

```
0088                    * USED AS SCRATCH PAD VARIABLES.
0089                    ** FLOATING POINT ACCUMULATOR #3 :PACKED: ($40-$44)
0090  0040              V40           RMB    1
0091  0041              V41           RMB    1
0092  0042              V42           RMB    1
0093  0043              V43           RMB    1
0094  0044              V44           RMB    1
0095                    ** FLOATING POINT ACCUMULATOR #4 :PACKED: ($45-$49)
0096  0045              V45           RMB    1
0097  0046              V46           RMB    1
0098  0047              V47           RMB    1
0099  0048              V48           RMB    2
0100                    ** FLOATING POINT ACCUMULATOR #5 :PACKED: ($4A $4E)
0101  004A              V4A           RMB    1
0102  004B              V4B           RMB    2
0103  004D              V4D           RMB    2
0104                    ** FLOATING POINT ACCUMULATOR #0
0105  004F              FP0EXP        RMB    1                   *PV FLOATING POINT ACCUMULATOR #0 EXPONENT
0106  0050              FPA0          RMB    4                   *PV FLOATING POINT ACCUMULATOR #0 MANTISSA
0107  0054              FP0SGN        RMB    1                   *PV FLOATING POINT ACCUMULATOR #0 SIGN
0108  0055              COEFCT        RMB    1                   POLYNOMIAL COEFFICIENT COUNTER
0109  0056              STRDES        RMB    5                   TEMPORARY STRING DESCRIPTOR
0110  005B              FPCARY        RMB    1                   FLOATING POINT CARRY BYTE
0111                    ** FLOATING POINT ACCUMULATOR #1
0112  005C              FP1EXP        RMB    1                   *PV FLOATING POINT ACCUMULATOR #1 EXPONENT
0113  005D              FPA1          RMB    4                   *PV FLOATING POINT ACCUMULATOR #1 MANTISSA
0114  0061              FP1SGN        RMB    1                   *PV FLOATING POINT ACCUMULATOR #1 SIGN
0115
0116  0062              RESSGN        RMB    1                   SIGN OF RESULT OF FLOATING POINT OPERATION
0117  0063              FPSBYT        RMB    1                   FLOATING POINT SUB BYTE (FIFTH BYTE)
0118  0064              COEFPT        RMB    2                   POLYNOMIAL COEFFICIENT POINTER
0119  0066              LSTTXT        RMB    2                   CURRENT LINE POINTER DURING LIST
0120  0068              CURLIN        RMB    2                   *PV CURRENT LINE # OF BASIC PROGRAM, $FFFF = DIRECT
0121  006A              DEVCFW        RMB    1                   *TV TAB FIELD WIDTH
0122  006B              DEVLCF        RMB    1                   *TV TAB ZONE
0123  006C              DEVPOS        RMB    1                   *TV PRINT POSITION
0124  006D              DEVWID        RMB    1                   *TV PRINT WIDTH
0125  006E              PRTDEV        RMB    1                   *TV PRINT DEVICE: 0=NOT CASSETTE, -1=CASSETTE
0126  006F              DEVNUM        RMB    1                   *PV DEVICE NUMBER: -3=DLOAD, -2=PRINTER,
0127                    *                                                       -1=CASSETTE, 0=SCREEN, 1-15=DISK
0128  0070              CINBFL        RMB    1                   *PV CONSOLE IN BUFFER FLAG: 00=NOT EMPTY, $FF=EMPTY
0129  0071              RSTFLG        RMB    1                   *PV WARM START FLAG: $55=WARM, OTHER=COLD
0130  0072              RSTVEC        RMB    2                   *PV WARM START VECTOR - JUMP ADDRESS FOR WARM START
0131  0074              TOPRAM        RMB    2                   *PV TOP OF RAM
0132  0076                            RMB    2                   SPARE: UNUSED VARIABLES
0133  0078              FILSTA        RMB    1                   *PV FILE STATUS FLAG: 0=CLOSED, 1=INPUT, 2=OUTPUT
0134  0079              CINCTR        RMB    1                   *PV CONSOLE IN BUFFER CHAR COUNTER
0135  007A              CINPTR        RMB    2                   *PV CONSOLE IN BUFFER POINTER
0136  007C              BLKTYP        RMB    1                   *TV CASS BLOCK TYPE: 0=HEADER, 1=DATA, $FF=EOF
0137  007D              BLKLEN        RMB    1                   *TV CASSETTE BYTE COUNT
0138  007E              CBUFAD        RMB    2                   *TV CASSETTE LOAD BUFFER POINTER
0139  0080              CCKSUM        RMB    1                   *TV CASSETTE CHECKSUM BYTE
0140  0081              CSRERR        RMB    1                   *TV ERROR FLAG/CHARACTER COUNT
0141  0082              CPULWD        RMB    1                   *TV PULSE WIDTH COUNT
0142  0083              CPERTM        RMB    1                   *TV BIT COUNTER
0143  0084              CBTPHA        RMB    1                   *TV BIT PHASE FLAG
0144  0085              CLSTSN        RMB    1                   *TV LAST SINE TABLE ENTRY
0145  0086              GRBLOK        RMB    1                   *TV GRAPHIC BLOCK VALUE FOR SET, RESET AND POINT
0146  0087              IKEYIM        RMB    1                   *TV INKEY$ RAM IMAGE
0147  0088              CURPOS        RMB    2                   *PV CURSOR LOCATION
0148  008A              ZERO          RMB    2                   *PV DUMMY - THESE TWO BYTES ARE ALWAYS ZERO
0149  008C              SNDTON        RMB    1                   *TV TONE VALUE FOR SOUND COMMAND
0150  008D              SNDDUR        RMB    2                   *TV DURATION VALUE FOR SOUND COMMAND
0151
0152                    ** THESE BYTES ARE MOVED DOWN FROM ROM
0153                    ***                                      INIT    DESCRIPTION
0154                    *                                        VALUE
0155  008F              CMPMID        RMB    1                   18      *PV 1200/2400 HERTZ PARTITION
0156  0090              CMP0          RMB    1                   24      *PV UPPER LIMIT OF 1200 HERTZ PERIOD
0157  0091              CMP1          RMB    1                   10      *PV UPPER LIMIT OF 2400 HERTZ PERIOD
0158  0092              SYNCLN        RMB    2                   128     *PV NUMBER OF $55'S TO CASSETTE LEADER
0159  0094              BLKCNT        RMB    1                   11      *PV CURSOR BLINK DELAY
0160  0095              LPTBTD        RMB    2                   88      *PV BAUD RATE CONSTANT (600)
0161  0097              LPTLND        RMB    2                   1       *PV PRINTER CARRIAGE RETURN DELAY
0162  0099              LPTCFW        RMB    1                   16      *PV TAB FIELD WIDTH
0163  009A              LPTLCF        RMB    1                   112     *PV LAST TAB ZONE
0164  009B              LPTWID        RMB    1                   132     *PV PRINTER WIDTH
0165  009C              LPTPOS        RMB    1                   0       *PV LINE PRINTER POSITION
0166  009D              EXECJP        RMB    2                   LB4AA   *PV JUMP ADDRESS FOR EXEC COMMAND
0167
0168                    ** THIS ROUTINE PICKS UP THE NEXT INPUT CHARACTER FROM
0169                    ** BASIC. THE ADDRESS OF THE NEXT BASIC BYTE TO BE
0170                    ** INTERPRETED IS STORED AT CHARAD.
0171
0172  009F  0C A7       GETNCH        INC    <CHARAD+1            *PV INCREMENT LS BYTE OF INPUT POINTER
0173  00A1  26 02                     BNE    GETCCH               *PV BRANCH IF NOT ZERO (NO CARRY)
0174  00A3  0C A6                     INC    <CHARAD              *PV INCREMENT MS BYTE OF INPUT POINTER
```

```
0175 00A5 B6          GETCCH      FCB    $B6               *PV OP CODE OF LDA EXTENDED
0176 00A6             CHARAD             2                 *PV THESE 2 BYTES CONTAIN ADDRESS OF THE CURRENT
0177                  *                                        CHARACTER WHICH THE BASIC INTERPRETER IS
0178                  *                                        PROCESSING
0179 00A8 7E AA 1A              JMP    BROMHK            JUMP BACK INTO THE BASIC RUM
0180
0181 00AB             VAB         RMB    1                 = LOW ORDER FOUR BYTES OF THE PRODUCT
0182 00AC             VAC         RMB    1                 = OF A FLOATING POINT MULTIPLICATION
0183 00AD             VAD         RMB    1                 = THESE BYTES ARE USE AS RANDOM DATA
0184 00AE             VAE         RMB    1                 = BY THE RND STATEMENT
0185
0186                  * EXTENDED BASIC VARIABLES
0187 00AF             TRCFLG      RMB    1                 *PV TRACE FLAG Ø=OFF ELSE=ON
0188 00B0             USRADR      RMB    2                 *PV ADDRESS OF THE START OF USR VECTORS
0189 00B2             FORCOL      RMB    1                 *PV FOREGROUND COLOR
0190 00B3             BAKCOL      RMB    1                 *PV BACKGROUND COLOR
0191 00B4             WCOLOR      RMB    1                 *TV WORKING COLOR BEING USED BY EX BASIC
0192 00B5             ALLCOL      RMB    1                 *TV ALL PIXELS IN THIS BYTE SET TO COLOR OF VB3
0193 00B6             PMODE       RMB    1                 *PV PMODE'S MODE ARGUMENT
0194 00B7             ENDGRP      RMB    2                 *PV END OF CURRENT GRAPHIC PAGE
0195 00B9             HORBYT      RMB    1                 *PV NUMBER OF BYTES/HORIZONTAL GRAPHIC LINE
0196 00BA             BEGGRP      RMB    2                 *PV START OF CURRENT GRAPHIC PAGE
0197 00BC             GRPRAM      RMB    1                 *PV START OF GRAPHIC RAM (MS BYTE)
0198 00BD             HORBEG      RMB    2                 *DV* *PV HORIZ COORD - START POINT
0199 00BF             VERBEG      RMB    2                 *DV* *PV VERT COORD - START POINT
0200 00C1             CSSYAL      RMB    1                 *PV SCREEN'S COLOR SET ARGUMENT
0201 00C2             SETFLG      RMB    1                 *PV PRESET/PSET FLAG: Ø=PRESET, 1=PSET
0202 00C3             HOREND      RMB    2                 *DV* *PV HORIZ COORD - ENDING POINT
0203 00C5             VEREND      RMB    2                 *DV* *PV VERT COORD - ENDING POINT
0204 00C7             HORDEF      RMB    2                 *PV HORIZ COORD - DEFAULT COORD
0205 00C9             VERDEF      RMB    2                 *PV VERT COORD - DEFAULT COORD
0206
0207                  * EXTENDED BASIC SCRATCH PAD VARIABLES
0208 00CB             VCB         RMB    2
0209 00CD             VCD         RMB    2
0210 00CF             VCF         RMB    2
0211 00D1             VD1         RMB    2
0212 00D3             VD3         RMB    1
0213 00D4             VD4         RMB    1
0214 00D5             VD5         RMB    1
0215 00D6             VD6         RMB    1
0216 00D7             VD7         RMB    1
0217 00D8             VD8         RMB    1
0218 00D9             VD9         RMB    1
0219 00DA             VDA         RMB    1
0220
0221 00DB             CHGFLG      RMB    1                 *TV FLAG TO INDICATE IF GRAPHIC DATA HAS BEEN CHANGED
0222 00DC             TMPSTK      RMB    2                 *TV STACK POINTER STORAGE DURING PAINT
0223 00DE             OCTAVE      RMB    1                 *PV OCTAVE VALUE (PLAY)
0224 00DF             VOLHI       RMB    1                 *DV* *PV VOLUME HIGH VALUE (PLAY)
0225 00E0             VOLLOW      RMB    1                 *DV* *PV VOLUME LOW VALUE (PLAY)
0226 00E1             NOTELN      RMB    1                 *PV NOTE LENGTH (PLAY)
0227 00E2             TEMPO       RMB    1                 *PV TEMPO VALUE (PLAY)
0228 00E3             PLYTMR      RMB    2                 *TV TIMER FOR THE PLAY COMMAND
0229 00E5             DOTYAL      RMB    1                 *TV DOTTED NOTE TIMER SCALE FACTOR
0230 00E6             DLBAUD      RMB    1                 *DV* *PV DLOAD BAUD RATE CONSTANT $B0=300, $2C=1200
0231 00E7             TIMOUT      RMB    1                 *DV* *PV DLOAD TIMEOUT CONSTANT
0232 00E8             ANGLE       RMB    1                 *DV* *PV ANGLE VALUE (DRAW)
0233 00E9             SCALE       RMB    1                 *DV* *PV SCALE VALUE (DRAW)
0234
0235                  * DSKCON VARIABLES
0236 00EA             DCOPC       RMB    1                 *PV DSKCON OPERATION CODE Ø-3
0237 00EB             DCDRV       RMB    1                 *PV DSKCON DRIVE NUMBER Ø 3
0238 00EC             DCTRK       RMB    1                 *PV DSKCON TRACK NUMBER Ø 34
0239 00ED             DSEC        RMB    1                 *PV DSKCON SECTOR NUMBER 1-18
0240 00EE             DCBPT       RMB    2                 *PV DSKCON DATA POINTER
0241 00F0             DCSTA       RMB    1                 *PV DSKCON STATUS BYTE
0242
0243 00F1             FCBTMP      RMB    2                 TEMPORARY FCB POINTER
0244
0245 00F3                         RMB    13                SPARE: UNUSED VARIABLES
0246
0247
0248                  *                                    BASIC   EXBASIC DOSBASIC
0249
0250 0100             SW3VEC      RMB    3                 $XXXX   $XXXX   $3B3B   SWI3 VECTOR
0251 0103             SW2VEC      RMB    3                 $XXXX   $XXXX   $3B3B   SWI2 VECTOR
0252 0106             SWIVEC      RMB    3                 $XXXX   $XXXX   $XXXX   SWI VECTOR
0253 0109             NMIVEC      RMB    3                 $XXXX   $XXXX   $D7AE   NMI VECTOR
0254 010C             IRQVEC      RMB    3                 $A9B3   $894C   $D7BC   IRQ VECTOR
0255 010F             FRQVEC      RMB    3                 $A0F6   $A0F6   $A0F6   FIRQ VECTOR
0256
0257 0112             TIMVAL
0258 0112             USRJMP      RMB    3                 JUMP ADDRESS FOR BASIC'S USR FUNCTION
0259                  *           RMB    2                 TIMER VALUE FOR EXBAS
0260                  *           RMB    1                 UNUSED BY EXBAS OR DISK BASIC
0261 0115             RVSEED      RMB    1                 * FLOATING POINT RANDOM NUMBER SEED EXPONENT
```

```
0262  0116                      RMB    4                * MANTISSA: INITIALLY SET TO $804FC75259
0263  011A            CASFLG    RMB    1                UPPER CASE/LOWER CASE FLAG: $FF=UPPER, 0=LOWER
0264  011B            DEBVAL    RMB    2                KEYBOARD DEBOUNCE DELAY (SET TO $45E)
0265  011D            EXPJMP    RMB    3                JUMP ADDRESS FOR EXPONENTIATION
0266                  **                                INITIALLY SET TO ERROR FOR BASIC, $8489 FOR EX BASIC
0267
0268                  ***         COMMAND INTERPRETATION VECTOR TABLE
0269
0270                  ** FOUR SETS OF 10 BYTE TABLES:
0271
0272
0273                  ** THE LAST USED TABLE MUST BE FOLLOWED BY A ZERO BYTE
0274                  * THE JUMP TABLE VECTORS (3,4 AND 8,9) POINT TO THE JUMP TABLE FOR
0275                  * THE FIRST TABLE. FOR ALL OTHER TABLES, THESE VECTORS POINT TO A
0276                  * ROUTINE WHICH WILL VECTOR YOU TO THE CORRECT JUMP TABLE.
0277                  * SUPER ENHANCED BASIC HAS MODIFIED THIS SCHEME SO THAT THE USER
0278                  * TABLE MAY NOT BE ACCESSED.  ANY ADDITIONAL TABLES WILL HAVE TO BE
0279                  * ACCESSED FROM A NEW COMMAND HANDLER.
0280
0281                  *           BYTE   DESCRIPTION
0282                  *           0      NUMBER OF RESERVED WORDS
0283                  *           1,2    LOOKUP TABLE OF RESERVED WORDS
0284                  *           3,4    JUMP TABLE FOR COMMANDS (FIRST TABLE)
0285                  *                  VECTOR TO EXPANSION COMMAND HANDLERS (ALL BUT FIRST TABLE)
0286                  *           5      NUMBER OF SECONDARY FUNCTIONS
0287                  *           6,7    LOOKUP TABLE OF SECONDARY FUNCTIONS (FIRST TABLE)
0288                  *                  VECTOR TO EXPANSION SECONDARY COMMAND HANDLERS (ALL BUT
0289                  *                  FIRST TABLE)
0290                  *           8,9    JUMP TABLE FOR SECONDARY FUNCTIONS
0291                  *           10     0 BYTE - END OF TABLE FLAG (LAST TABLE ONLY)
0292
0293  0120            COMVEC    RMB    10               BASIC'S TABLE
0294  012A                      RMB    10               EX BASIC'S TABLE
0295  0134                      RMB    10               DISC BASIC'S TABLE (UNUSED BY EX BASIC)
0296
0297                  **** USR FUNCTION VECTOR ADDRESSES (EX BASIC ONLY)
0298  013E                      RMB    2                USR 0 VECTOR
0299  0140                      RMB    2                USR 1
0300  0142                      RMB    2                USR 2
0301  0144                      RMB    2                USR 3
0302  0146                      RMB    2                USR 4
0303  0148                      RMB    2                USR 5
0304  014A                      RMB    2                USR 6
0305  014C                      RMB    2                USR 7
0306  014E                      RMB    2                USR 8
0307  0150                      RMB    2                USR 9
0308
0309                  *** THE ABOVE 20 BYTE USR ADDR VECTOR TABLE IS MOVED TO
0310                  *** $95F-$972 BY DISC BASIC. THE 20 BYTES FROM $13E-$151
0311                  *** ARE REDEFINED AS FOLLOWS:
0312
0313                  *           RMB    10               USER (SPARE) COMMAND INTERPRETATION TABLE SPACE
0314                  *           FCB    0                END OF COMM INTERP TABLE FLAG
0315                  *           RMB    9                UNUSED BY DISK BASIC
0316
0317                  *           COMMAND INTERPRETATION TABLE VALUES
0318                  *                 BYTE               BASIC   EX BASIC DISK BASIC
0319                  *                 0                  53                      BASIC TABLE
0320                  *                 1,2                $AA66
0321                  *                 3,4                $AB67
0322                  *                 5                  20
0323                  *                 6,7                $AB1A
0324                  *                 8,9                $AA29
0325
0326                  *                 0                          25              EX BASIC TABLE
0327                  *                 1,2                        $8183
0328                  *                 3,4                        $813C   $CE2E   ($CF0A 2.1)
0329                  *                 5                          14
0330                  *                 6,7                        $821E
0331                  *                 8,9                        $8168   $CE56   ($CF32 2.1)
0332
0333                  *                 0                          19 (20 2.1)     DISK BASIC TABLE
0334                  *                 1,2                        $C17F
0335                  *                 3,4                        $C2C0
0336                  *                 5                          6
0337                  *                 6,7                        $C201
0338                  *                 8,9                        $C236
0339
0340
0341  0152            KEYBUF    RMB    8                KEYBOARD MEMORY BUFFER
0342  015A            POTVAL    RMB    1                LEFT VERTICAL JOYSTICK DATA
0343  015B                      RMB    1                LEFT HORIZONTAL JOYSTICK DATA
0344  015C                      RMB    1                RIGHT VERTICAL JOYSTICK DATA
0345  015D                      RMB    1                RIGHT HORIZONTAL JOYSTICK DATA
0346
0347                  * BASIC'S RAM VECTORS - INITIALIZED TO RTS BY COLOR BASIC
0348                  * 25 SETS OF 3 BYTE INSTRUCTIONS WHICH ARE CALLED BY COLOR BASIC
```

```
0349                    * EXTENDED AND DISK BASIC. THEIR PURPOSE IS TO ALLOW ENHANCEMENTS (SUCH
0350                    * AS EX BASIC AND DOS BASIC) AS MORE ROMS ARE ADDED TO THE
0351                    * SYSTEM BY EFFECTIVELY ALLOWING MORE CODE TO BE ADDED TO THE
0352                    * ROUTINES IN EARLIER ROMS. THIS NEW CODE IS LOCATED IN THE NEW ROMS
0353                    * AND THE ADDRESS TO GET TO THE NEW CODE IS IN BYTES 1 & 2 OF THE
0354                    * RAM VECTOR. BYTE Ø WILL CONTAIN A $7E WHICH IS THE FIRST BYTE OF
0355                    * THE JMP INSTRUCTION.
0356                    * THE FIRST ADDRESS IN THIS TABLE IS THE ADDRESS IN BASIC WHICH
0357                    * CALLS THE RAM VECTOR, THE SECOND ADDRESS IS THE VALUE WHICH
0358                    * EX BASIC PUTS IN THE RAM VECTOR (IF ANY) AND THE THIRD ADDRESS
0359                    * IS THE VALUE WHICH DISK BASIC PUTS THERE (IF ANY)
0360
0361
0362                    *                                   2.Ø     2.1     1.Ø     1.1
0363  Ø15E              RVECØ       RMB    3                $A5F6           $C426   $C44B   OPEN COMMAND
0364  Ø161              RVEC1       RMB    3                $A5B9           $C838   $C888   DEVICE NUMBER VALIDITY CHECK
0365  Ø164              RVEC2       RMB    3                $A35F           $C843   $C893   SET PRINT PARAMETERS
0366  Ø167              RVEC3       RMB    3                $A282   $8273   $CB4A   $CC1C   CONSOLE OUT
0367  Ø16A              RVEC4       RMB    3                $A176   $8CF1   $C58F   $C5BC   CONSOLE IN
0368  Ø16D              RVEC5       RMB    3                $A3ED           $C818   $C848   INPUT DEVICE NUMBER CHECK
0369  Ø17Ø              RVEC6       RMB    3                $A4Ø6           $C81B   $C84B   PRINT DEVICE NUMBER CHECK
0370  Ø173              RVEC7       RMB    3                $A426           $CA3B   $CAE9   CLOSE ALL FILES
0371  Ø176              RVEC8       RMB    3                $A42D   $8286   $CA4B   $CAF9   CLOSE ONE FILE
0372  Ø179              RVEC9       RMB    3                $B918   $8E9Ø   $8E9Ø   $8E9Ø   PRINT
0373  Ø17C              RVEC1Ø      RMB    3                $BØ61           $CC5B   $CD35   INPUT
0374  Ø17F              RVEC11      RMB    3                $A549           $C859   $C8A9   BREAK CHECK
0375  Ø182              RVEC12      RMB    3                $A39Ø           $C6B7   $C6E4   INPUTTING A BASIC LINE
0376  Ø185              RVEC13      RMB    3                $A4BF           $CA36   $CAE4   TERMINATING BASIC LINE INPUT
0377  Ø188              RVEC14      RMB    3                $A5CE           $CA6Ø   $C9ØC   EOF COMMAND
0378  Ø18B              RVEC15      RMB    3                $B223   $8846   $CDF6   $CED2   EVALUATE AN EXPRESSION
0379  Ø18E              RVEC16      RMB    3                $AC46           $C6B7   $C6E4   RESERVED FOR ON ERROR GOTO COMMAND
0380  Ø191              RVEC17      RMB    3                $AC49   $88FØ   $C24D   $C265   ERROR DRIVER
0381  Ø194              RVEC18      RMB    3                $AE75   $829C   $C99Ø   $CA3E   RUN
0382  Ø197              RVEC19      RMB    3                $BD22   $87EF                   ASCII TO FLOATING POINT CONVERSION
0383  Ø19A              RVEC2Ø      RMB    3                $AD9E   $82B9           $C8BØ   BASIC'S COMMAND INTERPRETATION LOOP
0384  Ø19D              RVEC21      RMB    3                $A8C4                           RESET/SET/POINT COMMANDS
0385  Ø1AØ              RVEC22      RMB    3                $A91Ø                           CLS
0386                    *                                   $8162                           EXBAS' SECONDARY TOKEN HANDLER
0387                    *                                   $8AFA                           EXBAS' RENUM TOKEN CHECK
0388                    *                                   $975C           $C29A   $C2B2   EXBAS' GET/PUT
0389  Ø1A3              RVEC23      RMB    3                $B821   $83Ø4                   CRUNCH BASIC LINE
0390  Ø1A6              RVEC24      RMB    3                $B7C2                           UNCRUNCH BASIC LINE
0391
0392  Ø1A9              STRSTK      RMB    8*5              STRING DESCRIPTOR STACK
0393  Ø1D1              CFNBUF      RMB    9                CASSETTE FILE NAME BUFFER
0394  Ø1DA              CASBUF      RMB    256              CASSETTE FILE DATA BUFFER
0395  Ø2DA              LINHDR      RMB    2                LINE INPUT BUFFER HEADER
0396  Ø2DC              LINBUF      RMB    LBUFMX+1         BASIC LINE INPUT BUFFER
0397  Ø3D7              STRBUF      RMB    41               STRING BUFFER
0398
0399  Ø4ØØ              VIDRAM      RMB    2ØØ              VIDEO DISPLAY AREA
0400
0401                    *START OF ADDITIONAL RAM VARIABLE STORAGE (DISK BASIC ONLY)
0402  Ø6ØØ              DBUFØ       RMB    SECLEN           I/O BUFFER #Ø
0403  Ø7ØØ              DBUF1       RMB    SECLEN           I/O BUFFER #1
0404  Ø8ØØ              FATBLØ      RMB    FATLEN           FILE ALLOCATION TABLE - DRIVE Ø
0405  Ø84A              FATBL1      RMB    FATLEN           FILE ALLOCATION TABLE - DRIVE 1
0406  Ø894              FATBL2      RMB    FATLEN           FILE ALLOCATION TABLE - DRIVE 2
0407  Ø8DE              FATBL3      RMB    FATLEN           FILE ALLOCATION TABLE - DRIVE 3
0408  Ø928              FCBV1       RMB    16*2             FILE BUFFER VECTORS (15 USER, 1 SYSTEM)
0409  Ø948              RNBFAD      RMB    2                START OF FREE RANDOM FILE BUFFER AREA
0410  Ø94A              FCBADR      RMB    2                START OF FILE CONTROL BLOCKS
0411  Ø94C              DNAMBF      RMB    8                DISK FILE NAME BUFFER
0412  Ø954              DEXTBF      RMB    3                DISK FILE EXTENSION NAME BUFFER
0413  Ø957              DFLTYP      RMB    1                *DV* DISK FILE TYPE: Ø=BASIC, 1=DATA, 2=MACHINE
0414                    *                                        LANGUAGE, 3=TEXT EDITOR SOURCE FILE
0415  Ø958              DASCFL      RMB    1                *DV* ASCII FLAG: Ø=CRUNCHED OR BINARY, $FF=ASCII
0416  Ø959              DRUNFL      RMB    1                RUN FLAG: (IF BIT 1=1 THEN RUN, IF BIT Ø=1, THEN CLOSE
0417                    *                                   ALL FILES BEFORE RUNNING)
0418  Ø95A              DEFDRV      RMB    1                DEFAULT DRIVE NUMBER
0419  Ø95B              FCBACT      RMB    1                NUMBER OF FCBS ACTIVE
0420  Ø95C              DRESFL      RMB    1                RESET FLAG: <>Ø WILL CAUSE A 'NEW' & SHUT DOWN ALL FCBS
0421  Ø95D              DLOADFL     RMB    1                LOAD FLAG: CAUSE A 'NEW' FOLLOWING A LOAD ERROR
0422  Ø95E              DMRGFL      RMB    1                MERGE FLAG: Ø=NØ MERGE, $FF=MERGE
0423  Ø95F              DUSRVC      RMB    2Ø               DISK BASIC USR COMMAND VECTORS
0424                    *** DISK FILE WORK AREA FOR DIRECTORY SEARCH
0425                    *   EXISTING FILE
0426  Ø973              V973        RMB    1                SECTOR NUMBER
0427  Ø974              V974        RMB    2                RAM DIRECTORY IMAGE ADDRESS
0428  Ø976              V976        RMB    1                FIRST GRANULE NUMBER
0429                    *   UNUSED FILE
0430  Ø977              V977        RMB    1                SECTOR NUMBER
0431  Ø978              V978        RMB    2                RAM DIRECTORY IMAGE ADDRESS
0432
0433  Ø97A              WFATVL      RMB    2                WRITE FAT VALUE: NUMBER OF FREE GRANULES WHICH MUST BE TAKEN
0434                                                        FROM THE FAT TO TRIGGER A WRITE FAT TO DISK SEQUENCE
0435  Ø97C              DFFLEN      RMB    2                DIRECT ACCESS FILE RECORD LENGTH
```

```
0436  097E              DRØTRK       RMB    4                 CURRENT TRACK NUMBER, DRIVES Ø,1,2,3
0437  0982              NMIFLG       RMB    1                 NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT
0438  0983              DNMIVC       RMB    2                 NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI
0439                    *                                     INTERRUPT IF THE NMI FLAG IS SET
0440  0985              RDYTMR       RMB    1                 MOTOR TURN OFF TIMER
0441  0986              DRGRAM       RMB    1                 RAM IMAGE OF DSKREG ($FF40)
0442  0987              DVERFL       RMB    1                 VERIFY FLAG: Ø=OFF, $FF=ON
0443  0988              ATTCTR       RMB    1                 READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE
0444                    *                                     DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA
0445                    *                                     BEFORE IT GIVES UP AND ISSUES AN ERROR.
0446
0447  0989              DFLBUF       RMB    SECLEN            INITIALIZED TO SECLEN BY DISKBAS
0448
0449                    *RANDOM FILE RESERVED AREA
0450
0451                    *FILE CONTROL BLOCKS AND BUFFERS
0452
0453                    *GRAPHIC PAGE RESERVED AREA
0454
0455                    *BASIC PROGRAM
0456
0457                    *VARIABLE STORAGE AREA
0458
0459                    *ARRAY STORAGE AREA
0460
0461
0462                    * FREE MEMORY
0463
0464
0465                    *STACK
0466
0467                    *STRING SPACE
0468
0469                    *USER PROGRAM RESERVED AREA
0470
0471                    *END OF RAM
0472
0473  8000              ORG          $8000
0474
0475  8000                           RMB    $2000             EXTENDED BASIC ROM
0476  A000                           RMB    $2000             COLOR BASIC ROM
0477  C000              ROMPAK       EQU    *
0478  C000              DOSBAS       RMB    $2000             DISK BASIC ROM/ENHANCED BASIC INIT CODE
0479  E000                           RMB    $1F00             ENHANCED BASIC
0480
0481
0482                    * I/O AREA
0483
0484  FF00              PIA0         EQU    *                 PERIPHERAL INTERFACE ADAPTER ONE
0485
0486  FF00              BIT0         KEYBOARD ROW 1 AND RIGHT JOYSTICK SWITCH 1
0487                    BIT1         KEYBOARD ROW 2 AND LEFT JOYSTICK SWITCH 1
0488                    BIT2         KEYBOARD ROW 3 AND RIGHT JOYSTICK SWITCH 2
0489                    BIT3         KEYBOARD ROW 4 AND LEFT JOYSTICK SWITCH 2
0490                    BIT4         KEYBOARD ROW 5
0491                    BIT5         KEYBOARD ROW 6
0492                    BIT6         KEYBOARD ROW 7
0493                    BIT7         JOTSTICK COMPARISON IINPUT
0494
0495  FF01              BIT0         CONTROL OF HSYNC (63.5ps)          Ø = IRQ* TO CPU DISABLED
0496                                 INTERRUPT                          1 = IRQ* TO CPU ENABLED
0497                    BIT1         CONTROL OF INTERRUPT               Ø = FLAG SET ON FALLING EDGE OF HS
0498                                 POLARITY                           1 = FLAG SET ON RISING EDGE OF HS
0499                    BIT2         NORMALLY 1                         Ø = CHANGES FF00 TO DATA DIRECTION
0500                    BIT3         SEL 1                              LSB OF TWO ANALOG MUX SELECT LINES
0501                    BIT4         ALWAYS 1
0502                    BIT5         ALWAYS 1
0503                    BIT6         NOT USED
0504                    BIT7         HORIZONTAL SYNC INTERRUPT FLAG
0505
0506  FF02              BIT0         KEYBOARD COLUMN 1
0507                    BIT1         KEYBOARD COLUMN 2
0508                    BIT2         KEYBOARD COLUMN 3
0509                    BIT3         KEYBOARD COLUMN 4
0510                    BIT4         KEYBOARD COLUMN 5
0511                    BIT5         KEYBOARD COLUMN 6
0512                    BIT6         KEYBOARD COLUMN 7 / RAM SIZE OUTPUT
0513                    BIT7         KEYBOARD COLUMN 8
0514
0515  FF03              BIT0         CONTROL OF VSYNC (16.667ms)        Ø = IRQ* TO CPU DISABLED
0516                                 INTERRUPT                          1 = IRQ* TO CPU ENABLED
0517                    BIT1         CONTROL OF INTERRUPT               Ø = FLAG SET ON FALLING EDGE OF FS
0518                                 POLARITY                           1 = FLAG SET ON RISING EDGE OF FS
0519                    BIT2         NORMALLY 1                         Ø = CHANGES FF02 TO DATA DIRECTION
0520                    BIT3         SEL 2                              MSB OF TWO ANALOG MUX SELECT LINES
0521                    BIT4         ALWAYS 1
0522                    BIT5         ALWAYS 1
```

```
0523                    BIT6        NOT USED
0524                    BIT7        FIELD SYNC INTERRUPT FLAG
0525
0526  FF04              *           RMB    28               PIA0 IMAGES
0527  FF20              DA
0528  FF20              PIA1        EQU    *                PERIPHERAL INTERFACE ADAPTER TWO
0529
0530  FF20              BIT0        CASSETTE DATA INPUT
0531                    BIT1        RS-232C DATA OUTPUT
0532                    BIT2        6 BIT D/A LSB
0533                    BIT3        6 BIT D/A
0534                    BIT4        6 BIT D/A
0535                    BIT5        6 BIT D/A
0536                    BIT6        6 BIT D/A
0537                    BIT7        6 BIT D/A MSB
0538
0539  FF21              BIT0        CONTROL OF CD                    0 = FIRQ* TO CPU DISABLED
0540                                  (RS-232C STATUS)                 1 = FIRQ* TO CPU ENABLED
0541                    BIT1        CONTROL OF INTERRUPT             0 = FLAG SET ON FALLING EDGE OF CD
0542                                  POLARITY                         1 = FLAG SET ON RISING EDGE OF CD
0543                    BIT2        NORMALLY 1                       0 = CHANGES FF20 TO DATA DIRECTION
0544                    BIT3        CASSETTE MOTOR CONTROL           0 = OFF          1 = ON
0545                    BIT4        ALWAYS 1
0546                    BIT5        ALWAYS 1
0547                    BIT6        NOT USED
0548                    BIT7        CD INTERRUPT FLAG
0549
0550  FF22              BIT0        RS-232C DATA INPUT
0551                    BIT1        SINGLE BIT SOUND OUTPUT
0552                    BIT2        RAM SIZE INPUT
0553                    BIT3        RGB MONITOR SENSING INPUT        CSS
0554                    BIT4        VDG CONTROL OUTPUT               GM0 & UPPER/LOWER CASE*
0555                    BIT5        VDG CONTROL OUTPUT               GM1 & INVERT
0556                    BIT6        VDG CONTROL OUTPUT               GM2
0557                    BIT7        VDG CONTROL OUTPUT               A*/G
0558
0559  FF23              BIT0        CONTROL OF CARTRIDGE             0 = FIRQ* TO CPU DISABLED
0560                                  INTERRUPT                        1 = FIRQ* TO CPU ENABLED
0561                    BIT1        CONTROL OF INTERRUPT             0 = FLAG SET ON FALLING EDGE OF CART*
0562                                  POLARITY                         1 = FLAG SET ON RISING EDGE OF CART*
0563                    BIT2        NORMALLY 1                       0 = CHANGES FF22 TO DATA DIRECTION
0564                    BIT3        SOUND ENABLE
0565                    BIT4        ALWAYS 1
0566                    BIT5        ALWAYS 1
0567                    BIT6        NOT USED
0568                    BIT7        CARTRIDGE INTERRUPT FLAG
0569
0570  FF24              *           RMB    28               PIA1 IMAGES
0571  FF40              PIA2
0572  FF40              DSKREG      RMB    1                DISK CONTROL REGISTER
0573
0574  FF40              BIT0        DRIVE SELECT 0
0575                    BIT1        DRIVE SELECT 1
0576                    BIT2        DRIVE SELECT 2
0577                    BIT3        DRIVE MOTOR ENABLE               0 = MOTORS OFF          1 = MOTORS ON
0578                    BIT4        WRITE PRECOMPENSATION            0 = NO PRECOMP          1 = PRECOMP
0579                    BIT5        DENSITY FLAG                     0 = SINGLE              1 = DOUBLE
0580                    BIT6        DRIVE SELECT 3
0581                    BIT7        HALT FLAG                        0 = DISABLED            1 = ENABLED
0582
0583  FF41              *           RMB    7                DSKREG IMAGES
0584
0585                    * FLOPPY DISK CONTROLLER INTERNAL REGISTERS
0586  FF48              FDCREG      RMB    1                STATUS/COMMAND REGISTER
0587
0588                    COMMANDS    TYPE   COMMAND                  CODE
0589                                I      RESTORE                  $03
0590                                I      SEEK                     $17
0591                                I      STEP                     $23
0592                                I      STEP IN                  $43
0593                                I      STEP OUT                 $53
0594                                II     READ SECTOR              $80
0595                                II     WRITE SECTOR             $A0
0596                                III    READ ADDRESS             $C0
0597                                III    READ TRACK               $E4
0598                                III    WRITE TRACK              $F4
0599                                IV     FORCE INTERRUPT          $D0
0600
0601                    STATUS      BIT    TYPE I           READ ADDRESS/SECTOR/TRACK      WRITE SECTOR/TRACK
0602                                S0     BUSY             BUSY                           BUSY
0603                                S1     INDEX            DRQ                            DRQ
0604                                S2     TRACK 0          LOST DATA                      LOST DATA
0605                                S3     CRC ERROR        CRC ERROR (EXCEPT TRACK)       CRC ERROR (EXCEPT TRACK)
0606                                S4     SEEK ERROR       RNF (EXCEPT TRACK)             RNF (EXCEPT TRACK)
0607                                S5     HEAD LOADED      RECORD TYPE (SECTOR ONLY)      WRITE FAULT
0608                                S6     WRITE PROTECT                                   WRITE PROTECT
0609                                S7     NOT READY        NOT READY                      NOT READY
```

```
0610
0611  FF49                      RMB    1               TRACK REGISTER
0612  FF4A                      RMB    1               SECTOR REGISTER
0613  FF4B                      RMB    1               DATA REGISTER
0614  FF4C                      RMB    4               FDCREG IMAGES
0615
0616  FF5Ø                      RMB    16              UNUSED SPACE
0617  FF6Ø                      RMB    1               X COORDINATE FOR X-PAD
0618  FF61                      RMB    1               Y COORDINATE FOR X-PAD
0619  FF62                      RMB    1               STATUS REGISTER FOR X-PAD
0620  FF63                      RMB    5               UNUSED
0621                  * RS-232 PROGRAM PAK
0622  FF68                      RMB    1               READ/WRITE DATA REGISTER
0623  FF69                      RMB    1               STATUS REGISTER
0624  FF6A                      RMB    1               COMMAND REGISTER
0625  FF6B                      RMB    1               CONTROL REGISTER
0626  FF6C                      RMB    4
0627  FF7Ø                      RMB    13
0628  FF7D                      RMB    1               SOUND/SPEECH CARTRIDGE RESET
0629  FF7E                      RMB    1               SOUND/SPEECH CARTRIDGE READ/WRITE
0630  FF7F                      RMB    1               MULTI-PAK PROGRAMMING REGISTER
0631
0632  FF8Ø                      RMB    64              RESERVED FOR FUTURE EXPANSION
0633
0634
0635  FFCØ            SAMREG    EQU    *               SAM CONTROL REGISTERS
0636
0637  FFCØ            VØCLR     RMB    1               CLEAR COCO GRAPHICS MODE VØ
0638  FFC1            VØSET     RMB    1               SET COCO GRAPHICS MODE VØ
0639  FFC2            V1CLR     RMB    1               CLEAR COCO GRAPHICS MODE V1
0640  FFC3            V1SET     RMB    1               SET COCO GRAPHICS MODE V1
0641  FFC4            V2CLR     RMB    1               CLEAR COCO GRAPHICS MODE V2
0642  FFC5            V2SET     RMB    1               SET COCO GRAPHICS MODE V2
0643  FFC6            FØCLR     RMB    1               CLEAR COCO GRAPHICS OFFSET FØ
0644  FFC7            FØSET     RMB    1               SET COCO GRAPHICS OFFSET FØ
0645  FFC8            F1CLR     RMB    1               CLEAR COCO GRAPHICS OFFSET F1
0646  FFC9            F1SET     RMB    1               SET COCO GRAPHICS OFFSET F1
0647  FFCA            F2CLR     RMB    1               CLEAR COCO GRAPHICS OFFSET F2
0648  FFCB            F2SET     RMB    1               SET COCO GRAPHICS OFFSET F2
0649  FFCC            F3CLR     RMB    1               CLEAR COCO GRAPHICS OFFSET F3
0650  FFCD            F3SET     RMB    1               SET COCO GRAPHICS OFFSET F3
0651  FFCE            F4CLR     RMB    1               CLEAR COCO GRAPHICS OFFSET F4
0652  FFCF            F4SET     RMB    1               SET COCO GRAPHICS OFFSET F4
0653  FFDØ            F5CLR     RMB    1               CLEAR COCO GRAPHICS OFFSET F5
0654  FFD1            F5SET     RMB    1               SET COCO GRAPHICS OFFSET F5
0655  FFD2            F6CLR     RMB    1               CLEAR COCO GRAPHICS OFFSET F6
0656  FFD3            F6SET     RMB    1               SET COCO GRAPHICS OFFSET F6
0657  FFD4                      RMB    4               RESERVED
0658  FFD8            R1CLR     RMB    1               CLEAR CPU RATE, (Ø.89 MHz)
0659  FFD9            R1SET     RMB    1               SET CPU RATE, (1.78 MHz)
0660  FFDA                      RMB    4               RESERVED
0661  FFDE            ROMCLR    RMB    1               ROM DISABLED
0662  FFDF            ROMSET    RMB    1               ROM ENABLED
0663
0664  FFEØ                      RMB    18              RESERVED FOR FUTURE MPU ENHANCEMENTS
0665                  *         INTERRUPT VECTORS
0666  FFF2            SWI3      RMB    2
0667  FFF4            SWI2      RMB    2
0668  FFF6            FIRQ      RMB    2
0669  FFF8            IRQ       RMB    2
0670  FFFA            SWI       RMB    2
0671  FFFC            NMI       RMB    2
0672  FFFE            RESETV    RMB    2
```

```
0001   A000                          ORG   $A000
0002   A000  A1 CB          POLCAT   FDB   KEYIN                  GET A KEYSTROKE
0003   A002  A2 82          CHROUT   FDB   PUTCHR                 OUTPUT A CHARACTER
0004   A004  A7 7C          CSRDON   FDB   CASON                  TURN ON CASSETTE MOTOR, START READING
0005   A006  A7 0B          BLKIN    FDB   GETBLK                 READ A BLOCK FROM CASSETTE
0006   A008  A7 F4          BLKOUT   FDB   SNDBLK                 WRITE A BLOCK TO CASSETTE
0007   A00A  A9 DE          JOYIN    FDB   GETJOY                 READ JOYSTICKS
0008   A00C  A7 D8          WRTLDR   FDB   WRLDR                  TURN ON MOTOR AND WRITE $55 S TO CASSETTE
0009                         *
0010   A00E  10 CE 03 D7    LA00E    LDS   #LINBUF+LBUFMX+1        SET STACK TO TOP OF LINE INPUT BUFFER
0011   A012  86 37                   LDA   #$37                   *
0012   A014  B7 FF 23                STA   PIA1+3                 * ENABLE 63.5 MICROSECOND INTERRUPT
0013   A017  96 71                   LDA   RSTFLG                 GET WARM START FLAG
0014   A019  81 55                   CMPA  #$55                   IS IT A WARM START?
0015   A01B  26 57                   BNE   BACDST                 NO - D0 A COLD START
0016   A01D  9E 72                   LDX   RSTVEC                 WARM START VECTOR
0017   A01F  A6 84                   LDA   ,X                     GET FIRST BYTE OF WARM START ADDR
0018   A021  81 12                   CMPA  #$12                   IS IT NOP?
0019   A023  26 4F                   BNE   BACDST                 NO - DO A COLD START
0020   A025  6E 84                   JMP   ,X                     YES, G0 THERE
0021
0022   A027  31 8C E4       RESVEC   LEAY  LA00E,PC               POINT Y TO WARM START CHECK CODE
0023   A02A  8E FF 20       LA02A    LDX   #PIA1                  POINT X TO PIA1
0024   A02D  6F 1D                   CLR   -3,X                   CLEAR PIA0 CONTROL REGISTER A
0025   A02F  6F 1F                   CLR   -1,X                   CLEAR PIA0 CONTROL REGISTER B
0026   A031  6F 1C                   CLR   -4,X                   SET PIA0 SIDE A TO INPUT
0027   A033  CC FF 34                LDD   #$FF34                 *
0028   A036  A7 1E                   STA   -2,X                   * SET PIA0 SIDE B TO OUTPUT
0029   A038  E7 1D                   STB   -3,X                   * ENABLE PIA0 PERIPHERAL REGISTERS, DISABLE PIA0
0030   A03A  E7 1F                   STB   -1,X                   * MPU INTERRUPTS, SET CA2, CA1 TO OUTPUTS
0031   A03C  6F 01                   CLR   1,X                    CLEAR CONTROL REGISTER A ON PIA1
0032   A03E  6F 03                   CLR   3,X                    CLEAR CONTROL REGISTER B ON PIA1
0033   A040  4A                      DECA                         A REG NOW HAS $FE
0034   A041  A7 84                   STA   ,X                     BITS 1-7 ARE OUTPUTS, BIT 0 IS INPUT ON PIA1 SIDE A
0035   A043  86 F8                   LDA   #$F8                   =
0036   A045  A7 02                   STA   2,X                    = BITS 0-2 ARE INPUTS, BITS 3-7 ARE OUTPUTS ON B SIDE
0037   A047  E7 01                   STB   1,X                    * ENABLE PERIPHERAL REGISTERS, DISABLE PIA1 MPU
0038   A049  E7 03                   STB   3,X                    * INTERRUPTS AND SET CA2, CB2 AS OUTPUTS
0039   A04B  6F 02                   CLR   2,X                    SET 6847 MODE TO ALPHA-NUMERIC
0040   A04D  C6 02                   LDB   #$02                   *
0041   A04F  E7 84                   STB   ,X                     * MAKE RS232 OUTPUT MARKING
0042   A051  CE FF C0                LDU   #SAMREG                SAM CONTROL REGISTER ADDR
0043   A054  C6 10                   LDB   #16                    16 SAM CONTROL REGISTER BITS
0044   A056  A7 C1          LA056    STA   ,U++                   ZERO OUT SAM CONTROL REGISTER BIT
0045   A058  5A                      DECB                         * DECREMENT COUNTER AND
0046   A059  26 FB                   BNE   LA056                  * BRANCH IF NOT DONE
0047   A05B  B7 FF C9                STA   SAMREG+9               SET DISPLAY PAGE AT $400
0048   A05E  1F 9B                   TFR   B,DP                   SET DIRECT PAGE TO ZERO
0049   A060  C6 04                   LDB   #$04                   USE AS A MASK TO CHECK RAMSZ INPUT
0050   A062  A7 1E                   STA   -2,X                   SET RAMSZ STROBE HIGH
0051   A064  E5 02                   BITB  2,X                    CHECK RAMSZ INPUT
0052   A066  27 0A                   BEQ   LA072                  BRANCH IF JUMPER SET FOR 4K RAMS
0053   A068  6F 1E                   CLR   -2,X                   SET RAMSZ STROBE LOW
0054   A06A  E5 02                   BITB  2,X                    CHECK RAMSZ INPUT
0055   A06C  27 02                   BEQ   LA070                  BRANCH IF JUMPER SET FOR 64K RAMS
0056   A06E  33 5E                   LEAU  -2,U                   ADJUST POINTER TO SET SAM FOR 16K RAMS
0057   A070  A7 5D          LA070    STA   -3,U                   PROGRAM SAM FOR 16K OR 64K RAMS
0058   A072  6E A4          LA072    JMP   ,Y                     GO DO A WARM OR COLD START
0059                         * COLD START ENTRY
0060   A074  8E 04 01       BACDST   LDX   #VIDRAM+1              POINT X TO CLEAR 1ST 1K OF RAM
0061   A077  6F 83          LA077    CLR   ,--X                   MOVE POINTER DOWN TWO-CLEAR BYTE
0062   A079  30 01                   LEAX  1,X                    ADVANCE POINTER ONE
0063   A07B  26 FA                   BNE   LA077                  KEEP GOING IF NOT AT BOTTOM OF PAGE 0
0064   A07D  BD A9 28                JSR   LA928                  CLEAR SCREEN
0065   A080  6F 80                   CLR   ,X+                    CLEAR 1ST BYTE OF BASIC PROGRAM
0066   A082  9F 19                   STX   TXTTAB                 BEGINNING OF BASIC PROGRAM
0067   A084  A6 02          LA084    LDA   2,X                    LOOK FOR END OF MEMORY
0068   A086  43                      COMA                         * COMPLEMENT IT AND PUT IT BACK
0069   A087  A7 02                   STA   2,X                    * INTO SYSTEM MEMORY
0070   A089  A1 02                   CMPA  2,X                    IS IT RAM?
0071   A08B  26 06                   BNE   LA093                  BRANCH IF NOT (ROM, BAD RAM OR NO RAM)
0072   A08D  30 01                   LEAX  1,X                    MOVE POINTER UP ONE
0073   A08F  63 01                   COM   1,X                    RE-COMPLEMENT TO RESTORE BYTE
0074   A091  20 F1                   BRA   LA084                  KEEP LOOKING FOR END OF RAM
0075   A093  9F 74          LA093    STX   TOPRAM                 SAVE ABSOLUTE TOP OF RAM
0076   A095  9F 27                   STX   MEMSIZ                 SAVE TOP OF STRING SPACE
0077   A097  9F 23                   STX   STRTAB                 SAVE START OF STRING VARIABLES
0078   A099  30 89 FF 38             LEAX  -200,X                 CLEAR 200 - DEFAULT STRING SPACE TO 200 BYTES
0079   A09D  9F 21                   STX   FRETOP                 SAVE START OF STRING SPACE
0080   A09F  1F 14                   TFR   X,S                    PUT STACK THERE
0081   A0A1  8E A1 0D                LDX   #LA10D                 POINT X TO ROM SOURCE DATA
0082   A0A4  CE 00 8F                LDU   #CMPMID                POINT U TO RAM DESTINATION
0083   A0A7  C6 1C                   LDB   #28                    MOVE 28 BYTES
0084   A0A9  BD A5 9A                JSR   LA59A                  MOVE 28 BYTES FROM ROM TO RAM
0085   A0AC  CE 01 0C                LDU   #IRQVEC                POINT U TO NEXT RAM DESTINATION
0086   A0AF  C6 1E                   LDB   #30                    MOVE 30 MORE BYTES
0087   A0B1  BD A5 9A                JSR   LA59A                  MOVE 30 BYTES FROM ROM TO RAM
0088   A0B4  AE 14                   LDX   -12,X                  POINT X TO SYNTAX ERROR ADDRESS
0089   A0B6  AF 43          LA0B6    STX   3,U                    * SET EXBAS COMMAND INTERPRETATION
```

```
0090   A0B8  AF 48                    STX   8,U                    * HANDLERS TO SYNTAX ERROR
0091   A0BA  8E 01 5E                 LDX   #RVEC0                 POINT X TO START OF RAM VECTORS
0092   A0BD  CC 39 4B                 LDD   #$394B                 SET UP TO SAVE 75 RTS
0093   A0C0  A7 80           LA0C0    STA   ,X+                    FILL THE RAM VECTORS WITH RTS
0094   A0C2  5A                       DECB                         * DECREMENT COUNTER AND
0095   A0C3  26 FB                    BNE   LA0C0                  * BRANCH IF NOT DONE
0096   A0C5  B7 02 D9                 STA   LINHDR-1               PUT RTS IN LINHDR-1
0097   A0C8  BD AD 19                 JSR   LAD19                  G0 DO A  NEW
0098   A0CB  8E 45 58                 LDX   #$4558                 ASCII  EX  (FIRST TWO LETTERS OF  EXTENDED )
0099   A0CE  BC 80 00                 CMPX  EXBAS                  SEE IF EXTENDED ROM IS THERE
0100   A0D1  10 27 DF 2D              LBEQ  EXBAS+2                IF IT IS, BRANCH TO IT
0101   A0D5  1C AF                    ANDCC #$AF                   ENABLE IRQ, FIRQ
0102   A0D7  8E A1 46                 LDX   #LA147-1               POINT X TO COLOR BASIC COPYRIGHT MESSAGE
0103   A0DA  BD B9 9C                 JSR   LB99C                  PRINT  COLOR BASIC
0104   A0DD  8E A0 E8                 LDX   #BAWMST                WARM START ADDRESS
0105   A0E0  9F 72                    STX   RSTVEC                 SAVE IT
0106   A0E2  86 55                    LDA   #$55                   WARM START FLAG
0107   A0E4  97 71                    STA   RSTFLG                 SAVE IT
0108   A0E6  20 0B                    BRA   LA0F3                  GO TO BASIC S MAIN LOOP
0109   A0E8  12              BAWMST   NOP                          NOP REQ D FOR WARM START
0110   A0E9  0F 6F                    CLR   DEVNUM                 SET DEVICE NUMBER TO SCREEN
0111   A0EB  BD AD 33                 JSR   LAD33                  DO PART OF A NEW
0112   A0EE  1C AF                    ANDCC #$AF                   ENABLE IRQ,FIRQ
0113   A0F0  BD A9 28                 JSR   LA928                  CLEAR SCREEN
0114   A0F3  7E AC 73           LA0F3 JMP   LAC73                  GO TO MAIN LOOP OF BASIC
0115                           *
0116                           * FIRQ SERVICE ROUTINE
0117   A0F6  7D FF 23           BFRQSV TST   PIA1+3                 CARTRIDGE INTERRUPT?
0118   A0F9  2B 01                    BMI   LA0FC                  YES
0119   A0FB  3B                       RTI
0120   A0FC  BD A7 D1           LA0FC JSR   LA7D1                  DELAY FOR A WHILE
0121   A0FF  BD A7 D1                 JSR   LA7D1                  KEEP DELAYING
0122   A102  31 8C 03          LA102 LEAY  <LA108,PC              Y = ROM-PAK START UP VECTOR
0123   A105  7E A0 2A                 JMP   LA02A                  GO DO INITIALIZATION
0124   A108  0F 71             LA108 CLR   RSTFLG                 CLEAR WARM START FLAG
0125   A10A  7E C0 00                 JMP   ROMPAK                 JUMP TO EXTERNAL ROM PACK
0126                           *
0127                           * THESE BYTES ARE MOVED TO ADDRESSES $8F - $AA THE DIRECT PAGE
0128   A10D  12                 LA10D FCB   18                     MID BAND PARTITION OF 1200/2400 HERTZ PERIOD
0129   A10E  18                       FCB   24                     UPPER LIMIT OF 1200 HERTZ PERIOD
0130   A10F  0A                       FCB   10                     UPPER LIMIT OF 2400 HERTZ PERIOD
0131   A110  00 80                    FDB   128                    NUMBER OF 55 S TO CASSETTE LEADER
0132   A112  0B                       FCB   11                     CURSOR BLINK DELAY
0133   A113  00 58                    FDB   88                     CONSTANT FOR 600 BAUD VER 1.2 & UP
0134   A115  00 01                    FDB   1                      PRINTER CARRIAGE RETURN DELAY
0135   A117  10                       FCB   16                     TAB FIELD WIDTH
0136   A118  70                       FCB   112                    LAST TAB ZONE
0137   A119  84                       FCB   132                    PRINTER WIDTH
0138   A11A  00                       FCB   0                      LINE PRINTER POSITION
0139   A11B  B4 4A                    FDB   LB44A                  ARGUMENT OF EXEC COMMAND - SET TO  FC  ERROR
0140                           * LINE INPUT ROUTINE
0141   A11D  0C A7                    INC   CHARAD+1
0142   A11F  26 02                    BNE   LA123
0143   A121  0C A6                    INC   CHARAD
0144   A123  B6 00 00           LA123 LDA   >0000
0145   A126  7E AA 1A                 JMP   BROMHK
0146                           *
0147                           * THESE BYTES ARE MOVED TO ADDRESSES $10C-$129
0148   A129  7E A9 B3                 JMP   BIRQSV                 IRQ SERVICE
0149 > A12C  7E A0 F6                 JMP   BFRQSV                 FIRQ SERVICE
0150   A12F  7E B4 4A                 JMP   LB44A                  USR ADDRESS FOR 8K BASIC (INITIALIZED TO  FC  ERROR)
0151   A132  80 4F                    FCB   $80                    *RANDOM SEED
0152   A133  4F C7                    FDB   $4FC7                  *RANDON SEED OF MANTISSA
0153   A135  52 59                    FCB   $5259                  *.811635157
0154   A137  FF                       FCB   $FF                    UPPER CASE/LOWER CASE FLAG (STARTS SET TO UPPER)
0155   A138  04 5E                    FDB   DEBDEL                 KEYBOARD DEBOUNCE DELAY
0156   A13A  7E B2 77                 JMP   LB277                  DISPATCH FOR EXPONENTIATION (INITIALIZED TO SYNTAX ERROR)
0157                           * BASIC COMMAND INTERPRETATION TABLE ROM IMAGE
0158   A13D  35                 LA13D FCB   53                     53 BASIC COMMANDS
0159   A13E  AA 66              LA13E FDB   LAA66                  POINTS TO RESERVED WORDS
0160   A140  AB 67              LA140 FDB   LAB67                  POINTS TO JUMP TABLE FOR COMMANDS
0161   A142  14                 LA142 FCB   20                     20 BASIC SECONDARY COMMANDS
0162   A143  AB 1A              LA143 FDB   LAB1A                  POINTS TO SECONDARY FUNCTION RESERVED WORDS
0163   A145  AA 29              LA145 FDB   LAA29                  POINTS TO SECONDARY FUNCTION JUMP TABLE
0164                           * COPYRIGHT MESSAGES
0165   A147  43 4F 4C 4F 52 20 LA147 FCC   'COLOR BASIC 1.2'
0166   A14D  42 41 53 49 43 20
0167   A153  31 2E 32
0168   A156  0D                 LA156 FCB   CR
0169   A157  28 43 29 20 31 39 LA157 FCC   '(C) 1982 TANDY'
0170   A15D  38 32 20 54 41 4E
0171   A163  44 59
0172   A165  00                 LA165 FCB   $00
0173   A166  4D 49 43 52 4F 53 LA166 FCC   'MICROSOFT'
0174   A16C  4F 46 54
0175   A16F  0D 00              LA16F FCB   CR,$00
0176
0177   A171  8D 03              LA171 BSR   LA176                  GET A CHARACTER FROM CONSOLE IN
0178   A173  84 7F                    ANDA  #$7F                   MASK OFF BIT 7
```

```
0179    A175  39                        RTS
0180
0181                            * CONSOLE IN
0182    A176  BD Ø1 6A          LA176     JSR   RVEC4                    HOOK INTO RAM
0183    A179  ØF 7Ø                       CLR   CINBFL                   RESET CONSOLE IN BUFFER FLAG = FULL
0184    A17B  ØD 6F             LA17B     TST   DEVNUM                   CHECK DEVICE NUMBER
0185    A17D  27 32                       BEQ   LA1B1                    GØ DO CURSOR AND GET A KEY IF SCREEN MODE
0186    A17F  ØD 79                       TST   CINCTR                   TEST CHARACTER COUNTER
0187    A181  26 Ø3                       BNE   LA186                    NOT EMPTY - READ IN SOME CASSETTE DATA
0188    A183  Ø3 7Ø                       COM   CINBFL                   SET TO $FF: CONSOLE IN BUFFER EMPTY
0189    A185  39                LA185     RTS
0190                            *
0191    A186  34 74             LA186     PSHS  U,Y,X,B                  SAVE REGISTERS
0192    A188  9E 7A                       LDX   CINPTR                   PICK UP BUFFER POINTER
0193    A18A  A6 8Ø                       LDA   ,X+                      GET NEXT CHAR
0194    A18C  34 Ø2                       PSHS  A                        SAVE CHAR ON STACK
0195    A18E  9F 7A                       STX   CINPTR                   SAVE NEW BUFFER POINTER
0196    A19Ø  ØA 79             LA19Ø     DEC   CINCTR                   DECR CHAR COUNT
0197    A192  26 Ø3                       BNE   LA197                    RETURN IF BUFFER NOT EMPTY
0198    A194  BD A6 35                    JSR   LA635                    GO READ TAPE
0199    A197  35 F6             LA197     PULS  A,B,X,Y,U,PC             RESTORE REGISTERS
0200                            *
0201    A199  ØA 94             LA199     DEC   BLKCNT                   CURSOR BLINK DELAY
0202    A19B  26 ØE                       BNE   LA1AB                    NOT TIME FOR NEW COLOR
0203    A19D  C6 ØB                       LDB   #11                      *
0204    A19F  D7 94                       STB   BLKCNT                   *RESET DELAY COUNTER
0205    A1A1  9E 88                       LDX   CURPOS                   GET CURSOR POSITION
0206    A1A3  A6 84                       LDA   ,X                       GET CURRENT CURSOR CHAR
0207    A1A5  8B 1Ø                       ADDA  #$1Ø                     BUMP TO NEXT COLOR
0208    A1A7  8A 8F                       ORA   #$8F                     MAKE SURE IT S A SOLID GRAPHICS BLOCK
0209    A1A9  A7 84                       STA   ,X                       STORE TO SCREEN
0210    A1AB  8E Ø4 5E          LA1AB     LDX   #DEBDEL                  CURSOR BLINK DELAY
0211    A1AE  7E A7 D3          LA1AE     JMP   LA7D3                    DELAY WHILE X DECREMENTS TO ZERO
0212
0213                            * BLINK CURSOR WHILE WAITING FOR A KEYSTROKE
0214    A1B1  34 14             LA1B1     PSHS  X,B                      SAVE REGISTERS
0215    A1B3  8D E4             LA1B3     BSR   LA199                    GO DO CURSOR
0216    A1B5  8D 14                       BSR   KEYIN                    GO CHECK KEYBOARD
0217    A1B7  27 FA                       BEQ   LA1B3                    LOOP IF NO KEY DOWN
0218    A1B9  C6 6Ø                       LDB   #$6Ø                     BLANK
0219    A1BB  E7 9F ØØ 88                 STB   [CURPOS]                 BLANK CURRENT CURSOR CHAR ON SCREEN
0220    A1BF  35 94             LA1BF     PULS  B,X,PC
0221                            *
0222                            * THIS ROUTINE GETS A KEYSTROKE FROM THE KEYBOARD IF A KEY
0223                            * IS DOWN. IT RETURNS ZERO TRUE IF THERE WAS NO KEY DOWN.
0224                            *
0225    A1C1  7F FF Ø2          LA1C1     CLR   PIA0+2                   CLEAR COLUMN STROBE
0226    A1C4  B6 FF ØØ                    LDA   PIA0                     READ KEY ROWS
0227    A1C7  43                          COMA                           COMPLEMENT ROW DATA
0228    A1C8  48                          ASLA                           SHIFT OFF JOYSTICK DATA
0229    A1C9  27 79                       BEQ   LA244                    RETURN IF NO KEYS OR FIRE BUTTONS DOWN
0230    A1CB  34 54             KEYIN     PSHS  U,X,B                    SAVE REGISTERS
0231    A1CD  CE FF ØØ                    LDU   #PIA0                    POINT U TO PIA0
0232    A1DØ  8E Ø1 52                    LDX   #KEYBUF                  POINT X TO KEYBOARD MEMORY BUFFER
0233    A1D3  4F                          CLRA                           * CLEAR CARRY FLAG, SET COLUMN COUNTER (ACCA)
0234    A1D4  4A                          DECA                           * TO $FF
0235    A1D5  34 12                       PSHS  X,A                      SAVE COLUMN CTR & 2 BLANK (X REG) ON STACK
0236    A1D7  A7 42                       STA   2,U                      INITIALIZE COLUMN STROBE TO $FF
0237    A1D9  69 42             LA1D9     ROL   2,U                      * ROTATE COLUMN STROBE DATA LEFT 1 BIT, CARRY
0238    A1DB  24 43                       BCC   LA22Ø                    * INTO BIT Ø - BRANCH IF 8 SHIFTS DONE
0239    A1DD  6C 6Ø                       INC   ,S                       INCREMENT COLUMN COUNTER
0240    A1DF  8D 59                       BSR   LA23A                    READ KEYBOARD ROW DATA
0241    A1E1  A7 61                       STA   1,S                      TEMP STORE KEY DATA
0242    A1E3  A8 84                       EORA  ,X                       SET ANY BIT WHERE A KEY HAS MOVED
0243    A1E5  A4 84                       ANDA  ,X                       ACCA=Ø IF NO NEW KEY DOWN, <7Ø IF KEY WAS RELEASED
0244    A1E7  E6 61                       LDB   1,S                      GET NEW KEY DATA
0245    A1E9  E7 8Ø                       STB   ,X+                      STORE IT IN KEY MEMORY
0246    A1EB  4D                          TSTA                           WAS A NEW KEY DOWN?
0247    A1EC  27 EB                       BEQ   LA1D9                    NO-CHECK ANOTHER COLUMN
0248    A1EE  E6 42                       LDB   2,U                      * GET COLUMN STROBE DATA AND
0249    A1FØ  E7 62                       STB   2,S                      * TEMP STORE IT ON THE STACK
0250                            * THIS ROUTINE CONVERTS THE KEY DEPRESSION INTO A NUMBER
0251                            * FROM Ø-5Ø IN ACCB CORRESPONDING TO THE KEY THAT WAS DOWN
0252    A1F2  C6 F8                       LDB   #$F8                     TO MAKE SURE ACCB=Ø AFTER FIRST ADDB #8
0253    A1F4  CB Ø8             LA1F4     ADDB  #$Ø8                     ADD 8 FOR EACH ROW OF KEYBOARD
0254    A1F6  44                          LSRA                           ACCA HAS THE ROW NUMBER OF THIS KEY - ADD 8 FOR EACH ROW
0255    A1F7  24 FB                       BCC   LA1F4                    GO ON UNTIL A ZERO APPEARS IN THE CARRY FLAG
0256    A1F9  EB 6Ø                       ADDB  ,S                       ADD IN THE COLUMN NUMBER
0257                            * NOW CONVERT THE VALUE IN ACCB INTO ASCII
0258    A1FB  27 48                       BEQ   LA245                    THE  AT SIGN  KEY WAS DOWN
0259    A1FD  C1 1A                       CMPB  #26                      WAS IT A LETTER?
0260    A1FF  22 46                       BHI   LA247                    NO
0261    A2Ø1  CA 4Ø                       ORB   #$4Ø                     YES, CONVERT TO UPPER CASE ASCII
0262    A2Ø3  8D 29                       BSR   LA22E                    CHECK FOR THE SHIFT KEY
0263    A2Ø5  BA Ø1 1A                    ORA   CASFLG                   *  OR  IN THE CASE FLAG & BRANCH IF IN UPPER
0264    A2Ø8  26 Ø2                       BNE   LA2ØC                    * CASE MODE OR SHIFT KEY DOWN
0265    A2ØA  CA 2Ø                       ORB   #$2Ø                     CONVERT TO LOWER CASE
0266    A2ØC  E7 6Ø             LA2ØC     STB   ,S                       TEMP STORE ASCII VALUE
0267    A2ØE  BE Ø1 1B                    LDX   DEBVAL                   GET KEYBOARD DEBOUNCE
```

```
0268   A211  8D 9B                    BSR   LA1AE
0269   A213  C6 FF                    LDB   #$FF                  SET COLUMN STROBE TO ALL ONES (NO
0270   A215  8D 21                    BSR   LA238                 STROBE) AND READ KEYBOARD
0271   A217  4C                       INCA                        = INCR ROW DATA, ACCA NOW 0 IF NO JOYSTICK
0272   A218  26 06                    BNE   LA220                 = BUTTON DOWN. BRANCH IF JOYSTICK BUTTON DOWN
0273   A21A  E6 62             LA21A  LDB   2,S                   GET COLUMN STROBE DATA
0274   A21C  8D 1A                    BSR   LA238                 READ A KEY
0275   A21E  A1 61                    CMPA  1,S                   IS IT THE SAME KEY AS BEFORE DEBOUNCE?
0276   A220  35 12             LA220  PULS  A,X                   REMOVE TEMP SLOTS FROM THE STACK AND RECOVER
0277                           *                                  THE ASCII VALUE OF THE KEY
0278   A222  26 07                    BNE   LA22B                 NOT THE SAME KEY OR JOYSTICK BUTTON
0279   A224  81 12                    CMPA  #$12                  IS SHIFT ZERO DOWN?
0280   A226  26 04                    BNE   LA22C                 NO
0281   A228  73 01 1A                 COM   CASFLG                YES, TOGGLE UPPER CASE/LOWER CASE FLAG
0282   A22B  4F                LA22B  CLRA                       SET ZERO FLAG TO INDICATE NO NEW KEY DOWN
0283   A22C  35 D4             LA22C  PULS  B,X,U,PC              RESTORE REGISTERS
0284
0285                           * TEST FOR THE SHIFT KEY
0286   A22E  86 7F             LA22E  LDA   #$7F                  COLUMN STROBE
0287   A230  A7 42                    STA   2,U                   STORE TO P1A
0288   A232  A6 C4                    LDA   ,U                    READ KEY DATA
0289   A234  43                       COMA                        *
0290   A235  84 40                    ANDA  #$40                  * SET BIT 6 IF SHIFT KEY DOWN
0291   A237  39                       RTS                         RETURN
0292
0293                           * READ THE KEYBOARD
0294   A238  E7 42             LA238  STB   2,U                   SAVE NEW COLUMN STROBE VALUE
0295   A23A  A6 C4             LA23A  LDA   ,U                    READ PIA0, PORT A TO SEE IF KEY IS DOWN
0296                           *                                  A BIT WILL BE ZERO IF ONE IS
0297   A23C  8A 80                    ORA   #$80                  MASK OFF THE JOYSTICK COMPARATOR INPUT
0298   A23E  6D 42                    TST   $02,U                 ARE WE STROBING COLUMN 7?
0299   A240  2B 02                    BMI   LA244                 NO
0300   A242  8A C0                    ORA   #$C0                  YES, FORCE ROW 6 TO BE HIGH - THIS WILL CAUSE
0301                           *                                  THE SHIFT KEY TO BE IGNORED
0302   A244  39                LA244  RTS                         RETURN
0303
0304   A245  C6 33             LA245  LDB   #51                   CODE FOR  AT SIGN
0305   A247  8E A2 38          LA247  LDX   #CONTAB-$36           POINT X TO CONTROL CODE TABLE
0306   A24A  C1 21                    CMPB  #33                   KEY NUMBER <33?
0307   A24C  25 16                    BLO   LA264                 YES (ARROW KEYS, SPACE BAR, ZERO)
0308   A24E  8E A2 1A                 LDX   #CONTAB-$54           POINT X TO MIDDLE OF CONTROL TABLE
0309   A251  C1 30                    CMPB  #48                   KEY NUMBER >48?
0310   A253  24 0F                    BHS   LA264                 YES (ENTER,CLEAR,BREAK,AT SIGN)
0311   A255  8D D7                    BSR   LA22E                 CHECK SHIFT KEY (ACCA WILL CONTAIN STATUS)
0312   A257  C1 2B                    CMPB  #43                   IS KEY A NUMBER, COLON OR SEMICOLON?
0313   A259  23 02                    BLS   LA25D                 YES
0314   A25B  88 40                    EORA  #$40                  TOGGLE BIT 6 OF ACCA WHICH CONTAINS THE SHIFT DATA
0315                           *                                  ONLY FOR SLASH,HYPHEN,PERIOD,COMMA
0316   A25D  4D                LA25D  TSTA                       SHIFT KEY DOWN?
0317   A25E  26 AC                    BNE   LA20C                 YES
0318   A260  CB 10                    ADDB  #$10                  NO, ADD IN ASCII OFFSET CORRECTION
0319   A262  20 A8                    BRA   LA20C                 GO CHECK FOR DEBOUNCE
0320   A264  58                LA264  ASLB                        MULT ACCB BY 2 - THERE ARE 2 ENTRIES IN CONTROL
0321                           *                                  TABLE FOR EACH KEY - ONE SHIFTED, ONE NOT
0322   A265  8D C7                    BSR   LA22E                 CHECK SHIFT KEY
0323   A267  27 01                    BEQ   LA26A                 NOT DOWN
0324   A269  5C                       INCB                        ADD ONE TO GET THE SHIFTED VALUE
0325   A26A  E6 85             LA26A  LDB   B,X                   GET ASCII CODE FROM CONTROL TABLE
0326   A26C  20 9E                    BRA   LA20C                 GO CHECK DEBOUNCE
0327                           *
0328                           *
0329                           * CONTROL TABLE   UNSHIFTED, SHIFTED VALUES
0330   A26E  5E 5F             CONTAB FCB   $5E,$5F                UP ARROW
0331   A270  0A 5B                    FCB   $0A,$5B                DOWN ARROW
0332   A272  08 15                    FCB   $08,$15                RIGHT ARROW
0333   A274  09 5D                    FCB   $09,$5D                LEFT ARROW
0334   A276  20 20                    FCB   $20,$20                SPACE BAR
0335   A278  30 12                    FCB   $30,$12                ZERO
0336   A27A  0D 0D                    FCB   $0D,$0D                ENTER
0337   A27C  0C 5C                    FCB   $0C,$5C                CLEAR
0338   A27E  03 03                    FCB   $03,$03                BREAK
0339   A280  40 13                    FCB   $40,$13                AT SIGN
0340
0341                           * CONSOLE OUT
0342   A282  BD 01 67          PUTCHR JSR   RVEC3                 HOOK INTO RAM
0343   A285  34 04                    PSHS  B                     SAVE ACCB
0344   A287  D6 6F                    LDB   DEVNUM                GET DEVICE NUMBER
0345   A289  5C                       INCB                        SET FLAGS
0346   A28A  35 04                    PULS  B                     RESTORE ACCB
0347   A28C  2B 31                    BMI   LA2BF                 SEND TO LINE PRINTER
0348   A28E  26 7A                    BNE   LA30A                 SEND TO SCREEN
0349                           * SEND TO CASSETTE
0350   A290  34 16                    PSHS  X,B,A                 RESTORE REGISTERS
0351   A292  D6 78                    LDB   FILSTA                GET FILE STATUS
0352   A294  5A                       DECB                        INPUT FILE?
0353   A295  27 0F                    BEQ   LA2A6                 YES
0354   A297  D6 79                    LDB   CINCTR                TEMP CHAR CTR
0355   A299  5C                       INCB                        IS THE BUFFER FULL
0356   A29A  26 02                    BNE   LA29E                 NO
```

```
0357   A29C  8D 0A                  BSR   LA2A8                 YES, WRITE DATA BLOCK TO TAPE
0358   A29E  9E 7A           LA29E   LDX   CINPTR                GET BUFFER POINTER
0359   A2A0  A7 80                   STA   ,X+                   PUT CHAR IN CASSETTE BUFFER
0360   A2A2  9F 7A                   STX   CINPTR                STORE NEW BUFFER POINTER
0361   A2A4  0C 79                   INC   CINCTR                INCR BYTE COUNT
0362   A2A6  35 96           LA2A6   PULS  A,B,X,PC
0363
0364                         * WRITE A BLOCK OF DATA TO TAPE
0365   A2A8  C6 01           LA2A8   LDB   #1                    DATA BLOCK TYPE - NOT A HEADER BLOCK
0366   A2AA  D7 7C           LA2AA   STB   BLKTYP                BLOCK NUMBER
0367   A2AC  8E 01 DA                LDX   #CASBUF               CASSETTE BUFFER
0368   A2AF  9F 7E                   STX   CBUFAD                STARTING ADDRESS
0369   A2B1  D6 79                   LDB   CINCTR                GET NUMBER OF BYTES
0370   A2B3  D7 7D                   STB   BLKLEN                BYTE COUNT
0371   A2B5  34 62                   PSHS  U,Y,A                 SAVE REGISTERS
0372   A2B7  BD A7 E5                JSR   LA7E5                 WRITE A BLOCK ON TAPE
0373   A2BA  35 62                   PULS  A,Y,U                 RESTORE REGISTERS
0374   A2BC  7E A6 50                JMP   LA650                 RESET BUFFER POINTERS
0375
0376                         * SOFTWARE UART TO L1NE PRINTER
0377   A2BF  34 17           LA2BF   PSHS  X,B,A,CC              SAVE REGISTERS AND INTERRUPT STATUS
0378   A2C1  1A 50                   ORCC  #$50                  DISABLE IRQ,FIRQ
0379   A2C3  F6 FF 22        LA2C3   LDB   PIA1+2                GET RS 232 STATUS
0380   A2C6  54                      LSRB                        SHIFT RS 232 STATUS BIT INTO CARRY
0381   A2C7  25 FA                   BCS   LA2C3                 LOOP UNTIL READY
0382   A2C9  8D 30                   BSR   LA2FB                 SET OUTPUT TO MARKING
0383   A2CB  5F                      CLRB                        *
0384   A2CC  8D 2F                   BSR   LA2FD                 * TRANSMIT ONE START BIT
0385   A2CE  C6 08                   LDB   #8                    SEND 8 BITS
0386   A2D0  34 04           LA2D0   PSHS  B                     SAVE BIT COUNTER
0387   A2D2  5F                      CLRB                        CLEAR DA IMAGE I ZEROES TO DA WHEN SENDING RS 232 DATA
0388   A2D3  44                      LSRA                        ROTATE NEXT BIT OF OUTPUT CHARACTER TO CARRY FLAG
0389   A2D4  59                      ROLB                        * ROTATE CARRY FLAG INTO BIT ONE
0390   A2D5  58                      ASLB                        * AND ALL OTHER BITS SET TO ZERO
0391   A2D6  8D 25                   BSR   LA2FD                 TRANSMIT DATA BYTE
0392   A2D8  35 04                   PULS  B                     GET BIT COUNTER
0393   A2DA  5A                      DECB                        SENT ALL 8 BITS?
0394   A2DB  26 F3                   BNE   LA2D0                 NO
0395   A2DD  8D 1C                   BSR   LA2FB                 SEND STOP BIT (ACCB:0)
0396   A2DF  35 03                   PULS  CC,A                  RESTORE OUTPUT CHARACTER & INTERRUPT STATUS
0397   A2E1  81 0D                   CMPA  #CR                   IS IT CARRIAGE RETURN?
0398   A2E3  27 08                   BEQ   LA2ED                 YES
0399   A2E5  0C 9C                   INC   LPTPOS                INCREMENT CHARACTER COUNTER
0400   A2E7  D6 9C                   LDB   LPTPOS                CHECK FOR END OF LINE PRINTER LINE
0401   A2E9  D1 9B                   CMPB  LPTWID                AT END OF LINE PRINTER LINE?
0402   A2EB  25 06                   BLO   LA2F3                 NO
0403   A2ED  0F 9C           LA2ED   CLR   LPTPOS                RESET CHARACTER COUNTER
0404   A2EF  8D 14                   BSR   LA305                 *
0405   A2F1  8D 12                   BSR   LA305                 * DELAY FOR CARRIAGE RETURN
0406   A2F3  F6 FF 22        LA2F3   LDB   PIA1+2                WAIT FOR HANDSHAKE
0407   A2F6  54                      LSRB                        CHECK FOR R5232 STATUS?
0408   A2F7  25 FA                   BCS   LA2F3                 NOT YET READY
0409   A2F9  35 94                   PULS  B,X,PC                RESTORE REGISTERS
0410   A2FB  C6 02           LA2FB   LDB   #2                    SET RS232 OUTPUT HIGH (MARKING)
0411   A2FD  F7 FF 20        LA2FD   STB   DA                    STORE TO THE D/A CONVERTER REGISTER
0412   A300  8D 00                   BSR   LA302                 GO WAIT A WHILE
0413   A302  9E 95           LA302   LDX   LPTBTD                GET BAUD RATE
0414   A304  8C 9E 97                FCB   SKP2                  SKIP NEXT TWO BYTES
0415   A305  9E 97           LA305   LDX   LPTLND                PRINTER CARRIAGE RETURN DELAY
0416   A307  7E A7 D3                JMP   LA7D3                 DELAY ON DECREMENTING X
0417
0418                         * PUT A CHARACTER ON THE SCREEN
0419   A30A  34 16           LA30A   PSHS  X,B,A                 SAVE REGISTERS
0420   A30C  9E 88                   LDX   CURPOS                POINT X TO CURRENT CHARACTER POSITION
0421   A30E  81 08                   CMPA  #BS                   IS IT BACKSPACE?
0422   A310  26 0B                   BNE   LA31D                 NO
0423   A312  8C 04 00                CMPX  #VIDRAM               AT TOP OF SCREEN?
0424   A315  27 46                   BEQ   LA35D                 YES - DO NOT ALLOW BACKSPACE
0425   A317  86 60                   LDA   #$60                  BLANK
0426   A319  A7 82                   STA   ,-X                   PUT IN PREVIOUS POSITION
0427   A31B  20 27                   BRA   LA344                 SAVE NEW CURPOS
0428   A31D  81 0D           LA31D   CMPA  #CR                   ENTER KEY?
0429   A31F  26 0E                   BNE   LA32F                 BRANCH IF NOT
0430   A321  9E 88                   LDX   CURPOS                GET CURRENT CHAR POSITION
0431   A323  86 60           LA323   LDA   #$60                  BLANK
0432   A325  A7 80                   STA   ,X+                   PUT IT ON SCREEN
0433   A327  1F 10                   TFR   X,D                   *
0434   A329  C5 1F                   BITB  #$1F                  * TEST FOR BEGINNING OF NEW LINE
0435   A32B  26 F6                   BNE   LA323                 PUT OUT BLANKS TILL NEW LINE
0436   A32D  20 15                   BRA   LA344                 CHECK FOR SCROLLING
0437   A32F  81 20           LA32F   CMPA  #SPACE                *
0438   A331  25 2A                   BCS   LA35D                 * BRANCH IF CONTROL CHARACTER
0439   A333  4D                      TSTA                        SET FLAGS
0440   A334  2B 0C                   BMI   LA342                 IT IS GRAPHIC CHARACTER
0441   A336  81 40                   CMPA  #$40                  *
0442   A338  25 06                   BCS   LA340                 * BRANCH IF NUMBER OR SPECIAL CHARACTER
0443   A33A  81 60                   CMPA  #$60                  UPPER/LOWER CASE?
0444   A33C  25 04                   BCS   LA342                 BRANCH IF UPPER CASE ALPHA
0445   A33E  84 DF                   ANDA  #$DF                  CLEAR BIT 5, FORCE ASCII LOWER CASE TO BE UPPER CASE
```

```
0446   A340  88 40          LA340    EORA  #$40                    INVERT BIT 6, CHANGE UPPER CASE TO LOWER & VICE VERSA
0447   A342  A7 80          LA342    STA   ,X+                     STORE CHARACTER TO SCREEN
0448   A344  9F 88          LA344    STX   CURPOS                   SAVE CURRENT CHAR POSITION
0449   A346  8C 05 FF                CMPX  #VIDRAM+511              END OF SCREEN BUFFER?
0450   A349  23 12                   BLS   LA35D                    RETURN IF NO NEED TO SCROLL
0451   A34B  8E 04 00                LDX   #VIDRAM                  TOP OF SCREEN
0452
0453                        * SCROLL THE SCREEN
0454   A34E  EC 88 20       LA34E    LDD   32,X                     GET TWO BYTES
0455   A351  ED 81                   STD   ,X++                     MOVE THEM UP ONE ROW
0456   A353  8C 05 E0                CMPX  #VIDRAM+$1E0             AT THE LAST LINE?
0457   A356  25 F6                   BCS   LA34E                    NO
0458   A358  C6 60                   LDB   #$60                     BLANK
0459   A35A  BD A9 2D                JSR   LA92D                    BLANK LAST LINE
0460   A35D  35 96          LA35D    PULS  A,B,X,PC                 RESTORE REGISTERS
0461
0462                        * SET UP TAB FIELD WIDTH, TAB ZONE, CURRENT POSITION
0463                        * AND LINE WIDTH ACCORDING TO THE DEVICE SELECTED
0464   A35F  BD 01 64       LA35F    JSR   RVEC2                    HOOK INTO RAM
0465   A362  34 16                   PSHS  X,B,A                    SAVE REGISTERS
0466   A364  0F 6E                   CLR   PRTDEV                   RESET PRINT DEVICE NUMBER
0467   A366  96 6F                   LDA   DEVNUM                   GET DEVICE NUMBER
0468   A368  27 09                   BEQ   LA373                    BRANCH IF SCREEN
0469   A36A  4C                      INCA                           CHECK FOR CASSETTE
0470   A36B  27 17                   BEQ   LA384                    BRANCH IF CASSETTE
0471                        * END UP HERE IF PRINTER
0472   A36D  9E 99                   LDX   LPTCFW                   TAB FIELD WIDTH AND TAB ZONE
0473   A36F  DC 9B                   LDD   LPTWID                   PRINTER WIDTH AND POSITION
0474   A371  20 09                   BRA   LA37C                    SET PRINT PARAMETERS
0475                        * SCREEN DISPLAY VALUES
0476   A373  D6 89          LA373    LDB   CURPOS+1                 GET CURSOR LOC LS BYTE
0477   A375  C4 1F                   ANDB  #$1F                     KEEP ONLY COLUMN POSITION
0478   A377  8E 10 10                LDX   #$1010                   TAB FIELD WIDTH AND LAST TAB ZONE
0479   A37A  86 20                   LDA   #32                      DISPLAY SCREEN LINE WIDTH
0480   A37C  9F 6A          LA37C    STX   DEVCFW                   SAVE TAB FIELD WIDTH AND ZONE
0481   A37E  D7 6C                   STB   DEVPOS                   SAVE PRINT POSITION
0482   A380  97 6D                   STA   DEVWID                   SAVE PRINT WIDTH
0483   A382  35 96                   PULS  A,B,X,PC                 RESTORE REGISTERS
0484   A384  03 6E          LA384    COM   PRTDEV                   SET TO $FF FOR CASSETTE
0485   A386  8E 01 00                LDX   #$0100                   * TAB FIELD WIDTH = 1; ALL OTHER
0486   A389  4F                      CLRA                           * PARAMETERS = 0
0487   A38A  5F                      CLRB                           *
0488   A38B  20 EF                   BRA   LA37C                    SET PRINT PARAMETERS
0489
0490                        * THIS IS THE ROUTINE THAT GETS AN INPUT LINE FOR BASIC
0491                        * EXIT WITH BREAK KEY: CARRY = 1
0492                        * EXIT WITH ENTER KEY: CARRY = 0
0493   A38D  BD A9 28       LA38D    JSR   LA928                    CLEAR SCREEN
0494   A390  BD 01 82       LA390    JSR   RVEC12                   HOOK INTO RAM
0495   A393  0F 87                   CLR   IKEYIM                   RESET BREAK CHECK KEY TEMP KEY STORAGE
0496   A395  8E 02 DD                LDX   #LINBUF+1                INPUT LINE BUFFER
0497   A398  C6 01                   LDB   #1                       ACCB CHAR COUNTER: SET TO 1 TO ALLOW A
0498                        *                                      BACKSPACE AS FIRST CHARACTER
0499   A39A  BD A1 71       LA39A    JSR   LA171                    GO GET A CHARACTER FROM CONSOLE IN
0500   A39D  0D 70                   TST   CINBFL                   GET CONSOLE IN BUFFER FLAG
0501   A39F  26 2B                   BNE   LA3CC                    BRANCH IF NO MORE CHARACTERS IN INPUT FILE
0502   A3A1  0D 6F                   TST   DEVNUM                   CHECK DEVICE NUMBER
0503   A3A3  26 23                   BNE   LA3C8                    BRANCH IF NOT SCREEN
0504   A3A5  81 0C                   CMPA  #FORMF                   FORM FEED
0505   A3A7  27 E4                   BEQ   LA38D                    YES - CLEAR SCREEN
0506   A3A9  81 08                   CMPA  #BS                      BACKSPACE
0507   A3AB  26 07                   BNE   LA3B4                    NO
0508   A3AD  5A                      DECB                           YES - DECREMENT CHAR COUNTER
0509   A3AE  27 E0                   BEQ   LA390                    BRANCH IF BACK AT START OF LINE AGAIN
0510   A3B0  30 1F                   LEAX  -1,X                     DECREMENT BUFFER POINTER
0511   A3B2  20 34                   BRA   LA3E8                    ECHO CHAR TO SCREEN
0512   A3B4  81 15          LA3B4    CMPA  #$15                     SHIFT RIGHT ARROW?
0513   A3B6  26 0A                   BNE   LA3C2                    NO
0514                        * YES, RESET BUFFER TO BEGINNING AND ERASE CURRENT LINE
0515   A3B8  5A              LA3B8    DECB                           DEC CHAR CTR
0516   A3B9  27 D5                   BEQ   LA390                    GO BACK TO START IF CHAR CTR = 0
0517   A3BB  86 08                   LDA   #BS                      BACKSPACE?
0518   A3BD  BD A2 82                JSR   PUTCHR                   SEND TO CONSOLE OUT (SCREEN)
0519   A3C0  20 F6                   BRA   LA3B8                    KEEP GOING
0520   A3C2  81 03          LA3C2    CMPA  #3                       BREAK KEY?
0521   A3C4  1A 01                   ORCC  #1                       SET CARRY FLAG
0522   A3C6  27 05                   BEQ   LA3CD                    BRANCH IF BREAK KEY DOWN
0523   A3C8  81 0D          LA3C8    CMPA  #CR                      ENTER KEY?
0524   A3CA  26 0D                   BNE   LA3D9                    NO
0525   A3CC  4F              LA3CC    CLRA                           CLEAR CARRY FLAG IF ENTER KEY - END LINE ENTRY
0526   A3CD  34 01           LA3CD    PSHS  CC                       SAVE CARRY FLAG
0527   A3CF  BD B9 58                JSR   LB958                    SEND CR TO SCREEN
0528   A3D2  6F 84                   CLR   ,X                       MAKE LAST BYTE IN INPUT BUFFER = 0
0529   A3D4  8E 02 DC                LDX   #LINBUF                  RESET INPUT BUFFER POINTER
0530   A3D7  35 81                   PULS  CC,PC                    RESTORE CARRY FLAG
0531
0532                        * INSERT A CHARACTER INTO THE BASIC LINE INPUT BUFFER
0533   A3D9  81 20          LA3D9    CMPA  #$20                     IS IT CONTROL CHAR?
0534   A3DB  25 BD                   BLO   LA39A                    BRANCH IF CONTROL CHARACTER
```

```
0535   A3DD  81 7B                    CMPA  #'z-1                 *
0536   A3DF  24 B9                    BCC   LA39A                 * IGNORE IF > LOWER CASE Z
0537   A3E1  C1 FA                    CMPB  #LBUFMX               HAVE 250 OR MORE CHARACTERS BEEN ENTERED?
0538   A3E3  24 B5                    BCC   LA39A                 YES, IGNORE ANY MORE
0539   A3E5  A7 80                    STA   ,X+                   PUT IT IN INPUT BUFFER
0540   A3E7  5C                       INCB                        INCREMENT CHARACTER COUNTER
0541   A3E8  BD A2 82          LA3E8  JSR   PUTCHR                ECHO IT TO SCREEN
0542   A3EB  20 AD                    BRA   LA39A                 GO SET SOME MORE
0543
0544                           * INPUT DEVICE NUMBER CHECK
0545   A3ED  BD 01 6D          LA3ED  JSR   RVEC5                 HOOK INTO RAM
0546   A3F0  96 6F                    LDA   DEVNUM                DEVICE NUMBER
0547   A3F2  27 21                    BEQ   LA415                 RETURN IF SCREEN
0548   A3F4  4C                       INCA                        *
0549   A3F5  26 0C                    BNE   LA403                 * BRANCH IF NOT CASSETTE (BAD FILE MODE)
0550   A3F7  96 78                    LDA   FILSTA                GET FILE STATUS
0551   A3F9  26 05                    BNE   LA400                 FILE IS OPEN
0552   A3FB  C6 2C             LA3FB  LDB   #22*2                  FILE NOT OPEN  ERROR
0553   A3FD  7E AC 46                 JMP   LAC46                 JUMP TO ERROR SERVICING ROUTINE
0554   A400  4A                LA400  DECA                        *
0555   A401  27 12                    BEQ   LA415                 * FILE IS IN INPUT MODE, RETURN
0556   A403  7E A6 16          LA403  JMP   LA616                  BAD FILE MODE  ERROR
0557
0558                           * PRINT DEVICE NUMBER CHECK
0559   A406  BD 01 70          LA406  JSR   RVEC6                 HOOK INTO RAM
0560   A409  96 6F                    LDA   DEVNUM                GET DEVICE NUMBER
0561   A40B  4C                       INCA                        *
0562   A40C  26 07                    BNE   LA415                 * RETURN IF NOT TAPE
0563   A40E  96 78                    LDA   FILSTA                GET FILE STATUS
0564   A410  27 E9                    BEQ   LA3FB                  FILE NOT OPEN  ERROR
0565   A412  4A                       DECA                        *
0566   A413  27 EE                    BEQ   LA403                 *  BAD FILE MODE  - FILE IN INPUT MODE
0567   A415  39                LA415  RTS
0568
0569                           * CLOSE
0570   A416  27 0E             CLOSE  BEQ   LA426                 BRANCH IF NO NAME SPECIFIED
0571   A418  BD A5 A5                 JSR   LA5A5                 CHECK DEVICE NUMBER
0572   A41B  8D 10             LA41B  BSR   LA42D                 GO CLOSE A FILE
0573   A41D  9D A5                    JSR   GETCCH                GET CURRENT BASIC CHARACTER
0574   A41F  27 2A                    BEQ   LA44B                 RETURN IF NO MORE FILES
0575   A421  BD A5 A2                 JSR   LA5A2                 CHECK SYNTAX AND DEVICE NUMBER
0576   A424  20 F5                    BRA   LA41B                 KEEP CLOSING FILES
0577
0578                           * CLOSE ALL FILES HANDLER
0579   A426  BD 01 73          LA426  JSR   RVEC7                 HOOK INTO RAM
0580   A429  86 FF                    LDA   #-1                   CASSETTE DEVICE NUMBER
0581   A42B  97 6F                    STA   DEVNUM                SET DEVICE NUMBER
0582                           * CLOSE FILE HANDLER
0583   A42D  BD 01 76          LA42D  JSR   RVEC8                 HOOK INTO RAM
0584   A430  96 6F                    LDA   DEVNUM                GET DEVICE NUMBER
0585   A432  0F 6F                    CLR   DEVNUM                SET TO SCREEN
0586   A434  4C                       INCA                        *
0587   A435  26 14                    BNE   LA44B                 * BRANCH IF WAS NOT CASSETTE
0588   A437  96 78                    LDA   FILSTA                GET FILE STATUS
0589   A439  81 02                    CMPA  #2                    IS IT OUTPUT MODE
0590   A43B  26 0C                    BNE   LA449                 NO
0591   A43D  96 79                    LDA   CINCTR                GET CHARACTER BUFFER CTR
0592   A43F  27 03                    BEQ   LA444                 WRITE END OF PROG BLOCK IF BUFFER EMPTY
0593   A441  BD A2 A8                 JSR   LA2A8                 WRITE A BLOCK TO TAPE
0594   A444  C6 FF             LA444  LDB   #$FF                  END OF FILE TYPE BLOCK NUMBER
0595   A446  BD A2 AA                 JSR   LA2AA                 WRITE END OF FILE TYPE BLOCK
0596   A449  0F 78             LA449  CLR   FILSTA                CASSETTE FILE STATUS CLOSED
0597   A44B  39                LA44B  RTS
0598
0599                           * CSAVE
0600   A44C  BD A5 78          CSAVE  JSR   LA578                 GO SCAN OFF NAME
0601   A44F  9D A5                    JSR   GETCCH                GET CURRENT CHARACTER IN THE BASIC LINE
0602   A451  27 16                    BEQ   LA469                 BRANCH IF NONE
0603   A453  BD B2 6D                 JSR   LB26D                 SYNTAX ERROR IF NOT COMMA
0604   A456  C6 41                    LDB   #'A                   IS THIS AN ASCII SAVE?
0605   A458  BD B2 6F                 JSR   LB26F                 SYNTAX ERROR IF NOT  A
0606   A45B  26 EE                    BNE   LA44B                 RETURN IF NOT END OF LINE
0607   A45D  4F                       CLRA                        FILE TYPE = 0
0608   A45E  BD A6 5C                 JSR   LA65C                 WRITE OUT HEADER BLOCK
0609   A461  86 FF                    LDA   #-1                   CASSETTE CODE
0610   A463  97 6F                    STA   DEVNUM                SET DEVICE NUMBER TO CASSETTE
0611   A465  4F                       CLRA                        CLEAR CARRY - FORCE LIST TO BEGIN AT PROGRAM START
0612   A466  7E B7 64                 JMP   LIST                  GO DO A  LIST  TO CASSETTE
0613
0614                           * NON-ASCII CSAVE
0615   A469  4F                LA469  CLRA                        FILE TYPE = 0
0616   A46A  9E 8A                    LDX   ZERO                  ZERO OUT ASCII FLAG AND FILE MODE
0617   A46C  BD A6 5F                 JSR   LA65F                 WRITE HEADER BLOCK
0618   A46F  0F 78                    CLR   FILSTA                CLOSE FILES
0619   A471  0C 7C                    INC   BLKTYP                INCREMENT BLOCK NUMBER
0620   A473  BD A7 D8                 JSR   WRLDR                 WRITE 55 S TO CASSETTE
0621   A476  9E 19                    LDX   TXTTAB                ADDRESS OF PROGRAM START
0622   A478  9F 7E             LA478  STX   CBUFAD                STORE CURRENT BLOCK START ADDR
0623   A47A  86 FF                    LDA   #255                  255 BYTE BLOCKS
```

```
0624    A47C  97 7D                    STA   BLKLEN                  BLOCK SIZE
0625    A47E  DC 1B                    LDD   VARTAB                  END OF PROGRAM
0626    A480  93 7E                    SUBD  CBUFAD                  CURRENT BLOCK STARTING ADDR
0627    A482  27 0D                    BEQ   LA491                   BRANCH IF IT CAME OUT EXACT
0628    A484  10 83 00 FF              CMPD  #255                    MORE THAN 255 BYTES LEFT?
0629    A488  24 02                    BHS   LA48C                   YES
0630    A48A  D7 7D                    STB   BLKLEN                  USE ACTUAL BLOCK SIZE IF LESS THAN 255
0631    A48C  BD A7 F4          LA48C  JSR   SNDBLK                  WRITE BLOCK TO CASSETTE
0632    A48F  20 E7                    BRA   LA478                   DO ANOTHER BLOCK
0633    A491  00 7C             LA491  NEG   BLKTYP                  MAKE BLOCK NUMBER NEGATIVE (EOF BLOCK)
0634    A493  0F 7D                    CLR   BLKLEN                  ZERO BLOCK SIZE
0635    A495  7E A7 E7                 JMP   LA7E7                   WRITE A BLOCK, TURN OFF MOTOR
0636
0637                            * CLOAD
0638    A498  0F 78             CLOAD  CLR   FILSTA                  CLOSE FILES
0639    A49A  81 4D                    CMPA  #'M                     IS IT CLOADM?
0640    A49C  27 60                    BEQ   LA4FE                   BRANCH IF SO
0641    A49E  32 62                    LEAS  2,S                     GET RID OF THE RETURN
0642    A4A0  BD A5 C5                 JSR   LA5C5                   GO GET FILE NAME
0643    A4A3  BD A6 48                 JSR   LA648                   SEARCH FOR FILE
0644    A4A6  7D 01 E4                 TST   CASBUF+10               GET FILE MODE (NON-ZERO=DATA OR ASCII)
0645    A4A9  27 1D                    BEQ   LA4C8                   ZERO = CRUNCHED BASIC OR MACHINE LANG
0646    A4AB  B6 01 E3                 LDA   CASBUF+9                GET ASCII FLAG
0647    A4AE  27 1D                    BEQ   LA4CD                    BAD FILE NODE  0 = CRUNCHED OR MACH LANG
0648    A4B0  BD AD 19                 JSR   LAD19                   DO A  NEW
0649    A4B3  86 FF                    LDA   #-1                     TAPE DEVICE NUMBER
0650    A4B5  97 6F                    STA   DEVNUM                  SET DEVICE NUMBER TO TAPE
0651    A4B7  0C 78                    INC   FILSTA                  FILE TYPE = INPUT
0652    A4B9  BD A6 35                 JSR   LA635                   GO LOAD ASCII RECORD
0653    A4BC  7E AC 7C                 JMP   LAC7C                   GO LOAD AND CRUNCH INPUT
0654
0655                            * COME HERE FROM BASIC S DIRECT LOOP IF CONSOLE
0656                            * IN BUFFER EMPTY
0657    A4BF  BD 01 85          LA4BF  JSR   RVEC13                  HOOK INTO RAM
0658    A4C2  BD A4 2D                 JSR   LA42D                   CLOSE ACTIVE FILE
0659    A4C5  7E AC 73                 JMP   LAC73                   GO TO BASIC S DIRECT LOOP
0660
0661                            * CLOAD A CRUNCHED BASIC
0662    A4C8  B6 01 E2          LA4C8  LDA   CASBUF+8                FILE TYPE
0663    A4CB  27 03                    BEQ   LA4D0                   ZERO IS CSAVE TYPE
0664    A4CD  7E A6 16          LA4CD  JMP   LA616                    BAD FILE MODE  IF NOT BASIC FILE
0665    A4D0  BD AD 19          LA4D0  JSR   LAD19                   DO A  NEW
0666    A4D3  BD A7 7C                 JSR   CASON                   TURN ON TAPE, START READING
0667    A4D6  9E 19                    LDX   TXTTAB                  GET START OF PROGRAM ADDRESS
0668    A4D8  9F 7E             LA4D8  STX   CBUFAD                  STORE IT IN LOAD BUFFER
0669    A4DA  DC 7E                    LDD   CBUFAD                  GET START ADDRESS TO D REG
0670    A4DC  4C                       INCA                          ADD 256 TO LOAD ADDRESS
0671    A4DD  BD AC 37                 JSR   LAC37                   SEE IF ROOM BELOW STACK FOR ONE BLOCK
0672    A4E0  BD A7 0B                 JSR   GETBLK                  READ A BLOCK
0673    A4E3  26 13                    BNE   LA4F8                   GOT AN ERROR DURING READ
0674    A4E5  96 7C                    LDA   BLKTYP                  BLOCK NUMBER
0675    A4E7  27 0F                    BEQ   LA4F8                   I/O ERROR IF HEADER BLOCK TYPE
0676    A4E9  2A ED                    BPL   LA4D8                   REAR MORE IF BLOCK NUMBER POSITIVE
0677    A4EB  9F 1B                    STX   VARTAB                  SET END OF PROGRAM ADDRESS
0678    A4ED  8D 4C                    BSR   LA53B                   TURN OFF TAPE DECK
0679    A4EF  8E AB EC                 LDX   #LABED-1                POINT TO  OK  MESSAGE
0680    A4F2  BD B9 9C                 JSR   LB99C                   PRINT  OK  TO CONSOLE OUT
0681    A4F5  7E AC E9                 JMP   LACE9                   RESET INPUT POINTER, CLEAR VARIABLES AND
0682                            *                                     RETURN TO MAIN LOOP OF BASIC
0683
0684    A4F8  BD AD 19          LA4F8  JSR   LAD19                   DO A  NEW
0685    A4FB  7E A6 19          LA4FB  JMP   LA619                    I/O ERROR
0686
0687                            * CLOADM
0688    A4FE  9D 9F             LA4FE  JSR   GETNCH                  GET NEXT CHARACTER IN BASIC LINE
0689    A500  8D 76                    BSR   LA578                   GO SCAN OFF NAME
0690    A502  BD A6 48                 JSR   LA648                   SEARCH FOR FILE
0691    A505  9E 8A                    LDX   ZERO                    STORE ZERO TO X REG, DEFAULT OFFSET VALUE
0692    A507  9D A5                    JSR   GETCCH                  CHECK FOR AN OFFSET
0693    A509  27 06                    BEQ   LA511                   BRANCH IF NO OFFSET
0694    A50B  BD B2 6D                 JSR   LB26D                   SYNTAX CHECK FOR COMMA
0695    A50E  BD B7 3D                 JSR   LB73D                   EVALUATE OFFSET; RETURN VALUE IN X
0696    A511  B6 01 E2          LA511  LDA   CASBUF+8                CHECK FILE MODE
0697    A514  81 02                    CMPA  #2                      IS IT MACHINE LANGUAGE?
0698    A516  26 B5                    BNE   LA4CD                    BAD FILE MODE  ERROR IF NOT
0699    A518  FC 01 E5                 LDD   CASBUF+11               GET TRANSFER ADDR FROM TAPE
0700    A51B  33 8B                    LEAU  D,X                     ADD OFFSET
0701    A51D  DF 9D                    STU   EXECJP                  STORE TRANSFER ADDR IN EXEC ARGUMENT
0702    A51F  7D 01 E4                 TST   CASBUF+10               CHECK FILE MODE
0703    A522  26 A9                    BNE   LA4CD                    BAD FILE MODE  ERROR
0704    A524  FC 01 E7                 LDD   CASBUF+13               GET LOAD ADDR FROM TAPE
0705    A527  30 8B                    LEAX  D,X                     ADD OFFSET
0706    A529  9F 7E                    STX   CBUFAD                  STORE IN BUFFER START ADDRESS POINTER
0707    A52B  BD A7 7C                 JSR   CASON                   START UP TAPE
0708    A52E  BD A7 0B          LA52E  JSR   GETBLK                  READ A BLOCK
0709    A531  26 C8                    BNE   LA4FB                   BRANCH IF I/O ERROR
0710    A533  9F 7E                    STX   CBUFAD                  STORE NEW START ADDR (ONE BLOCK HIGHER)
0711    A535  0D 7C                    TST   BLKTYP                  CHECK BLOCK NUMBER
0712    A537  27 C2                    BEQ   LA4FB                   BRANCH IF I/O ERROR (HEADER BLOCK)
```

```
0713   A539  2A F3                    BPL   LA52E                  GO READ SOME MORE
0714   A53B  7E A7 E9          LA53B    JMP   LA7E9                  GO TURN OFF TAPE DECK
0715
0716                           * EXEC
0717   A53E  27 05             EXEC     BEQ   LA545                  BRANCH IF NO ARGUMENT
0718   A540  BD B7 3D                   JSR   LB73D                  EVALUATE ARGUMENT - ARGUMENT RETURNED IN X
0719   A543  9F 9D                      STX   EXECJP                 STORE X TO EXEC JUMP ADDRESS
0720   A545  6E 9F 00 9D       LA545    JMP   [EXECJP]               GO DO IT
0721
0722                           * BREAK CHECK
0723   A549  BD 01 7F          LA549    JSR   RVEC11                 HOOK INTO RAM
0724   A54C  96 6F                      LDA   DEVNUM                 GET DEVICE NUMBER
0725   A54E  4C                         INCA                         CHECK FOR TAPE
0726   A54F  27 50                      BEQ   LA5A1                  RETURN IF TAPE
0727   A551  7E AD EB                   JMP   LADEB                  GO DO BREAK KEY CHECK
0728
0729                           * THIS ROUTINE EVALUATES AN ARGUMENT
0730                           * AND MAKES SURE IT IS WITHIN LIMITS OF VIDEO DISPLAY RAM
0731   A554  BD B3 E4          LA554    JSR   LB3E4                  EVALUATE EXPRESSION AND RETURN VALUE IN ACCD
0732   A557  83 01 FF                   SUBD  #511                   ONLY 512 VIDEO DISPLAY LOCATIONS
0733   A55A  10 22 0E EC                LBHI  LB44A                  BRANCH IF > 511 TO  ILLEGAL FUNCTION CALL
0734   A55E  C3 05 FF                   ADDD  #VIDRAM+511            ADD BACK IN OFFSET + START OF VIDEO RAM
0735   A561  DD 88                      STD   CURPOS                 PUT THE CURSOR THERE
0736   A563  39                         RTS
0737
0738                           * INKEY$
0739   A564  96 87             INKEY    LDA   IKEYIM                 WAS A KEY DOWN IN THE BREAK CHECK?
0740   A566  26 03                      BNE   LA56B                  YES
0741   A568  BD A1 CB                   JSR   KEYIN                  GO GET A KEY
0742   A56B  0F 87             LA56B    CLR   IKEYIM                 CLEAR INKEY RAM IMAGE
0743   A56D  97 53                      STA   FPA0+3                 STORE THE KEY IN FPA0
0744   A56F  10 26 11 1C               LBNE  LB68F                  CONVERT FPA0+3 TO A STRING
0745   A573  97 56                      STA   STRDES                 SET LENGTH OF STRING = 0 IF NO KEY DOWN
0746   A575  7E B6 9B                   JMP   LB69B                  PUT A NULL STRING ONTO THE STRING STACK
0747
0748                           * STRIP A FILENAME OFF OF THE BASIC INPUT LINE
0749   A578  8E 01 D1          LA578    LDX   #CFNBUF                POINT TO FILE NAME BUFFER
0750   A57B  6F 80                      CLR   ,X+                    CLEAR THE FIRST BYTE - IT WILL CONTAIN THE COUNT
0751                           *                                     OF THE NUMBER OF CHARACTERS IN THE NAME
0752   A57D  86 20                      LDA   #SPACE                 SPACE
0753   A57F  A7 80             LA57F    STA   ,X+                    BLANK FILL 8 CHARS
0754   A581  8C 01 DA                   CMPX  #CASBUF                DONE?
0755   A584  26 F9                      BNE   LA57F                  NO
0756   A586  9D A5                      JSR   GETCCH                 GET CURRENT INPUT CHAR
0757   A588  27 17                      BEQ   LA5A1                  RETURN IF NO NAME
0758   A58A  BD B1 56                   JSR   LB156                  GET THE FILE NAME - EVALUATE EXPRESSION
0759   A58D  BD B6 54                   JSR   LB654                  POINT X TO START OF NAME (TOP STRING ON STRING STACK)
0760   A590  CE 01 D1                   LDU   #CFNBUF                CASSETTE FILE NAME BUFFER
0761   A593  E7 C0                      STB   ,U+                    STORE THE NUMBER OF BYTES IN THE NAME
0762   A595  27 0A                      BEQ   LA5A1                  NULL NAME (BLANK NAME)
0763   A597  8C                         FCB   SKP2                   SKIP THE NEXT TWO BYTES
0764   A597  C6 08             LA598    LDB   #8                     MOVE 8 BYTES
0765
0766                           * MOVE ACCB BYTES FROM (X) TO (U)
0767   A59A  A6 80             LA59A    LDA   ,X+                    GET BYTE FROM X
0768   A59C  A7 C0                      STA   ,U+                    STORE IT AT U
0769   A59E  5A                         DECB                         MOVED ALL BYTES?
0770   A59F  26 F9                      BNE   LA59A                  NO
0771   A5A1  39                LA5A1    RTS
0772
0773                           * GET DEVICE NUMBER FROM BASIC LINE - CHECK VALIDITY
0774   A5A2  BD B2 6D          LA5A2    JSR   LB26D                  CHECK FOR COMMA, SYNTAX ERROR IF NONE
0775   A5A5  81 23             LA5A5    CMPA  #'#                    IS NEXT CHARACTER A NUMBER?
0776   A5A7  26 02                      BNE   LA5AB                  NO
0777   A5A9  9D 9F                      JSR   GETNCH                 GET NEXT BASIC INPUT CHARACTER
0778   A5AB  BD B1 41          LA5AB    JSR   LB141                  EVALUATE EXPRESSION
0779   A5AE  BD B3 ED          LA5AE    JSR   INTCNV                 CONVERT FPA0 TO INTEGER, RETURN VALUE IN ACCD
0780   A5B1  59                         ROLB                         MSB OF ACCB TO CARRY
0781   A5B2  89 00                      ADCA  #0                     ADD MSB OF ACCB TO ACCA
0782   A5B4  26 69                      BNE   LA61F                   DEVICE # ERROR  IF ACCA<FF80 OR >007F
0783   A5B6  56                         RORB                         RESTORE ACCB
0784   A5B7  D7 6F                      STB   DEVNUM                 STORE B IN DEVICE NUMBER
0785   A5B9  BD 01 61                   JSR   RVEC1                  HOOK INTO RAM
0786   A5BC  27 06                      BEQ   LA5C4                  BRANCH IF DEVICE NUMBER SET TO SCREEN
0787   A5BE  2A 5F                      BPL   LA61F                   DEVICE NUMBER  ERROR IF POSITIVE DEVICE NUMBER
0788   A5C0  C1 FE                      CMPB  #-2                    LOWEST LEGAL DEVICE NUMBER
0789   A5C2  2D 5B                      BLT   LA61F                   DEVICE NUMBER ERROR
0790   A5C4  39                LA5C4    RTS
0791
0792                           ** THIS ROUTINE WILL SCAN OFF THE FILE NAME FROM A BASIC LINE
0793                           ** AND RETURN A SYNTAX ERROR IF THERE ARE ANY CHARACTERS
0794                           ** FOLLOWING THE END OF THE NAME
0795   A5C5  8D B1             LA5C5    BSR   LA578                  SCAN OFF NAME
0796   A5C7  9D A5                      JSR   GETCCH                 GET CURRENT INPUT CHAR FROM BASIC LINE
0797   A5C9  27 F9             LA5C9    BEQ   LA5C4                  RETURN IF END OF LINE
0798   A5CB  7E B2 77                   JMP   LB277                  SYNTAX ERROR IF ANY MORE CHARACTERS
0799
0800                           * EOF
0801   A5CE  BD 01 88          EOF      JSR   RVEC14                 HOOK INTO RAM
```

```
0802   A5D1  96 6F          LDA   DEVNUM          GET DEVICE NUMBER
0803   A5D3  34 02          PSHS  A               SAVE IT
0804   A5D5  8D D7          BSR   LA5AE           CHECK DEVICE NUMBER
0805   A5D7  BD A3 ED       JSR   LA3ED           CHECK FOR PROPER FILE AND MODE
0806   A5DA  5F             CLRB                  NOT EOF FLAG = 0
0807   A5DB  96 6F          LDA   DEVNUM          TEST DEVICE NUMBER
0808   A5DD  27 05          BEQ   LA5E4           BRANCH IF NOT SET TO DISPLAY
0809   A5DF  0D 79          TST   CINCTR          ANY CHARACTERS LEFT TO SEND?
0810   A5E1  26 01          BNE   LA5E4           YES
0811   A5E3  53             COMB                  NO - EOF: SET FLAG = -1 ($FF)
0812   A5E4  35 02    LA5E4 PULS  A               GET DEVICE NUMBER BACK AGAIN
0813   A5E6  97 6F          STA   DEVNUM          RESTORE IT
0814   A5E8  1D       LA5E8 SEX                   CONVERT ACCB TO 2 DIGIT SIGNED INTEGER
0815   A5E9  7E B4 F4       JMP   GIVABF          CONVERT ACCD TO FLOATING POINT
0816
0817                       * SKIPF
0818   A5EC  8D D7    SKIPF BSR   LA5C5           SCAN OFF THE BASIC FILE NAME
0819   A5EE  8D 58          BSR   LA648           LOOK FOR THAT FILE ON TAPE
0820   A5F0  BD A6 D1       JSR   LA6D1           READ THE FILE
0821   A5F3  26 24          BNE   LA619            I/O ERROR
0822   A5F5  39             RTS
0823
0824                       * OPEN
0825   A5F6  BD 01 5E OPEN  JSR   RVEC0           HOOK INTO RAM
0826   A5F9  BD B1 56       JSR   LB156           GET FILE STATUS (INPUT,OUTPUT)
0827   A5FC  BD B6 A4       JSR   LB6A4           GET FIRST BYTE OF STATUS STRING TO ACCB
0828   A5FF  34 04          PSHS  B               SAVE IT ON STACK
0829   A601  8D 9F          BSR   LA5A2           CHECK FOR SYNTAX AND GET DEVICE NUMBER
0830   A603  BD B2 6D       JSR   LB26D           SYNTAX CHECK FOR COMMA, SYNTAX ERROR IF NOT
0831   A606  8D BD          BSR   LA5C5           GET FILE NAME
0832   A608  96 6F          LDA   DEVNUM          GET DEVICE NUMBER
0833   A60A  0F 6F          CLR   DEVNUM          SET DEVICE NUMBER TO SCREEN
0834   A60C  35 04          PULS  B               GET STATUS AGAIN
0835   A60E  C1 49          CMPB  #'I             IS IT INPUT MODE?
0836   A610  27 12          BEQ   LA624           YES
0837   A612  C1 4F          CMPB  #'O             IS IT OUTPUT MODE?
0838   A614  27 42          BEQ   LA658           YES
0839
0840                       * IF IT ISN T INPUT OR OUTPUT, BAD FILE MODE
0841   A616  C6 2A    LA616 LDB   #21*2           ERROR # 21  BAD FILE MODE
0842   A618  8C             FCB   SKP2            SKIP TWO BYTES
0843   A619  C6 28    LA619 LDB   #20*2           ERROR # 20  I/O ERROR
0844   A61B  8C             FCB   SPK2            SKIP TWO BYTES
0845   A61C  C6 24    LA61C LDB   #18*2           ERROR # 18  FILE ALREADY OPEN
0846   A61E  8C             FCB   SKP2            SKIP TWO BYTES
0847   A61F  C6 26    LA61F LDB   #19*2           ERROR # 19  DEVICE NUMBER ERROR
0848   A621  7E AC 46       JMP   LAC46           JUMP TO ERROR HANDLER
0849                       *
0850   A624  4C       LA624 INCA                  DEVICE NUMBER SET TO TAPE?
0851   A625  2B EF          BMI   LA616            BAD FILE MODE  IF DEVNUM = NEG BUT NOT CASSETTE
0852   A627  26 2E          BNE   LA657           RETURN IF DEVNUM WAS SET TO SCREEN OR DISK
0853                       * SET TO TAPE
0854   A629  8D 1D          BSR   LA648           GET HEADER BLOCK
0855   A62B  B6 01 E3       LDA   CASBUF+9        GET ASCII FLAG
0856   A62E  B4 01 E4       ANDA  CASBUF+10        AND  IT WITH FILE MODE
0857   A631  27 E3          BEQ   LA616            BAD FILE MODE  - CRUNCHED F1LE OR MACH LANG
0858   A633  0C 78          INC   FILSTA          OPEN FILE FOR INPUT
0859   A635  BD A7 01 LA635 JSR   LA701           START TAPE, READ A BLOCK
0860   A638  26 DF          BNE   LA619            I/O ERROR
0861   A63A  0D 7C          TST   BLKTYP          CHECK BLOCK NUMBER
0862   A63C  27 DB          BEQ   LA619            I/O ERROR  IF HEADER BLOCK
0863   A63E  2B 17          BMI   LA657           BRANCH IF THIS IS THE LAST BLOCK
0864   A640  96 7D          LDA   BLKLEN          CHAR COUNT
0865   A642  27 F1          BEQ   LA635           READ ANOTHER BLOCK IF NULL BLOCK
0866   A644  97 79    LA644 STA   CINCTR          STORE IN TEMP CHARACTER COUNTER
0867   A646  20 0A          BRA   LA652           RESET BUFFER POINTER
0868
0869                       * SEARCH FOR FILE NAME IN CNMBUF
0870   A648  0D 78    LA648 TST   FILSTA          IS THE FILE OPEN?
0871   A64A  26 D0          BNE   LA61C           YES-  FILE ALREADY OPEN
0872   A64C  8D 33          BSR   LA681           SEARCH FOR CORRECT FILE NAME
0873   A64E  26 C9          BNE   LA619            I/O ERROR
0874   A650  0F 79    LA650 CLR   CINCTR          CLEAR CHARACTER COUNTER
0875   A652  8E 01 DA LA652 LDX   #CASBUF         CASSETTE INPUT BUFFER ADDRESS
0876   A655  9F 7A          STX   CINPTR          RESET IT
0877   A657  39       LA657 RTS
0878
0879                       * WRITE OUT THE HEADER BLOCK
0880
0881                       ** CASBUF                          FILE NAME
0882                       ** CASBUF+8                        FILE TYPE
0883                       ** CASBUF+9                        ASCII FLAG
0884                       ** CASBUF+10                       FILE MODE
0885                       ** CASBUF+11,12                    TRANSFER ADDRESS
0886                       ** CASBUF+13,14                    START ADDRESS
0887
0888                       * ENTER HERE FOR DATA FILES W/DEVICE NUMBER IN ACCA
0889   A658  4C       LA658 INCA                  CHECK FOR CASSETTE DEVICE NUMBER
0890   A659  26 FC          BNE   LA657           RETURN IF DEVICE NUMBER WASN T TAPE
```

```
0891   A65B  4C                      INCA                          MAKE FILE TYPE = 1
0892                           * ENTER HERE FOR ASCII FILES
0893   A65C  8E FF FF          LA65C LDX   #$FFFF                  SET ASCII FLAG AND MODE = $FF
0894   A65F  0D 78             LA65F TST   FILSTA                  IS FILE OPEN?
0895   A661  26 B9                   BNE   LA61C                   YES-  FILE ALREADY OPEN
0896   A663  CE 01 DA                LDU   #CASBUF                 CASSETTE INPUT BUFFER
0897   A666  DF 7E                   STU   CBUFAD                  STORE IN STARTING ADDRESS
0898   A668  A7 48                   STA   8,U                     FILE TYPE IN CASBUF+8
0899   A66A  AF 49                   STX   9,U                     ASCII FLAG & MODE IN CASBUF+9, CASBUF+10
0900                           *      CASBUF     +8      +9    +10
0901                           *                                     TYPE   ASCII   MODE
0902                           * BASIC CRUNCHED                 00      00     00
0903                           * BASIC ASCII                          00      FF     FF
0904                           * DATA                                    01      FF     FF
0905                           * MACHINE LANGUAGE        02      00     00
0906                           * MACHINE BLK LOAD         02      00     FF
0907
0908   A66C  8E 01 D2                LDX   #CFNBUF+1               POINT X TO FILE NAME BUFFER
0909   A66F  BD A5 98                JSR   LA598                   MOVE 8 BYTES FROM (X) TO (U)
0910   A672  0F 7C                   CLR   BLKTYP                  ZERO BLOCK NUMBER
0911   A674  86 0F                   LDA   #15                     15 BYTES IN THE HEADER BLOCK
0912   A676  97 7D                   STA   BLKLEN                  CHAR COUNT
0913   A678  BD A7 E5                JSR   LA7E5                   GO WRITE ONE BLOCK
0914   A67B  86 02                   LDA   #2                      OUTPUT FILE
0915   A67D  97 78                   STA   FILSTA                  STORE IN FILE MODE
0916   A67F  20 CF                   BRA   LA650                   RESET POINTERS
0917
0918                           * SEARCH FOR CORRECT CASSETTE FILE NAME
0919   A681  8E 01 DA          LA681 LDX   #CASBUF                 CASSETTE BUFFER
0920   A684  9F 7E                   STX   CBUFAD                  LOAD ADDRESS POINTER
0921   A686  96 68              LA686 LDA   CURLIN                  GET CURRENT LINE NUMBER MSB (CURLIN)
0922   A688  4C                      INCA                          IN DIRECT MODE IF ACCA = $FF
0923   A689  26 0B                   BNE   LA696                   BRANCH IF NOT DIRECT MODE
0924   A68B  BD A9 28                JSR   LA928                   CLEAR SCREEN
0925   A68E  9E 88                   LDX   CURPOS                  CURRENT SCREEN CHAR POSITION
0926   A690  C6 53                   LDB   #'S                      S  MEANS SEARCHING
0927   A692  E7 81                   STB   ,X++                    PUT AN  S  ON THE SCREEN
0928   A694  9F 88                   STX   CURPOS                  STORE NEW CURSOR LOCATION
0929   A696  8D 69              LA696 BSR   LA701                   READ ONE BLOCK FROM TAPE
0930   A698  DA 7C                   ORB   BLKTYP                   OR  ERROR FLAG WITH BLOCK NUMBER
0931   A69A  26 34                   BNE   LA6D0                   BRANCH IF NOT BLOCK ZERO OR ERROR
0932   A69C  8E 01 DA                LDX   #CASBUF                 POINT TO CASSETTE BUFFER
0933   A69F  CE 01 D2                LDU   #CFNBUF+1               POINT TO DESIRED NAME
0934   A6A2  C6 08                   LDB   #8                      EIGHT CHARACTERS MAX IN NAME
0935   A6A4  6F E2                   CLR   ,-S                     ZERO A BYTE ON THE STACK
0936   A6A6  A6 80              LA6A6 LDA   ,X+                     GET CHAR FROM CASSETTE BLOCK
0937   A6A8  10 9E 68                LDY   CURLIN                  GET CURLIN
0938   A6AB  31 21                   LEAY  1,Y                     DIRECT MODE?
0939   A6AD  26 05                   BNE   LA6B4                   FALL THROUGH IF DIRECT MODE
0940   A6AF  0F 6F                   CLR   DEVNUM                  SET DEVICE NUMBER TO SCREEN
0941   A6B1  BD A2 82                JSR   PUTCHR                  OUTPUT A CHAR
0942   A6B4  A0 C0              LA6B4 SUBA  ,U+                     SUBTRACT A CHAR FROM DESIRED NAME
0943                                                                NON-ZERO RESULT IF NO MATCH
0944   A6B6  AA E4                   ORA   ,S                       OR  WITH TOP OF STACK, RESULT WILL BE NON-ZERO IF MISMATCH
0945   A6B8  A7 E4                   STA   ,S                      SAVE IT
0946   A6BA  5A                      DECB                          DONE ALL 8 CHARACTERS?
0947   A6BB  26 E9                   BNE   LA6A6                   NO
0948   A6BD  A6 E0                   LDA   ,S+                     SEE IF ALL CHARS WERE OK
0949   A6BF  27 0A                   BEQ   LA6CB                   BRANCH IF GOOD COMPARE
0950   A6C1  6D 57                   TST   -9,U                    CHECK THE NUMBER OF CHARACTERS IN THE CLOAD STATEMENT
0951   A6C3  27 06                   BEQ   LA6CB                   IF NO NAME SPECIFIED, ANY FILE IS OK
0952                           * DIDN'T FIND THE RIGHT FILE IF HERE
0953   A6C5  8D 0A                   BSR   LA6D1                   LOOK FOR FILE
0954   A6C7  26 07                   BNE   LA6D0                   RETURN IF ERROR
0955   A6C9  20 BB                   BRA   LA686                   GO LOOK SOME MORE
0956   A6CB  86 46              LA6CB LDA   #'F                     *
0957   A6CD  8D 29                   BSR   LA6F8                   * PUT  F  ON THE SCREEN IF DIRECT MODE
0958   A6CF  4F                      CLRA                          SET ZERO FLAG TO INDICATE NO ERRORS
0959   A6D0  39                 LA6D0 RTS
0960   A6D1  7D 01 E4           LA6D1 TST   CASBUF+10               CHECK FILE MODE
0961   A6D4  26 09                   BNE   LA6DF                   BRANCH IF ASCII OR DATA
0962   A6D6  BD A7 7C                JSR   CASON                   TURN ON TAPE DECK
0963   A6D9  8D 30              LA6D9 BSR   GETBLK                  LOAD A BLOCK FROM TAPE
0964   A6DB  8D 08                   BSR   LA6E5                   CHECK FOR ERROR OR LAST BLOCK
0965   A6DD  20 FA                   BRA   LA6D9                   KEEP GOING
0966   A6DF  8D 20              LA6DF BSR   LA701                   READ ONE BLOCK FROM TAPE
0967   A6E1  8D 02                   BSR   LA6E5                   CHECK FOR ERROR OR LAST BLOCK
0968   A6E3  20 FA                   BRA   LA6DF                   KEEP READING BLOCKS
0969   A6E5  26 06              LA6E5 BNE   LA6ED                   GOT AN ERROR ON READING IN BLOCK
0970   A6E7  96 7C                   LDA   BLKTYP                  GET BLOCK NUMBER
0971   A6E9  40                      NEGA                          CHECK FOR LAST BLOCK
0972   A6EA  2B 14                   BMI   LA700                   RETURN IF NOT AN END OF PROGRAM BLOCK
0973   A6EC  4A                      DECA                          IF BLOCK NUMBER WAS $FF, ACCA IS NOW ZERO - THIS WILL
0974                           *                                    CAUSE CLOAD TO IGNORE ERRORS IN THE
0975                           *                                    BLOCKS WHICH IT IS SKIPPING WHILE
0976                           *                                    LOOKING FOR THE CORRECT FILE NAME.
0977   A6ED  97 81              LA6ED STA   CSRERR                  STORE ACCA TO ERROR FLAG
0978   A6EF  32 62                   LEAS  2,S                     REMOVE RETURN ADDRESS FROM STACK
0979   A6F1  20 12                   BRA   LA705                   TURN OFF MOTOR
```

```
0980   A6F3  B6 04 00        LA6F3    LDA   VIDRAM          GET FIRST CHAR ON SCREEN
0981   A6F6  88 40                    EORA  #$40            REVERSE THE VIDEO
0982   A6F8  D6 68              LA6F8  LDB   CURLIN          GET CURLIN MSB
0983   A6FA  5C                       INCB                  CHECK FOR DIRECT MODE
0984   A6FB  26 03                    BNE   LA700           BRANCH IF NOT DIRECT MODE
0985   A6FD  B7 04 00                 STA   VIDRAM          PUT IT ON SCREEN
0986   A700  39                LA700  RTS
0987
0988                           * READ A BLOCK FROM CASSETTE
0989   A701  8D 79              LA701  BSR   CASON           START TAPE, AND LOOK FOR A BUNCH OF $55 OR $AA BYTES
0990   A703  8D 06                    BSR   GETBLK          READ A BLOCK
0991   A705  BD A7 E9           LA705  JSR   LA7E9           TURN OFF MOTOR
0992   A708  D6 81                    LDB   CSRERR          GET ERROR STATUS
0993   A70A  39                       RTS
0994   A70B  1A 50              GETBLK ORCC  #$50            DISABLE IRQ,FIRQ
0995   A70D  8D E4                    BSR   LA6F3           REVERSE VIDEO UPPER LEFT CHAR IF DIRECT MODE
0996   A70F  9E 7E                    LDX   CBUFAD          GET LOAD ADDRESS
0997   A711  4F                       CLRA                  RESET ACCA
0998   A712  8D 41              LA712  BSR   LA755           READ A BIT FROM TAPE, RETURN IT IN CARRY FLAG
0999   A714  46                       RORA                  PUT BIT IN MSB OF ACCA
1000   A715  81 3C                    CMPA  #$3C            GET SYNC ED ON $3C
1001   A717  26 F9                    BNE   LA712           NOT SYNC ED YET
1002   A719  8D 2E                    BSR   LA749           GET BLOCK NUMBER
1003   A71B  97 7C                    STA   BLKTYP          SAVE IT
1004   A71D  8D 2A                    BSR   LA749           GET CHAR COUNT
1005   A71F  97 7D                    STA   BLKLEN          SAVE IT
1006   A721  9B 7C                    ADDA  BLKTYP          ACCUMULATE CHECKSUM
1007   A723  97 80                    STA   CCKSUM          SAVE IT
1008   A725  96 7D                    LDA   BLKLEN          GET BACK CHAR COUNT
1009   A727  97 81                    STA   CSRERR          TEMP SAVE
1010   A729  27 10                    BEQ   LA73B           NULL SET OF CHARACTERS
1011   A72B  8D 1C              LA72B  BSR   LA749           GET BYTE FROM TAPE
1012   A72D  A7 84                    STA   ,X              FILL MEMORY WITH TAPE DATA
1013   A72F  A1 80                    CMPA  ,X+             SEE IF WE READ BACK SAME THING
1014   A731  26 11                    BNE   LA744           BRANCH IF NOT PUTTING IT IN RAM
1015   A733  9B 80                    ADDA  CCKSUM          ACCUMULATE CHECKSUM
1016   A735  97 80                    STA   CCKSUM          TEMP STORE CHECKSUM
1017   A737  0A 81                    DEC   CSRERR          DECR TEMP CHAR COUNT
1018   A739  26 F0                    BNE   LA72B           GET ANOTHER CHARACTER
1019   A73B  8D 0C              LA73B  BSR   LA749           GET CHECKSUM FROM TAPE
1020   A73D  90 80                    SUBA  CCKSUM          COMPARE TO CALCULATED CHECKSUM
1021   A73F  27 05                    BEQ   LA746           BRANCH IF OK
1022   A741  86 01                    LDA   #1              CHECKSUM ERROR FLAG
1023   A743  8C                       FCB   SKP2            SKIP TWO BYTES
1024   A744  86 02              LA744  LDA   #2              NON-RAM ERROR FLAG
1025   A746  97 81              LA746  STA   CSRERR          1 IF CHECKSUM ERROR, 2 IF LOADING INTO NON-RAM
1026   A748  39                       RTS
1027
1028                           * GET A BYTE FROM TAPE
1029   A749  86 08              LA749  LDA   #8              8 BITS/BYTE
1030   A74B  97 82                    STA   CPULWD          TEMP COUNTER
1031   A74D  8D 06              LA74D  BSR   LA755           READ A BIT FROM TAPE
1032   A74F  46                       RORA                  PUT IT INTO ACCA
1033   A750  0A 82                    DEC   CPULWD          GOT ALL 8 BITS
1034   A752  26 F9                    BNE   LA74D           NO
1035   A754  39                       RTS
1036
1037                           * READ A BIT FROM THE TAPE
1038   A755  8D 06              LA755  BSR   LA75D           GET THE TIME BETWEEN TRANSITIONS
1039   A757  D6 83                    LDB   CPERTM          * GET PERIOD TIMER
1040   A759  5A                       DECB                  *
1041   A75A  D1 8F                    CMPB  CMPMID          CMPMID CONTAINS 18 INITIALLY, AND IS USED TO DETERMINE
1042                           *                             WHETHER THE BIT READ IS A ONE OR ZERO
1043                           *                             IF THE PERIOD TIMER IS < 18, THE BIT
1044                           *                             IS CONSIDERED TO BE A ONE, IF > 18, IT IS ZERO
1045   A75C  39                       RTS
1046
1047                           * MAIN TIMING LOOP
1048   A75D  0F 83              LA75D  CLR   CPERTM          RESET PERIOD TIMER
1049   A75F  0D 84                    TST   CBTPHA          CHECK TO SEE IF SYNC ED ON THE HI-LO TRANSITION OR LO-HI
1050   A761  26 10                    BNE   LA773           BRANCH ON HI-LO TRANSITION
1051                           * LO - HI TRANSITION
1052   A763  8D 07              LA763  BSR   LA76C           READ CASSETTE INPUT BIT
1053   A765  25 FC                    BCS   LA763           LOOP UNTIL IT IS LO
1054   A767  8D 03              LA767  BSR   LA76C           READ CASSETTE INPUT DATA
1055   A769  24 FC                    BCC   LA767           WAIT UNTIL IT GOES HI
1056   A76B  39                       RTS
1057
1058                           * READ CASSETTE INPUT BIT OF THE PIA
1059   A76C  0C 83              LA76C  INC   CPERTM          INCREMENT PERIOD TIMER
1060   A76E  F6 FF 20                  LDB   PIA1            GET CASSETTE INPUT BIT
1061   A771  56                       RORB                  PUT CASSETTE BIT INTO THE CARRY FLAG
1062   A772  39                       RTS
1063
1064                           * WAIT FOR HI - LO TRANSITION
1065   A773  8D F7              LA773  BSR   LA76C           READ CASSETTE INPUT DATA
1066   A775  24 FC                    BCC   LA773           LOOP UNTIL IT IS HI
1067   A777  8D F3              LA777  BSR   LA76C           READ CASSETTE INPUT
1068   A779  25 FC                    BCS   LA777           LOOP UNTIL IT IS LO
```

```
1069   A77B  39                        RTS
1070
1071                           *** LOOK FOR THE SYNC BYTES - RETURN WITH ACCA = Ø IF SYNC ED
1072                           *** ON HI - LO TRANSITION, ACCA = $AØ IF SYNC ED ON THE
1073                           *** LO - HI TRANSITION OF THE INPUT SIGNAL FROM THE CASSETTE.
1074   A77C  1A 50             CASON     ORCC  #$50                    DISABLE IRQ,FIRQ
1075   A77E  8D 4A                       BSR   LA7CA                   TURN ON TAPE DECK MOTOR
1076   A780  ØF 82                       CLR   CPULWD                  RESET UP TO SPEED COUNTER
1077   A782  8D DF             LA782     BSR   LA763                   WAIT FOR LO-HI TRANSITION
1078   A784  8D 27             LA784     BSR   LA7AD                   WAIT FOR HI-LO TRANSITION
1079   A786  22 ØF                       BHI   LA797                   CASSETTE SPEED IN RANGE FOR 12ØØ HZ
1080   A788  8D 1D             LA788     BSR   LA7A7                   WAIT FOR LO-HI TRANSITION
1081   A78A  25 ØF                       BCS   LA79B                   CASSETTE SPEED IN RANGE FOR 24ØØ HZ
1082   A78C  ØA 82                       DEC   CPULWD                  DECREMENT UP TO SPEED COUNTER IF SYNC ED ON LO-HI
1083   A78E  96 82                       LDA   CPULWD                  GET IT
1084   A790  81 AØ                       CMPA  #-96                    HAVE THERE BEEN 96 CONSECUTIVE 1-Ø-1-Ø PATTERNS
1085   A792  26 EE             LA792     BNE   LA782                   NO
1086   A794  97 84                       STA   CBTPHA                  SAVE WHICH TRANSITION (HI-LO OR LO-HI)
1087   A796  39                          RTS
1088   A797  8D ØE             LA797     BSR   LA7A7                   WAIT FOR LO-HI TRANSITION
1089   A799  22 E9                       BHI   LA784                   BRANCH IF TWO CONSECUTIVE 12ØØ HZ PULSES
1090   A79B  8D 1Ø             LA79B     BSR   LA7AD                   WAIT FOR HI-LO TRANSITION
1091   A79D  25 E9                       BCS   LA788                   BRANCH IF TWO CONSECUTIVE 24ØØ HZ PULSES
1092   A79F  ØC 82                       INC   CPULWD                  INCREMENT UP TO SPEED COUNTER IF SYNC ED ON HI-LO
1093   A7A1  96 82                       LDA   CPULWD                  GET IT
1094   A7A3  8Ø 6Ø                       SUBA  #96                     GOT ENOUGH SYNC PULSES? - ACCA WILL BE ZERO IF
1095                           *                                       THERE HAVE BEEN 96 CONSECUTIVE Ø-1-Ø-1 PATTERNS
1096   A7A5  2Ø EB                       BRA   LA792
1097   A7A7  ØF 83             LA7A7     CLR   CPERTM                  RESET PERIOD TIMER
1098   A7A9  8D BC                       BSR   LA767                   WAIT UNTIL CASSETTE INPUT GOES HI
1099   A7AB  2Ø Ø4                       BRA   LA7B1
1100   A7AD  ØF 83             LA7AD     CLR   CPERTM                  RESET PERIOD TIMER
1101   A7AF  8D C6                       BSR   LA777                   WAIT UNTIL CASSETTE GOES LO
1102   A7B1  D6 83             LA7B1     LDB   CPERTM                  GET PERIOD TIMER
1103   A7B3  D1 9Ø                       CMPB  CMPØ                    UPPER LIMIT OF 12ØØ HZ PERIOD
1104   A7B5  22 Ø3                       BHI   LA7BA                   BRANCH IF CASSETTE SPEED IS TOO SLOW OR DROPOUT
1105   A7B7  D1 91                       CMPB  CMP1                    UPPER LIMIT OF 24ØØ HZ PERIOD
1106   A7B9  39                          RTS
1107   A7BA  ØF 82             LA7BA     CLR   CPULWD                  RESET UP TO SPEED COUNTER
1108   A7BC  39                          RTS
1109
1110                           * MOTOR
1111   A7BD  1F 89             MOTOR     TFR   A,B                     SAVE CURRENT TOKEN IN ACCB
1112   A7BF  9D 9F                       JSR   GETNCH                  GET NEXT INPUT CHARACTER FROM BASIC
1113   A7C1  C1 AA                       CMPB  #$AA                     OFF  TOKEN
1114   A7C3  27 24                       BEQ   LA7E9                   YES
1115   A7C5  C1 88                       CMPB  #$88                     ON  TOKEN
1116   A7C7  BD A5 C9                    JSR   LA5C9                   SYNTAX ERROR IF IT WASN T  ON  OR  OFF
1117   A7CA  B6 FF 21           LA7CA     LDA   PIA1+1                  READ CRA OF U4
1118   A7CD  8A Ø8                       ORA   #$Ø8                    TURN ON BIT 3 WHICH ENABLES MOTOR DELAY
1119   A7CF  8D 1F                       BSR   LA7FØ                   PUT IT BACK
1120   A7D1  9E 8A             LA7D1     LDX   ZERO                    GET READY TO WAIT A WHILE
1121
1122                           * DELAY WHILE DECREMENTING X TO ZERO
1123   A7D3  3Ø 1F             LA7D3     LEAX  -1,X                    DECREMENT X
1124   A7D5  26 FC                       BNE   LA7D3                   BRANCH IF NOT ZERO
1125   A7D7  39                          RTS
1126
1127                           * SEND SYNCLN $55 S TO TAPE
1128   A7D8  1A 50             WRLDR     ORCC  #$50                    DISABLE INTERRUPTS
1129   A7DA  8D EE                       BSR   LA7CA                   TURN ON TAPE DECK MOTOR
1130   A7DC  9E 92                       LDX   SYNCLN                  GET COUNT OF $55 S TO SEND
1131   A7DE  8D 48             LA7DE     BSR   LA828                   SEND $55 TO TAPE
1132   A7EØ  3Ø 1F                       LEAX  -1,X                    ARE ALL $55 S SENT?
1133   A7E2  26 FA                       BNE   LA7DE                   NO
1134   A7E4  39                          RTS
1135
1136                           * WRITE SYNC BYTES AND A BLOCK TO TAPE
1137   A7E5  8D F1             LA7E5     BSR   WRLDR                   WRITE SYNC BYTES TO TAPE
1138   A7E7  8D ØB             LA7E7     BSR   SNDBLK                  GO WRITE A BLOCK
1139
1140                           * TURN OFF TAPE DECK MOTOR
1141   A7E9  1C AF             LA7E9     ANDCC #$AF                    ENABLE IRQ,FIRQ
1142   A7EB  B6 FF 21                    LDA   PIA1+1                  READ CRA OF U4
1143   A7EE  84 F7                       ANDA  #$F7                    TURN OFF BIT 3
1144   A7FØ  B7 FF 21           LA7FØ     STA   PIA1+1                  PUT IT BACK
1145   A7F3  39                          RTS
1146
1147                           * WRITE A BLOCK TO CASSETTE
1148                           * BUFFER SIZE IN BLKLEN
1149                           * STARTING ADDR IN CBUFAD
1150                           * BLOCK NUMBER IN BLKTYP
1151   A7F4  1A 50             SNDBLK    ORCC  #$50                    DISABLE IRQ,FIRQ
1152   A7F6  D6 7D                       LDB   BLKLEN                  GET CHAR COUNT
1153   A7F8  D7 81                       STB   CSRERR                  TEMP CHAR COUNT
1154   A7FA  96 7D                       LDA   BLKLEN                  GET CHAR COUNT (INCLUDED IN CHECKSUM)
1155   A7FC  27 Ø7                       BEQ   LA8Ø5                   BRANCH IF NO CHARACTERS - NULL
1156   A7FE  9E 7E                       LDX   CBUFAD                  GET STARTING ADDRESS
1157   A8ØØ  AB 8Ø             LA8ØØ     ADDA  ,X+                     CHECKSUM THE BUFFER
```

```
1158  A802  5A                        DECB                    DONE ALL CHARACTERS?
1159  A803  26 FB                     BNE   LA800              NO
1160  A805  9B 7C           LA805     ADDA  BLKTYP             ADD IN THE BLOCK NUMBER
1161  A807  97 80                     STA   CCKSUM             SAVE THE CHECKSUM
1162  A809  9E 7E                     LDX   CBUFAD             GET STARTING ADDRESS
1163  A80B  8D 1B                     BSR   LA828              SEND $55 TO TAPE
1164  A80D  86 3C                     LDA   #$3C               SYNC CHAR
1165  A80F  8D 19                     BSR   LA82A              SEND TO TAPE
1166  A811  96 7C                     LDA   BLKTYP             GET BLOCK NUMBER
1167  A813  8D 15                     BSR   LA82A              SEND BLOCK NUMBER TO TAPE
1168  A815  96 7D                     LDA   BLKLEN             GET CHARACTER COUNT
1169  A817  8D 11                     BSR   LA82A              SEND CHAR COUNT TO TAPE
1170  A819  4D                        TSTA                    SET FLAGS
1171  A81A  27 08                     BEQ   LA824              BRANCH IF CHAR COUNT IS ZERO
1172  A81C  A6 80           LA81C     LDA   ,X+                GET BUFFER CHARACTER
1173  A81E  8D 0A                     BSR   LA82A              SEND BUFFER TO TAPE
1174  A820  0A 81                     DEC   CSRERR             DECR TEMP CHAR COUNT
1175  A822  26 F8                     BNE   LA81C              NOT DONE YET
1176  A824  96 80           LA824     LDA   CCKSUM             GET CHECKSUM
1177  A826  8D 02                     BSR   LA82A              SEND CHECKSUM TO TAPE
1178  A828  86 55           LA828     LDA   #$55               SEND A $55 TO TAPE
1179
1180                        * THIS ROUTINE SENDS THE A REG TO TAPE
1181  A82A  34 02           LA82A     PSHS  A                  SAVE OUTPUT CHARACTER
1182  A82C  C6 01                     LDB   #1                 ACCB CONTAINS A MASK USED TO DETERMINE WHETHER A
1183                        *                                  BIT IN THE OUTPUT CHARACTER IS HI OR LO
1184  A82E  96 85           LA82E     LDA   CLSTSN             GET THE ENDING VALUE OF THE LAST SINE CYCLE
1185  A830  B7 FF 20                  STA   DA                 STORE IN THE D/A CONVERTER
1186  A833  10 8E A8 5C              LDY   #LA85C             SINE LOOK-UP TABLE FOR GENERATING FSK
1187  A837  E5 E4                     BITB  ,S                 IS THE CURRENT BIT A ONE OR A ZERO ?
1188  A839  26 0D                     BNE   LA848              IF A 1, DO HIGH FREQ
1189                        * LOW FREQUENCY LOOK UP
1190  A83B  A6 A0           LA83B     LDA   ,Y+                USE EVERY BYTE IN TABLE IF LOW FREQUENCY
1191  A83D  10 8C A8 80               CMPY  #LA85C+36          END OF SINE TABLE?
1192  A841  27 12                     BEQ   LA855              YES
1193  A843  B7 FF 20                  STA   DA                 SEND NEXT VALUE TO D/A CONVERTER
1194  A846  20 F3                     BRA   LA83B              GET NEXT VALUE
1195                        * HIGH FREQUENCY LOOK UP
1196  A848  A6 A1           LA848     LDA   ,Y++               USE EVERY OTHER BYTE IF HIGH FREQUENCY
1197  A84A  10 8C A8 80               CMPY  #LA85C+36          END OF SINE TABLE?
1198  A84E  27 05                     BEQ   LA855              YES
1199  A850  B7 FF 20                  STA   DA                 SEND NEXT VALUE TO D/A CONVERTER
1200  A853  20 F3                     BRA   LA848              GET NEXT VALUE
1201  A855  97 85           LA855     STA   CLSTSN             SAVE THE LAST VALUE SENT TO THE D/A CONVERTER
1202  A857  58                        ASLB                    SHIFT MASK BIT LEFT
1203  A858  24 D4                     BCC   LA82E              DONE WHEN MASK BIT IS SHIFTED INTO CARRY FLAG
1204  A85A  35 82                     PULS  A,PC               RESTORE OUTPUT CHARACTER AND RETURN
1205
1206                        * THIS IS A LOOK-UP TABLE OF SINE VALUES FOR THE TAPE DECK FSK
1207                        * (BIT 1 IS USED TO KEEP THE SERIAL OUTPUT MARKING)
1208  A85C  82 92 AA BA CA DA LA85C     FCB   $82,$92,$AA,$BA,$CA,$DA
1209  A862  EA F2 FA FA FA F2           FCB   $EA,$F2,$FA,$FA,$FA,$F2
1210  A868  EA DA CA BA AA 92           FCB   $EA,$DA,$CA,$BA,$AA,$92
1211  A86E  7A 6A 52 42 32 22           FCB   $7A,$6A,$52,$42,$32,$22
1212  A874  12 0A 02 02 02 0A           FCB   $12,$0A,$02,$02,$02,$0A
1213  A87A  12 22 32 42 52 6A           FCB   $12,$22,$32,$42,$52,$6A
1214
1215                        * SET
1216  A880  8D 3F           SET       BSR   LA8C1              6ET ABSOLUTE SCREEN POSITION OF GRAPHICS BLOCK
1217  A882  34 10                     PSHS  X                  SAVE CHARACTER LOCATION
1218  A884  BD B7 38                  JSR   LB738              SYNTAX CHECK FOR COMMA - RETURN EXPR VALUE IN ACCB
1219  A887  35 10                     PULS  X                  REGET CHARACTER LOCATION
1220  A889  C1 08                     CMPB  #8                 NINE ALLOWABLE COLORS
1221  A88B  22 48                     BHI   LA8D5              ILLEGAL COLOR -  ILLEGAL FUNCTION CALL
1222  A88D  5A                        DECB                    CHANGE COLOR NUMBERS FROM 0-8 TO (-1 TO 7)
1223  A88E  2B 05                     BMI   LA895              BRANCH IF SET (X,Y,0)
1224  A890  86 10                     LDA   #$10               $10 OFFSET BETWEEN DIFFERENT COLORS
1225  A892  3D                        MUL                     MULT BY COLOR FOR TOTAL OFFSET
1226  A893  20 08                     BRA   LA89D              GO SAVE THE COLOR
1227  A895  E6 84           LA895     LDB   ,X                 GET CURRENT CHAR FROM SCREEN
1228  A897  2A 03                     BPL   LA89C              BRANCH IF NOT GRAPHIC
1229  A899  C4 70                     ANDB  #$70               SAVE ONLY THE COLOR INFO
1230  A89B  21                        FCB   SKP1               SKIP THE NEXT BYTE
1231  A89C  5F              LA89C     CLRB                    RESET ASCII BLOCK TO ZERO COLOR
1232  A89D  34 04                     PSHS  B                  SAVE COLOR INFO
1233  A89F  8D 6C                     BSR   LA90D              SYNTAX CHECK FOR  )
1234  A8A1  A6 84                     LDA   ,X                 GET CURRENT CHARACTER FROM SCREEN
1235  A8A3  2B 01                     BMI   LA8A6              BRANCH IF GRAPHIC
1236  A8A5  4F                        CLRA                    RESET ASCII CHARACTER TO ALL PIXELS OFF
1237  A8A6  84 0F           LA8A6     ANDA  #$0F               SAVE ONLY PIXEL ON/OFF INFO
1238  A8A8  9A 86                     ORA   GRBLOK              OR  WITH WHICH PIXEL TO TURN ON
1239  A8AA  AA E0                     ORA   ,S+                 OR  IN THE COLOR
1240  A8AC  8A 80           LA8AC     ORA   #$80               FORCE GRAPHIC
1241  A8AE  A7 84                     STA   ,X                 DISPLAY IT ON THE SCREEN
1242  A8B0  39                        RTS
1243
1244                        * RESET
1245  A8B1  8D 0E           RESET     BSR   LA8C1              GET ABSOLUTE SCREEN ADDRESS OF THIS CHARACTER
1246  A8B3  8D 58                     BSR   LA90D              SYNTAX CHECK FOR ")"
```

```
1247   A8B5  4F                        CLRA                        * ACCA=ZERO GRAPHIC BLOCK - FOR USE IN CASE YOU RE
1248                                                                * TRYING TO RESET A NON GRAPHIC BLOCK
1249   A8B6  E6 84                     LDB   ,X                    GET CURRENT CHAR FROM SCREEN
1250   A8B8  2A F2                     BPL   LA8AC                 BRANCH IF NON-GRAPHIC
1251   A8BA  03 86                     COM   GRBLOK                INVERT PIXEL ON/OFF MASK
1252   A8BC  D4 86                     ANDB  GRBLOK                AND IT WITH CURRENT ON/OFF DATA
1253   A8BE  E7 84                     STB   ,X                    DISPLAY IT
1254   A8C0  39                        RTS
1255
1256                       *** THIS ROUTINE WILL CHECK SYNTAX AND CHECK FOR LEGAL VALUES
1257                       *** OF SET,RESET & POINT HORIZONTAL AND VERTICAL PARAMETERS
1258                       *** AND RETURN THEIR ABSOLUTE SCREEN ADDRESS IN THE X REGISTER
1259                       *** WHICH OF THE FOUR PIXELS OF THE GRAPHIC BLOCK SELECTED
1260                       *** IS RETURNED IN GRBLOK.
1261   A8C1  BD B2 6A      LA8C1     JSR   LB26A                 SYNTAX CHECK FOR "("
1262   A8C4  BD 01 9D      LA8C4     JSR   RVEC21                HOOK INTO RAM
1263   A8C7  BD B7 0B                JSR   LB70B                 EVALUATE EXPRESSION - RETURN VALUE IN ACCB
1264   A8CA  C1 3F                   CMPB  #63                   ONLY 64 HORIZONTAL GRAPHIC BLOCKS
1265   A8CC  22 07                   BHI   LA8D5                  ILLEGAL FUNCTION CALL
1266   A8CE  34 04                   PSHS  B                     SAVE HOR COORD
1267   A8D0  BD B7 38                JSR   LB738                 SYNTAX CHECK FOR COMMA AND EVALUATE EXPR
1268   A8D3  C1 1F                   CMPB  #31                   ONLY 32 VERTICAL BLOCKS
1269   A8D5  22 71         LA8D5     BHI   LA948                  ILLEGAL FUNCTION CALL
1270   A8D7  34 04                   PSHS  B                     SAVE VERT COORD
1271   A8D9  54                      LSRB                        DIVIDE BY TWO BECAUSE THERE ARE 2 GRAPHIC PIXELS/HOR
1272                       *                                     CHARACTER POSITION (BYTE)
1273   A8DA  86 20                   LDA   #32                   32 BYTES/ROW
1274   A8DC  3D                      MUL                         GET ROW OFFSET OF CHAR POSITION
1275   A8DD  8E 04 00                LDX   #VIDRAM               SCREEN BUFFER ADDRESS
1276   A8E0  30 8B                   LEAX  D,X                   ADD ROW OFFSET TO SCREEN BUFFER ADDRESS
1277   A8E2  E6 61                   LDB   1,S                   GET HOR COORD
1278   A8E4  54                      LSRB                        2 VERTICAL PIXELS/CHARACTER POSITION
1279   A8E5  3A                      ABX                         ADD VERTICAL OFFSET TO CHARACTER ADDRESS
1280   A8E6  35 06                   PULS  A,B                   GET VER COORD TO ACCA, HOR COORD TO ACCB
1281   A8E8  84 01                   ANDA  #1                    KEEP ONLY LSB OF VER COORD
1282   A8EA  56                      RORB                        LSB OF HOR COORD TO CARRY FLAG
1283   A8EB  49                      ROLA                        LSB OF HOR TO BIT 0 OF ACCA
1284   A8EC  C6 10                   LDB   #$10                  MAKE A BIT MASK - TURN ON BIT 4
1285   A8EE  54            LA8EE     LSRB                        SHIFT IT RIGHT ONCE
1286   A8EF  4A                      DECA                        SHIFTED IT ENOUGH?
1287   A8F0  2A FC                   BPL   LA8EE                 NO
1288   A8F2  D7 86                   STB   GRBLOK                ACCB=8 FOR UPPER LEFT PIXEL, =4 FOR UPPER RIGHT
1289                       *                                     PIXEL =2 FOR LOWER LEFT, =1 FOR LOWER RIGHT
1290   A8F4  39                      RTS
1291
1292                       * POINT
1293   A8F5  8D CD         POINT     BSR   LA8C4                 EVALUATE EXPRESSION
1294   A8F7  C6 FF                   LDB   #$FF                  INITIAL VALUE OF ON/OFF FLAG = OFF (FALSE)
1295   A8F9  A6 84                   LDA   ,X                    GET CURRENT GRAPHIC CHARACTER
1296   A8FB  2A 0D                   BPL   LA90A                 BRANCH IF NON-GRAPHIC (ALWAYS FALSE)
1297   A8FD  94 86                   ANDA  GRBLOK                 AND  CURR CHAR WITH THE PIXEL IN QUESTION
1298   A8FF  27 08                   BEQ   LA909                 BRANCH IF THE ELEMENT IS OFF
1299   A901  E6 84                   LDB   ,X                    GET CURRENT CHARACTER
1300   A903  54                      LSRB                        * SHIFT RIGHT
1301   A904  54                      LSRB                        * SHIFT RIGHT
1302   A905  54                      LSRB                        * SHIFT RIGHT
1303   A906  54                      LSRB                        * SHIFT RIGHT - NOW THE HIGH NIBBLE IS IN THE LOW NIBBLE
1304   A907  C4 07                   ANDB  #7                    KEEP ONLY THE COLOR INFO
1305   A909  5C            LA909     INCB                        ACCB=0 FOR NO COLOR, =1 TO 8 OTHERWISE
1306   A90A  BD A5 E8      LA90A     JSR   LA5E8                 CONVERT ACCB TO FLOATING POINT
1307   A90D  7E B2 67      LA90D     JMP   LB267                 SYNTAX CHECK FOR  )
1308
1309                       * CLS
1310   A910  BD 01 A0      CLS       JSR   RVEC22                HOOK INTO RAM
1311   A913  27 13                   BEQ   LA928                 BRANCH IF NO ARGUMENT
1312   A915  BD B7 0B                JSR   LB70B                 CALCULATE ARGUMENT, RETURN VALUE IN ACCB
1313   A918  C1 08                   CMPB  #8                    VALID ARGUMENT?
1314   A91A  22 1B                   BHI   LA937                 IF ARGUMENT >8, GO PRINT  MICROSOFT
1315   A91C  5D                      TSTB                        SET FLAGS
1316   A91D  27 06                   BEQ   LA925                 COLOR 0
1317   A91F  5A                      DECB                        ACCB NOW CONTAINS 0-7
1318   A920  86 10                   LDA   #$10                  EACH GRAPHIC BLOCK SEPARATED BY $10 FROM ONE ANOTHER
1319   A922  3D                      MUL                         ACCB CONTAINS ONE OF 8 OFFSETS
1320   A923  CA 0F                   ORB   #$0F                  BITS 0-3 SET FOR SOLID COLOR GRAPHIC BLOCK
1321   A925  CA 80      LA925       ORB   #$80                  BIT 7 SET FOR GRAPHICS
1322   A927  8C                      FCB   SKP2                  SKIP TWO BYTES
1323
1324                       * CLEAR SCREEN
1325   A928  C6 60      LA928        LDB   #$60                  BLANK
1326   A92A  8E 04 00                LDX   #VIDRAM               GET ADDR OF START OF SCREEN BUFFER
1327   A92D  9F 88      LA92D        STX   CURPOS                SAVE IT IN CURPOS
1328   A92F  E7 80      LA92F        STB   ,X+                   FILL SCREEN WITH CONTENTS OF ACCB
1329   A931  8C 05 FF                CMPX  #VIDRAM+511           END OF SCREEN?
1330   A934  23 F9                   BLS   LA92F                 NO
1331   A936  39                      RTS
1332   A937  8D EF      LA937       BSR   LA928                 CLEAR SCREEN
1333   A939  8E A1 65                LDX   #LA166-1              *
1334   A93C  7E B9 9C                JMP   LB99C                 * PRINT  MICROSOFT
1335
```

```
1336  A93F  BD B2 6D        LA93F   JSR   LB26D                  SYNTAX CHECK FOR A COMMA
1337  A942  BD B7 ØB        LA942   JSR   LB7ØB                  EVALUATE EXPRESSION, RETURN VALUE IN ACCB
1338  A945  5D              LA945   TSTB                         SET FLAGS
1339  A946  26 3C                    BNE   LA984                  RETURN IF NON ZERO
134Ø  A948  7E B4 4A        LA948   JMP   LB44A                   ILLEGAL FUNCTION CALL  IF ZERO
1341
1342                        * SOUND
1343  A94B  8D F5           SOUND   BSR   LA942                  EVALUATE EXPRESSION (FREQUENCY)
1344  A94D  D7 8C                   STB   SNDTON                 SAVE IT
1345  A94F  8D EE                   BSR   LA93F                  EVALUATE EXPRESSION (SOUND LENGTH)
1346  A951  86 Ø4                   LDA   #4                     CONSTANT FACTOR
1347  A953  3D                      MUL                          EXPAND LENGTH EXPRESSION
1348  A954  DD 8D                   STD   SNDDUR                 SAVE LENGTH OF SOUND
1349  A956  B6 FF Ø3                LDA   PIAØ+3                 GET CONTROL REGISTER OF PIAØ, PORT B
135Ø  A959  8A Ø1                   ORA   #1                     *
1351  A95B  B7 FF Ø3                STA   PIAØ+3                 * ENABLE 6Ø HZ INTERRUPT (PIAØ IRQ)
1352  A95E  ØF Ø8                   CLR   ARYDIS                 CLEAR THE ARRAY DISABLE FLAG - FOR NO APPARENT REASON
1353  A96Ø  8D 4Ø                   BSR   LA9A2                  CONNECT D/A SOUND INPUT TO OUTPUT OF SOUND MUX
1354  A962  8D 12                   BSR   LA976                  TURN ON AUDIO - ENABLE SOUND MUX
1355  A964  8D 1F           LA964   BSR   LA985                  STORE 2.5 VOLTS TO D/A AND WAIT
1356  A966  86 FE                   LDA   #$FE                   DATA TO MAKE D/A OUT = 5 VOLTS
1357  A968  8D 1D                   BSR   LA987                  STORE IT TO D/A AND WAIT
1358  A96A  8D 19                   BSR   LA985                  STORE 2.5 VOLTS TO D/A AND WAIT
1359  A96C  86 Ø2                   LDA   #2                     DATA TO MAKE D/A OUT = Ø VOLTS
136Ø  A96E  8D 17                   BSR   LA987                  STORE IT TO D/A AND WAIT
1361  A97Ø  9E 8D                   LDX   SNDDUR                 * IS SNDDUR = Ø? - THE IRQ INTERRUPT SERVICING
1362                        *                                    * ROUTINE WILL DECREMENT SNDDUR
1363  A972  26 FØ                   BNE   LA964                  NOT DONE YET
1364
1365                        * THESE ROUTINES WILL ENABLE/DISABLE THE ANALOG MUX
1366  A974  4F              LA974   CLRA                         BIT 3 OF ACCA = Ø, DISABLE ANALOG MUX
1367  A975  8C                      FCB   SKP2                   SKIP TWO BYTES
1368  A976  86 Ø8           LA976   LDA   #8                     BIT 3 OF ACCA = 1, ENABLE ANALOG MUX
1369  A978  A7 E2                   STA   ,-S                    SAVE ACCA ON STACK
137Ø  A97A  B6 FF 23                LDA   PIA1+3                 GET CONTROL REGISTER OF PIA1, PORT B
1371  A97D  84 F7                   ANDA  #$F7                   RESET BIT 3
1372  A97F  AA E Ø                  ORA   ,S+                    OR IN BIT 3 OF ACCA (SAVED ON STACK)
1373  A981  B7 FF 23                STA   PIA1+3                 SET/RESET CB2 OF U4
1374  A984  39              LA984   RTS
1375  A985  86 7E           LA985   LDA   #$7E                   DATA VALUE TO MAKE D/A OUTPUT = 2.5 VOLTS
1376  A987  B7 FF 2Ø         LA987   STA   DA                     STORE IT IN D/A
1377  A98A  96 8C                   LDA   SNDTON                 GET FREQUENCY
1378  A98C  4C              LA98C   INCA                         INCREMENT IT
1379  A98D  26 FD                   BNE   LA98C                  LOOP UNTIL DONE
138Ø  A98F  39                      RTS
1381
1382                        * AUDIO
1383  A99Ø  1F 89           AUDIO   TFR   A,B                    SAVE ON/OFF TOKEN IN ACCB
1384  A992  9D 9F                   JSR   GETNCH                 MOVE BASIC POINTER TO NEXT CHARACTER
1385  A994  C1 AA                   CMPB  #$AA                    OFF  TOKEN?
1386  A996  27 DC                   BEQ   LA974                  YES - TURN OFF ANALOG MUX
1387  A998  CØ 88                   SUBB  #$88                    ON  TOKEN
1388  A99A  BD A5 C9                JSR   LA5C9                  SYNTAX ERROR IF NOT  OFF  OR  ON
1389  A99D  5C                      INCB                         NOW ACCB = 1
139Ø  A99E  8D Ø2                   BSR   LA9A2                  ROUTE CASSETTE TO SOUND MULTIPLEXER
1391  A9AØ  2Ø D4                   BRA   LA976                  ENABLE SOUND MULTIPLEXER
1392
1393                        * THIS ROUTINE WILL TRANSFER BIT Ø OF ACCB TO SEL 1 OF
1394                        * THE ANALOG MULTIPLEXER AND BIT 1 OF ACCB TO SEL 2.
1395  A9A2  CE FF Ø1        LA9A2   LDU   #PIAØ+1                POINT U TO PIAØ CONTROL REG
1396  A9A5  8D ØØ                   BSR   LA9A7                  PROGRAM 1ST CONTROL REGISTER
1397  A9A7  A6 C4            LA9A7   LDA   ,U                     GET PIA CONTROL REGISTER
1398  A9A9  84 F7                   ANDA  #$F7                   RESET CA2 (CB2) OUTPUT BIT
1399  A9AB  57                      ASRB                         SHIFT ACCB BIT Ø TO CARRY FLAG
14ØØ  A9AC  24 Ø2                   BCC   LA9BØ                  BRANCH IF CARRY = ZERO
14Ø1  A9AE  8A Ø8                   ORA   #$Ø8                   FORCE BIT 3=1; SET CA2(CB2)
14Ø2  A9BØ  A7 C1            LA9BØ   STA   ,U++                   PUT IT BACK IN THE PIA CONTROL REGISTER
14Ø3  A9B2  39                      RTS
14Ø4
14Ø5                        * IRQ SERVICE
14Ø6  A9B3  B6 FF Ø3        BIRQSV  LDA   PIAØ+3                 CHECK FOR 6ØHZ INTERRUPT
14Ø7  A9B6  2A ØD                   BPL   LA9C5                  RETURN IF 63.5 MICROSECOND INTERRUPT
14Ø8  A9B8  B6 FF Ø2                LDA   PIAØ+2                 RESET PIAØ, PORT B INTERRUPT FLAG
14Ø9  A9BB  BE ØØ 8D                LDX   >SNDDUR                GET INTERRUPT TIMER (SOUND COMMAND)
141Ø  A9BE  27 Ø5                   BEQ   LA9C5                  RETURN IF TIMER = Ø
1411  A9CØ  3Ø 1F                   LEAX  -1,X                   DECREMENT TIMER IF NOT = Ø
1412  A9C2  BF ØØ 8D                STX   >SNDDUR                SAVE NEW TIMER VALUE
1413  A9C5  3B              LA9C5   RTI                          RETURN FROM INTERRUPT
1414
1415                        * JOYSTK
1416  A9C6  BD B7 ØE        JOYSTK  JSR   LB7ØE                  EVALUATE JOYSTICK ARGUMENT
1417  A9C9  C1 Ø3                   CMPB  #3                     TWO JOYSTICKS MAXIMUM (HOR & VER FOR EACH)
1418  A9CB  1Ø 22 ØA 7B             LBHI  LB44A                   ILLEGAL FUNCTION CALL  IF >3
1419  A9CF  5D                      TSTB                         SET FLAGS
142Ø  A9DØ  26 Ø2                   BNE   LA9D4                  GET NEW DATA ONLY IF JOYSTK(Ø)
1421  A9D2  8D ØA                   BSR   GETJOY                 GET NEW DATA FOR ALL JOYSTICKS
1422  A9D4  8E Ø1 5A        LA9D4   LDX   #POTVAL                POINT X TO JOYSTICK DATA BUFFER
1423  A9D7  D6 53                   LDB   FPAØ+3                 WHICH JOYSTICK DID YOU WANT?
1424  A9D9  E6 85                   LDB   B,X                    PUT ITS DATA INTO ACCB
```

```
A9DB  7E B4 F3            JMP   LB4F3                  CONVERT ACCB INTO FLOATING POINT NUMBER
                *
                * JOYSTK DATA AT:
                *     $15A      $15B      $15C      $15D
                *     LEFT      LEFT      RIGHT     RIGHT
                *     VERT      HORIZ     VERT      HORIZ
                *
                ** THIS IS A 6 BIT SOFTWARE A/D CONVERSION ROUTINE
A9DE  8D 94     GETJOY    BSR   LA974                TURN OFF AUDIO
A9E0  8E 01 5E            LDX   #POTVAL+4            POINT X TO JOYSTICK DATA BUFFER
A9E3  C6 03               LDB   #3                   GET FOUR SETS OF DATA (4 JOYSTICKS)
A9E5  86 0A     LA9E5     LDA   #10                  10 TRIES TO GET STABLE READING
A9E7  ED E3               STD   ,--S                 STORE JOYSTICK NUMBER AND TRY NUMBER ON THE STACK
A9E9  8D B7               BSR   LA9A2                SET THE SELECT INPUTS ON ANALOG MULTIPLEXER
A9EB  CC 40 80  LA9EB     LDD   #$4080               ACCA IS A SHIFT COUNTER OF HOW MANY BITS TO CONVERT
                *                                    AND WI1L BE $40 (6 BITS) FOR THE COLOR
                *                                    COMPUTER. ACCB CONTAINS A VALUE EQUAL TO 1/2
                *                                    THE CURRENT TRIAL DIFFERENCE. INITIALLY =$80 (2.5 VOLTS).
A9EE  A7 E2     LA9EE     STA   ,-S                  TEMP STORE SHIFT COUNTER ON STACK
A9F0  CA 02               ORB   #2                   KEEP RS 232 SERIAL OUT MARKING
A9F2  F7 FF 20            STB   DA                   STORE IN D/A CONVERTER
A9F5  C8 02               EORB  #2                   PUT R5232 OUTPUT BIT BACK TO ZERO
A9F7  B6 FF 00            LDA   PIA0                 HIGH BIT IS FROM COMPARATOR
A9FA  2B 03               BMI   LA9FF                BRANCH IF COMPARATOR OUTPUT IS HIGH
A9FC  E0 E4               SUBB  ,S                   SUBTRACT 1/2 THE CURRENT TRIAL DIFFERENCE
A9FE  8C                  FCB   SKP2                 SKIP NEXT TWO BYTES
A9FF  EB E4     LA9FF     ADDB  ,S                   ADD 1/2 OF THE CURRENT TRIAL DIFFERENCE
AA01  A6 E0               LDA   ,S+                  PULL SHIFT COUNTER OFF THE STACK
AA03  44                  LSRA                       SHIFT IT RIGHT ONCE
AA04  81 01               CMPA  #1                   HAVE ALL THE SHIFTS BEEN DONE?
AA06  26 E6               BNE   LA9EE                NO
AA08  54                  LSRB                       YES - THE DATA IS IN THE TOP 6 BYTES OF ACCB
AA09  54                  LSRB                       PUT IT INTO THE BOTTOM SIX
AA0A  E1 1F               CMPB  -1,X                 IS THIS VALUE EQUAL TO THE LAST TRY?
AA0C  27 04               BEQ   LAA12                YES - GO SAVE THE VALUE
AA0E  6A E4               DEC   ,S                   NO-DECREMENT TRIES COUNTER
AA10  26 D9               BNE   LA9EB                BRANCH IF YOU HAVEN T TRIED 10 TIMES
                *                                    IF YOU FALL THROUGH HERE YOU HAVE TRIED TO GET THE SAME READING
                *                                    10 TIMES AND NEVER GOTTEN A MATCH. AS A RESULT YOU JUST FALL
                *                                    THROUGH AND USE THE LAST VALUE READ IN.
AA12  E7 82     LAA12     STB   ,-X                  SAVE THE DIGITIZED VALUE
AA14  EC E1               LDD   ,S++                 GET THE NUMBER OF THE JOYSTICK JUST DONE
AA16  5A                  DECB                       DECR JOYSTK NUMBER
AA17  2A CC               BPL   LA9E5                BRANCH IF THE LAST ONE DONE WASN T NUMBER 0
AA19  39                  RTS
                *
                * SET CARRY IF NUMERIC - RETURN WITH
                * ZERO FLAG SET IF ACCA = 0 OR 3A(:) - END
                * OF BASIC LINE OR SUB LINE
AA1A  81 3A     BROMHK    CMPA  #'9+1                IS THIS CHARACTER >=(ASCII 9)+1?
AA1C  24 0A               BHS   LAA28                BRANCH IF > 9; Z SET IF = COLON
AA1E  81 20               CMPA  #SPACE               SPACE?
AA20  26 02               BNE   LAA24                NO - SET CARRY IF NUMERIC
AA22  0E 9F               JMP   GETNCH               IF SPACE, GET NECT CHAR (IGNORE SPACES)
AA24  80 30     LAA24     SUBA  #'0                  * SET CARRY IF
AA26  80 D0               SUBA  #-'0                 * CHARACTER > ASCII 0
AA28  39        LAA28     RTS
                *
                * DISPATCH TABLE FOR SECONDARY FUNCTIONS
                * TOKENS ARE PRECEEDED BY $FF                 TOKEN #
AA29  BC 7A     LAA29     FDB   SGN                  SGN      80
AA2B  BC EE               FDB   INT                  INT      81
AA2D  BC 93               FDB   ABS                  ABS      82
AA2F  01 12               FDB   $0112                USR      83
AA31  BF 1F               FDB   RND                  RND      84
AA33  BF 78               FDB   SIN                  SIN      85
AA35  B7 50               FDB   PEEK                 PEEK     86
AA37  B6 81               FDB   LEN                  LEN      87
AA39  B4 FD               FDB   STR                  STR$     88
AA3B  B7 16               FDB   VAL                  VAL      89
AA3D  B6 A0               FDB   ASC                  ASC      8A
AA3F  B6 8C               FDB   CHR                  CHR$     8B
AA41  A5 CE               FDB   EOF                  EOF      8C
AA43  A9 C6               FDB   JOYSTK               JOYSTK   8D
AA45  B6 AB               FDB   LEFT                 LEFT$    8E
AA47  B6 C8               FDB   RIGHT                RIGHT$   8F
AA49  B6 CF               FDB   MID                  MID$     90
AA4B  A8 F5               FDB   POINT                POINT    91
AA4D  A5 64               FDB   INKEY                INKEY$   92
AA4F  B4 EE               FDB   MEM                  MEM      93
                *
                * THIS TABLE CONTAINS PRECEDENCES AND DISPATCH ADDRESSES FOR ARITHMETIC
                * AND LOGICAL OPERATORS - THE NEGATION OPERATORS DO NOT ACT ON TWO OPERANDS
                * SO THEY ARE NOT LISTED IN THIS TABLE. THEY ARE TREATED SEPARATELY IN THE
                * EXPRESSION EVALUATION ROUTINE. THEY ARE:
                * UNARY NEGATION (-), PRECEDENCE &7D AND LOGICAL NEGATION (NOT), PRECEDENCE $5A
                * THE RELATIONAL OPERATORS < > = ARE ALSO NOT LISTED, PRECEDENCE $64.
                * A PRECEDENCE VALUE OF ZERO INDICATES END OF EXPRESSION OR PARENTHESES
                *
```

```
1514    AA51  79                LAA51     FCB   $79
1515    AA52  B9 C5                       FDB   LB9C5                  +
1516    AA54  79                          FCB   $79
1517    AA55  B9 BC                       FDB   LB9BC                  -
1518    AA57  7B                          FCB   $7B
1519    AA58  BA CC                       FDB   $BACC                  *
1520    AA5A  7B                          FCB   $7B
1521    AA5B  BB 91                       FDB   $BB91                  /
1522    AA5D  7F                          FCB   $7F
1523    AA5E  01 1D                       FDB   $011D                  EXPONENTIATION
1524    AA60  50                          FCB   $50
1525    AA61  B2 D5                       FDB   $B2D5                  AND
1526    AA63  46                          FCB   $46
1527    AA64  B2 D4                       FDB   LB2D4                  OR
1528                              *
1529                              * THIS IS THE RESERVED WORD TABLE
1530                              *                                    TOKEN #
1531    AA66  46 4F D2            LAA66     FCC   'FO',$80+'R'           80
1532    AA69  47 CF                       FCC   'G',$80+'O'            81
1533    AA6B  52 45 CD                    FCC   'RE',$80+'M'           82
1534    AA6E  A7                          FCB   '' + $80               83
1535    AA6F  45 4C 53 C5                 FCC   'ELS',$80+'E'          84
1536    AA73  49 C6                       FCC   'I',$80+'F'            85
1537    AA75  44 41 54 C1                 FCC   'DAT',$80+'A'          86
1538    AA79  50 52 49 4E D4              FCC   'PRIN',$80+'T'         87
1539    AA7E  4F CE                       FCC   'O',$80+'N'            88
1540    AA80  49 4E 50 55 D4              FCC   'INPU',$80+'T'         89
1541    AA85  45 4E C4                    FCC   'EN',$80+'D'           8A
1542    AA88  4E 45 58 D4                 FCC   'NEX',$80+'T'          8B
1543    AA8C  44 49 CD                    FCC   'DI',$80+'M'           8C
1544    AA8F  52 45 41 C4                 FCC   'REA',$80+'D'          8D
1545    AA93  52 55 CE                    FCC   'RU',$80+'N'           8E
1546    AA96  52 45 53 54 4F 52           FCC   'RESTOR',$80+'E'       8F
1547    AA9C  C5
1548    AA9D  52 45 54 55 52 CE           FCC   'RETUR',$80+'N'        90
1549    AAA3  53 54 4F D0                 FCC   'STO',$80+'P'          91
1550    AAA7  50 4F 4B C5                 FCC   'POK',$80+'E'          92
1551    AAAB  43 4F 4E D4                 FCC   'CON',$80+'T'          93
1552    AAAF  4C 49 53 D4                 FCC   'LIS',$80+'T'          94
1553    AAB3  43 4C 45 41 D2              FCC   'CLEA',$80+'R'         95
1554    AAB8  4E 45 D7                    FCC   'NE',$80+'W'           96
1555    AABB  43 4C 4F 41 C4              FCC   'CLOA',$80+'D'         97
1556    AAC0  43 53 41 56 C5              FCC   'CSAV',$80+'E'         98
1557    AAC5  4F 50 45 CE                 FCC   'OPE',$80+'N'          99
1558    AAC9  43 4C 4F 53 C5              FCC   'CLOS',$80+'E'         9A
1559    AACE  4C 4C 49 53 D4              FCC   'LLIS',$80+'T'         9B
1560    AAD3  53 45 D4                    FCC   'SE',$80+'T'           9C
1561    AAD6  52 45 53 45 D4              FCC   'RESE',$80+'T'         9D
1562    AADB  43 4C D3                    FCC   'CL',$80+'S'           9E
1563    AADE  4D 4F 54 4F D2              FCC   'MOTO',$80+'R'         9F
1564    AAE3  53 4F 55 4E C4              FCC   'SOUN',$80+'D'         A0
1565    AAE8  41 55 44 49 CF              FCC   'AUDI',$80+'O'         A1
1566    AAED  45 58 45 C3                 FCC   'EXE',$80+'C'          A2
1567    AAF1  53 4B 49 50 C6              FCC   'SKIP',$80+'F'         A3
1568    AAF6  54 41 42 A8                 FCC   'TAB',$80+'('          A4
1569    AAFA  54 CF                       FCC   'T',$80+'O'            A5
1570    AAFC  53 55 C2                    FCC   'SU',$80+'B'           A6
1571    AAFF  54 48 45 CE                 FCC   'THE',$80+'N'          A7
1572    AB03  4E 4F D4                    FCC   'NO',$80+'T'           A8
1573    AB06  53 54 45 D0                 FCC   'STE',$80+'P'          A9
1574    AB0A  4F 46 C6                    FCC   'OF',$80+'F'           AA
1575    AB0D  AB                          FCC   '+' + $80              AB
1576    AB0E  AD                          FCB   '-' + $80              AC
1577    AB0F  AA                          FCB   '*' + $80              AD
1578    AB10  AF                          FCB   '/' + $80              AE
1579    AB11  DE                          FCB   '^' + $80              AF
1580    AB12  41 4E C4                    FCC   'AN',$80+'D'           B0
1581    AB15  4F D2                       FCC   'O',$80+'R'            B1
1582    AB17  BE                          FCB   '>' + $80              B2
1583    AB18  BD                          FCB   '=' + $80              B3
1584    AB19  BC                          FCB   '<' + $80              B4
1585                              *
1586                              * TOKENS FOR THE SECONDARY FUNCTIONS ARE PRECEEDED BY $FF
1587                              *                                    TOKEN #
1588    AB1A  53 47 CE            LAB1A     FCC   'SG',$80+'N'           80
1589    AB1D  49 4E D4                    FCC   'IN',$80+'T'           81
1590    AB20  41 42 D3                    FCC   'AB',$80+'S'           82
1591    AB23  55 53 D2                    FCC   'US',$80+'R'           83
1592    AB26  52 4E C4                    FCC   'RN',$80+'D'           84
1593    AB29  53 49 CE                    FCC   'SI',$80+'N'           85
1594    AB2C  50 45 45 CB                 FCC   'PEE',$80+'K'          86
1595    AB30  4C 45 CE                    FCC   'LE',$80+'N'           87
1596    AB33  53 54 52 A4                 FCC   'STR',$80+'$'          88
1597    AB37  56 41 CC                    FCC   'VA',$80+'L'           89
1598    AB3A  41 53 C3                    FCC   'AS',$80+'C'           8A
1599    AB3D  43 48 52 A4                 FCC   'CHR',$80+'$'          8B
1600    AB41  45 4F C6                    FCC   'EO',$80+'F'           8C
1601    AB44  4A 4F 59 53 54 CB           FCC   'JOYST',$80+'K'        8D
1602    AB4A  4C 45 46 54 A4              FCC   'LEFT',$80+'$'         8E
```

```
1603  AB4F  52 49 47 48 54 A4        FCC   'RIGHT',$80+'$'          8F
1604  AB55  4D 49 44 A4              FCC   'MID',$80+'$'            90
1605  AB59  50 4F 49 4E D4           FCC   'POIN',$80+'T'           91
1606  AB5E  49 4E 4B 45 59 A4        FCC   'INKEY',$80+'$'          92
1607  AB64  4D 45 CD                 FCC   'ME',$80+'M'             93
1608                          *
1609                          * DISPATCH TABLE FOR COMMANDS                TOKEN #
1610  AB67  AD 47             LAB67 FDB   FOR                       FOR     80
1611  AB69  AE 86                   FDB   GO                        GO      81
1612  AB6B  AE E3                   FDB   REM                       REM     82
1613  AB6D  AE E3                   FDB   REM                       REM     83
1614  AB6F  AE E3                   FDB   REM                       ELSE    84
1615  AB71  AF 14                   FDB   IF                        IF      85
1616  AB73  AE E0                   FDB   DATA                      DATA    86
1617  AB75  B8 F7                   FDB   PRINT                     PRINT   87
1618  AB77  AF 42                   FDB   ON                        ON      88
1619  AB79  AF F5                   FDB   INPUT                     INPUT   89
1620  AB7B  AE 02                   FDB   END                       END     8A
1621  AB7D  B0 F8                   FDB   NEXT                      NEXT    8B
1622  AB7F  B3 4E                   FDB   DIM                       DIM     8C
1623  AB81  B0 46                   FDB   READ                      READ    8D
1624  AB83  AE 75                   FDB   RUN                       RUN     8E
1625  AB85  AD E4                   FDB   RESTOR                    RESTORE 8F
1626  AB87  AE C0                   FDB   RETURN                    RETURN  90
1627  AB89  AE 09                   FDB   STOP                      STOP    91
1628  AB8B  B7 57                   FDB   POKE                      POKE    92
1629  AB8D  AE 30                   FDB   CONT                      CONTINUE93
1630  AB8F  B7 64                   FDB   LIST                      LIST    94
1631  AB91  AE 41                   FDB   CLEAR                     CLEAR   95
1632  AB93  AD 17                   FDB   NEW                       NEW     96
1633  AB95  A4 98                   FDB   CLOAD                     CLOAD   97
1634  AB97  A4 4C                   FDB   CSAVE                     CSAVE   98
1635  AB99  A5 F6                   FDB   OPEN                      OPEN    99
1636  AB9B  A4 16                   FDB   CLOSE                     CLOSE   9A
1637  AB9D  B7 5E                   FDB   LLIST                     LLIST   9B
1638  AB9F  A8 80                   FDB   SET                       SET     9C
1639  ABA1  A8 B1                   FDB   RESET                     RESET   9D
1640  ABA3  A9 10                   FDB   CLS                       CLS     9E
1641  ABA5  A7 BD                   FDB   MOTOR                     MOTOR   9F
1642  ABA7  A9 4B                   FDB   SOUND                     SOUND   A0
1643  ABA9  A9 90                   FDB   AUDIO                     AUDIO   A1
1644  ABAB  A5 3E                   FDB   EXEC                      EXEC    A2
1645  ABAD  A5 EC                   FDB   SKIPF                     SKIPF   A3
1646                          *
1647                          * ERROR MESSAGES AND THEIR NUMBERS AS USED INTERNALLY
1648  ABAF  4E 46             LABAF FCC   'NF'                      0  NEXT WITHOUT FOR
1649  ABB1  53 4E                   FCC   'SN'                      1  SYNTAX ERROR
1650  ABB3  52 47                   FCC   'RG'                      2  RETURN WITHOUT GOSUB
1651  ABB5  4F 44                   FCC   'OD'                      3  OUT OF DATA
1652  ABB7  46 43                   FCC   'FC'                      4  ILLEGAL FUNCTION CALL
1653  ABB9  4F 56                   FCC   'OV'                      5  OVERFLOW
1654  ABBB  4F 4D                   FCC   'OM'                      6  OUT OF MEMORY
1655  ABBD  55 4C                   FCC   'UL'                      7  UNDEFINED LINE NUMBER
1656  ABBF  42 53                   FCC   'BS'                      8  BAD SUBSCRIPT
1657  ABC1  44 44                   FCC   'DD'                      9  REDIMENSIONED ARRAY
1658  ABC3  2F 30                   FCC   '/0'                      10 DIVISION BY ZERO
1659  ABC5  49 44                   FCC   'ID'                      11 ILLEGAL DIRECT STATEMENT
1660  ABC7  54 4D                   FCC   'TM'                      12 TYPE MISMATCH
1661  ABC9  4F 53                   FCC   'OS'                      13 OUT OF STRING SPACE
1662  ABCB  4C 53                   FCC   'LS'                      14 STRING TOO LONG
1663  ABCD  53 54                   FCC   'ST'                      15 STRING FORMULA TOO COMPLEX
1664  ABCF  43 4E                   FCC   'CN'                      16 CAN'T CONTINUE
1665  ABD1  46 44                   FCC   'FD'                      17 BAD FILE DATA
1666  ABD3  41 4F                   FCC   'AO'                      18 FILE ALREADY OPEN
1667  ABD5  44 4E                   FCC   'DN'                      19 DEVICE NUMBER ERROR
1668  ABD7  49 4F                   FCC   'IO'                      20 I/O ERROR
1669  ABD9  46 4D                   FCC   'FM'                      21 BAD FILE MODE
1670  ABDB  4E 4F                   FCC   'NO'                      22 FILE NOT OPEN
1671  ABDD  49 45                   FCC   'IE'                      23 INPUT PAST END OF FILE
1672  ABDF  44 53                   FCC   'DS'                      24 DIRECT STATEMENT IN FILE
1673
1674  ABE1  20 45 52 52 4F 52 LABE1 FCC   ' ERROR'
1675  ABE7  00                      FCB   $00
1676  ABE8  20 49 4E 20       LABE8 FCC   ' IN '
1677  ABEC  00                      FCB   $00
1678  ABED  0D                LABED FCB   CR
1679  ABEE  4F 4B             LABEE FCC   'OK'
1680  ABF0  0D 00                   FCB   CR,$00
1681  ABF2  0D                LABF2 FCB   CR
1682  ABF3  42 52 45 41 4B          FCC   'BREAK'
1683  ABF8  00                      FCB   $00
1684                          * SEARCH THE STACK FOR  GOSUB/RETURN  OR  FOR/NEXT  DATA.
1685                          * THE  FOR/NEXT  INDEX VARIABLE DESCRIPTOR ADDRESS BEING
1686                          * SOUGHT IS STORED IN VARDES. EACH BLOCK OF FOR/NEXT DATA IS 18
1687                          * BYTES WITH A $80 LEADER BYTE AND THE GOSUB/RETURN DATA IS 5 BYTES
1688                          * WITH AN $A6 LEADER BYTE.  THE FIRST NON "FOR/NEXT" DATA
1689                          * IS CONSIDERED  GOSUB/RETURN
1690  ABF9  30 64             LABF9 LEAX  4,S                       POINT X TO 3RD ADDRESS ON STACK - IGNORE THE
1691                          *                                     FIRST TWO RETURN ADDRESSES ON THE STACK
```

```
1692   ABFB  C6 12          LABFB    LDB   #18                  18 BYTES SAVED ON STACK FOR EACH  FOR  LOOP
1693   ABFD  9F 0F                   STX   TEMPTR               SAVE POINTER
1694   ABFF  A6 84                   LDA   ,X                   GET 1ST BYTE
1695   AC01  80 80                   SUBA  #$80                 * CHECK FOR TYPE OF STACK JUMP FOUND
1696   AC03  26 15                   BNE   LAC1A                * BRANCH IF NOT  FOR/NEXT
1697   AC05  AE 01                   LDX   1,X                  = GET INDEX VARIABLE DESCRIPTOR
1698   AC07  9F 11                   STX   TMPTR1               = POINTER AND SAVE IT IN TMPTR1
1699   AC09  9E 3B                   LDX   VARDES               GET INDEX VARIABLE BEING SEARCHED FOR
1700   AC0B  27 09                   BEQ   LAC16                BRANCH IF DEFAULT INDEX VARIABLE - USE THE
1701                        *                                   FIRST  FOR/NEXT  DATA FOUND ON STACK
1702                        *                                   IF NO INDEX VARIABLE AFTER  NEXT
1703   AC0D  9C 11                   CMPX  TMPTR1               DOES THE STACK INDEX MATCH THE ONE
1704                        *                                   BEING SEARCHED FOR?
1705   AC0F  27 09                   BEQ   LAC1A                YES
1706   AC11  9E 0F                   LDX   TEMPTR               * RESTORE INITIAL POINTER, ADD
1707   AC13  3A                      ABX                        * 18 TO IT AND LOOK FOR
1708   AC14  20 E5                   BRA   LABFB                * NEXT BLOCK OF DATA
1709   AC16  9E 11          LAC16    LDX   TMPTR1               = GET 1ST INDEX VARIABLE FOUND AND
1710   AC18  9F 3B                   STX   VARDES               = SAVE AS  NEXT  INDEX
1711   AC1A  9E 0F          LAC1A    LDX   TEMPTR               POINT X TO START OF  FOR/NEXT  DATA
1712   AC1C  4D                      TSTA                       SET ZERO FLAG IF  FOR/NEXT  DATA
1713   AC1D  39                      RTS
1714                        * CHECK FOR MEMORY SPACE FOR NEW TOP OF
1715                        * ARRAYS AND MOVE ARRAYS TO NEW LOCATION
1716   AC1E  8D 17          LAC1E    BSR   LAC37                 ACCD = NEW BOTTOM OF FREE RAM - IS THERE
1717                        *                                   ROOM FOR THE STACK?
1718                        * MOVE BYTES FROM V43(X) TO V41(U) UNTIL (X) = V47 AND
1719                        * SAVE FINAL VALUE OF U IN V45
1720   AC20  DE 41          LAC20    LDU   V41                  POINT U TO DESTINATION ADDRESS (V41)
1721   AC22  33 41                   LEAU  1,U                  ADD ONE TO U - COMPENSATE FOR FIRST PSHU
1722   AC24  9E 43                   LDX   V43                  POINT X TO SOURCE ADDRESS (V43)
1723   AC26  30 01                   LEAX  1,X                  ADD ONE - COMPENSATE FOR FIRST LDA ,X
1724   AC28  A6 82          LAC28    LDA   ,-X                  GRAB A BYTE FROM SOURCE
1725   AC2A  36 02                   PSHU  A                    MOVE IT TO DESTINATION
1726   AC2C  9C 47                   CMPX  V47                  DONE?
1727   AC2E  26 F8                   BNE   LAC28                NO - KEEP MOVING BYTES
1728   AC30  DF 45                   STU   V45                  SAVE FINAL DESTINATION ADDRESS
1729   AC32  39             LAC32    RTS
1730                        * CHECK TO SEE IF THERE IS ROOM TO STORE 2*ACCB
1731                        * BYTES IN FREE RAM - OM ERROR IF NOT
1732   AC33  4F             LAC33    CLRA                       * ACCD CONTAINS NUMBER OF EXTRA
1733   AC34  58                      ASLB                       * BYTES TO PUT ON STACK
1734   AC35  D3 1F                   ADDD  ARYEND               END OF PROGRAM AND VARIABLES
1735   AC37  C3 00 3A       LAC37    ADDD  #STKBUF              ADD STACK BUFFER - ROOM FOR STACK?
1736   AC3A  25 08                   BCS   LAC44                BRANCH IF GREATER THAN $FFFF
1737   AC3C  10 DF 17                STS   BOTSTK               CURRENT NEW BOTTOM OF STACK STACK POINTER
1738   AC3F  10 93 17                CMPD  BOTSTK               ARE WE GOING TO BE BELOW STACK?
1739   AC42  25 EE                   BCS   LAC32                YES - NO ERROR
1740   AC44  C6 0C          LAC44    LDB   #6*2                 OUT OF MEMORY ERROR
1741
1742                        * ERROR SERVICING ROUTINE
1743   AC46  BD 01 8E       LAC46    JSR   RVEC16               HOOK INTO RAM
1744   AC49  BD 01 91                JSR   RVEC17               HOOK INTO RAM
1745   AC4C  BD A7 E9                JSR   LA7E9                TURN OFF CASSETTE
1746   AC4F  BD A9 74                JSR   LA974                DISABLE ANA MUX
1747   AC52  BD AD 33                JSR   LAD33                RESET STACK, STRING STACK, CONTINUE POINTER
1748   AC55  0F 6F                   CLR   DEVNUM               SET DEVICE NUMBER TO SCREEN
1749   AC57  BD B9 5C                JSR   LB95C                SEND A CR TO SCREEN
1750   AC5A  BD B9 AF                JSR   LB9AF                SEND A  ?  TO SCREEN
1751   AC5D  8E AB AF                LDX   #LABAF               POINT TO ERROR TABLE
1752   AC60  3A                      ABX                        ADD MESSAGE NUMBER OFFSET
1753   AC61  8D 3D                   BSR   LACA0                * GET TWO CHARACTERS FROM X AND
1754   AC63  8D 3B                   BSR   LACA0                * SEND TO CONSOLE OUT (SCREEN)
1755   AC65  8E AB E0                LDX   #LABE1-1             POINT TO "ERROR" MESSAGE
1756   AC68  BD B9 9C       LAC68    JSR   LB99C                PRINT MESSAGE POINTED TO BY X
1757   AC6B  96 68                   LDA   CURLIN               GET CURRENT LINE NUMBER (CURL IN)
1758   AC6D  4C                      INCA                       TEST FOR DIRECT MODE
1759   AC6E  27 03                   BEQ   LAC73                BRANCH IF DIRECT MODE
1760   AC70  BD BD C5                JSR   LBDC5                PRINT  IN ****
1761
1762                        * THIS IS THE MAIN LOOP OF BASIC WHEN IN DIRECT MODE
1763   AC73  BD B9 5C       LAC73    JSR   LB95C                MOVE CURSOR TO START OF LINE
1764   AC76  8E AB ED                LDX   #LABE1-1             POINT X TO  OK , CR MESSAGE
1765   AC79  BD B9 9C                JSR   LB99C                PRINT  OK , CR
1766   AC7C  BD A3 90       LAC7C    JSR   LA390                GO GET AN INPUT LINE
1767   AC7F  CE FF FF                LDU   #FFFF                THE LINE NUMBER FOR DIRECT MODE IS $FFFF
1768   AC82  DF 68                   STU   CURLIN               SAVE IT IN CURLIN
1769   AC84  25 F6                   BCS   LAC7C                BRANCH IF LINE INPUT TERMINATED BY BREAK
1770   AC86  0D 70                   TST   CINBFL               CHECK CONSOLE INPUT BUFFER STATUS
1771   AC88  10 26 F8 33             LBNE  LA4BF                BRANCH IF BUFFER EMPTY - CLOSE FILE IF EMPTY
1772   AC8C  9F A6                   STX   CHARAD               SAVE (X) AS CURRENT INPUT POINTER - THIS WILL
1773                        *                                   ENABLE THE  LIVE KEYBOARD  (DIRECT) MODE. THE
1774                        *                                   LINE JUST ENTERED WILL BE INTERPRETED
1775   AC8E  9D 9F                   JSR   GETNCH               GET NEXT CHARACTER FROM BASIC
1776   AC90  27 EA                   BEQ   LAC7C                NO LINE INPUT - GET ANOTHER LINE
1777   AC92  25 11                   BCS   LACA5                BRANCH IF NUMER1C - THERE WAS A LINE NUMBER BEFORE
1778                        *                                   THE STATEMENT ENTERED, SO THIS STATEMENT
1779                        *                                   WILL BE MERGED INTO THE BASIC PROGRAM
1780   AC94  C6 30                   LDB   #2*24                 DIRECT STATEMENT IN FILE  ERROR
```

```
1781   AC96  ØD 6F              TST   DEVNUM                  * CHECK DEVICE NUMBER AND
1782   AC98  26 AC              BNE   LAC46                   * ISSUE  DS  ERROR IF DEVNUM <> Ø
1783   AC9A  BD B8 21           JSR   LB821                   GO CRUNCH LINE
1784   AC9D  7E AD CØ           JMP   LADCØ                   GO EXECUTE THE STATEMENT (LIVE KEYBOARD)
1785                    *
1786   ACAØ  A6 8Ø      LACAØ   LDA   ,X+                     GET A CHARACTER
1787   ACA2  7E B9 B1           JMP   LB9B1                   SEND TO CONSOLE OUT
1788                    * TAKE A LINE FROM THE LINE INPUT BUFFER
1789                    * AND INSERT IT INTO THE BASIC PROGRAM
1790   ACA5  BD AF 67   LACA5   JSR   LAF67                   CONVERT LINE NUMBER TO BINARY
1791   ACA8  9E 2B              LDX   BINVAL                  GET CONVERTED LINE NUMBER
1792   ACAA  BF Ø2 DA           STX   LINHDR                  STORE IT IN LINE INPUT HEADER
1793   ACAD  BD B8 21           JSR   LB821                   GO CRUNCH THE LINE
1794   ACBØ  D7 Ø3              STB   TMPLOC                  SAVE LINE LENGTH
1795   ACB2  8D 4D              BSR   LADØ1                   FIND OUT WHERE TO INSERT LINE
1796   ACB4  25 12              BCS   LACC8                   BRANCH IF LINE NUMBER DOES NOT ALREADY EXIST
1797   ACB6  DC 47              LDD   V47                     GET ABSOLUTE ADDRESS OF LINE NUMBER
1798   ACB8  A3 84              SUBD  ,X                      SUBTRACT ADDRESS OF NEXT LINE NUMBER
1799   ACBA  D3 1B              ADDD  VARTAB                  * ADD TO CURRENT END OF PROGRAM - THIS WILL REMOVE
18ØØ   ACBC  DD 1B              STD   VARTAB                  * THE LENGTH OF THIS LINE NUMBER FROM THE PROGRAM
18Ø1   ACBE  EE 84              LDU   ,X                      POINT U TO ADDRESS OF NEXT LINE NUMBER
18Ø2                    * DELETE OLD LINE FROM BASIC PROGRAM
18Ø3   ACCØ  37 Ø2      LACCØ   PULU  A                       GET A BYTE FROM WHAT S LEFT OF PROGRAM
18Ø4   ACC2  A7 8Ø              STA   ,X+                     MOVE IT DOWN
18Ø5   ACC4  9C 1B              CMPX  VARTAB                  COMPARE TO END OF BASIC PROGRAM
18Ø6   ACC6  26 F8              BNE   LACCØ                   BRANCH IF NOT AT END
18Ø7   ACC8  B6 Ø2 DC    LACC8   LDA   LINBUF                  * CHECK TO SEE IF THERE IS A LINE IN
18Ø8   ACCB  27 1C              BEQ   LACE9                   * THE BUFFER AND BRANCH IF NONE
18Ø9   ACCD  DC 1B              LDD   VARTAB                  = SAVE CURRENT END OF
1810   ACCF  DD 43              STD   V43                     = PROGRAM IN V43
1811   ACD1  DB Ø3              ADDB  TMPLOC                  * ADD LENGTH OF CRUNCHED LINE,
1812   ACD3  89 ØØ              ADCA  #Ø                      * PROPOGATE CARRY AND SAVE NEW END
1813   ACD5  DD 41              STD   V41                     * OF PROGRAM IN V41
1814   ACD7  BD AC 1E           JSR   LAC1E                   = MAKE SURE THERE S ENOUGH RAM FOR THIS
1815                    *                                     = LINE & MAKE A HOLE IN BASIC FOR NEW LINE
1816   ACDA  CE Ø2 D8           LDU   #LINHDR-2               POINT U TO LINE TO BE INSERTED
1817   ACDD  37 Ø2      LACDD   PULU  A                       GET A BYTE FROM NEW LINE
1818   ACDF  A7 8Ø              STA   ,X+                     INSERT IT IN PROGRAM
1819   ACE1  9C 45              CMPX  V45                     * COMPARE TO ADDRESS OF END OF INSERTED
182Ø   ACE3  26 F8              BNE   LACDD                   * LINE AND BRANCH IF NOT DONE
1821   ACE5  9E 41              LDX   V41                     = GET AND SAVE
1822   ACE7  9F 1B              STX   VARTAB                  = END OF PROGRAM
1823   ACE9  8D 36      LACE9   BSR   LAD21                   RESET INPUT POINTER, CLEAR VARIABLES, INITIALIZE
1824   ACEB  8D Ø2              BSR   LACEF                   ADJUST START OF NEXT LINE ADDRESSES
1825   ACED  2Ø 8D              BRA   LAC7C                   REENTER BASIC S INPUT LOOP
1826                    * COMPUTE THE START OF NEXT LINE ADDRESSES FOR THE BASIC PROGRAM
1827   ACEF  9E 19      LACEF   LDX   TXTTAB                  POINT X TO START OF PROGRAM
1828   ACF1  EC 84      LACF1   LDD   ,X                      GET ADDRESS OF NEXT LINE
1829   ACF3  27 21              BEQ   LAD16                   RETURN IF END OF PROGRAM
183Ø   ACF5  33 Ø4              LEAU  4,X                     POINT U TO START OF BASIC TEXT IN LINE
1831   ACF7  A6 CØ      LACF7   LDA   ,U+                     * SKIP THROUGH THE LINE UNTIL A
1832   ACF9  26 FC              BNE   LACF7                   * ZERO (END OF LINE) IS FOUND
1833   ACFB  EF 84              STU   ,X                      SAVE THE NEW START OF NEXT LINE ADDRESS
1834   ACFD  AE 84              LDX   ,X                      POINT X TO START OF NEXT LINE
1835   ACFF  2Ø F0              BRA   LACF1                   KEEP GOING
1836                    *
1837                    * FIND A LINE NUMBER IN THE BASIC PROGRAM
1838                    * RETURN WITH CARRY SET IF NO MATCH FOUND
1839   ADØ1  DC 2B      LADØ1   LDD   BINVAL                  GET THE LINE NUMBER TO FIND
184Ø   ADØ3  9E 19              LDX   TXTTAB                  BEGINNING OF PROGRAM
1841   ADØ5  EE 84      LADØ5   LDU   ,X                      GET ADDRESS OF NEXT LINE NUMBER
1842   ADØ7  27 Ø9              BEQ   LAD12                   BRANCH IF END OF PROG
1843   ADØ9  1Ø A3 Ø2           CMPD  2,X                     IS IT A MATCH?
1844   ADØC  23 Ø6              BLS   LAD14                   CARRY SET IF LOWER; CARRY CLEAR IF MATCH
1845   ADØE  AE 84              LDX   ,X                      X = ADDRESS OF NEXT LINE
1846   AD1Ø  2Ø F3              BRA   LADØ5                   KEEP LOOPING FOR LINE NUMBER
1847   AD12  1A Ø1      LAD12   ORCC  #1                      SET CARRY FLAG
1848   AD14  9F 47      LAD14   STX   V47                     SAVE MATCH LINE NUMBER OR NUMBER OF LINE JUST AFTER
1849                                                          WHERE IT SHOULD HAVE BEEN
185Ø   AD16  39        LAD16   RTS
1851
1852                    * NEW
1853   AD17  26 FB      NEW     BNE   LAD14                   BRANCH IF ARGUMENT GIVEN
1854   AD19  9E 19      LAD19   LDX   TXTTAB                  GET START OF BASIC
1855   AD1B  6F 8Ø              CLR   ,X+                     * PUT 2 ZERO BYTES THERE - ERASE
1856   AD1D  6F 8Ø              CLR   ,X+                     * THE BASIC PROGRAM
1857   AD1F  9F 1B              STX   VARTAB                  AND THE NEXT ADDRESS IS NOW THE END OF PROGRAM
1858   AD21  9E 19      LAD21   LDX   TXTTAB                  GET START OF BASIC
1859   AD23  BD AE BB           JSR   LAEBB                   PUT INPUT POINTER ONE BEFORE START OF BASIC
1860                    * ERASE ALL VARIABLES
1861   AD26  9E 27      LAD26   LDX   MEMSIZ                  * RESET START OF STRING VARIABLES
1862   AD28  9F 23              STX   STRTAB                  * TO TOP OF STRING SPACE
1863   AD2A  BD AD E4           JSR   RESTOR                  RESET  DATA  POINTER TO START OF BASIC
1864   AD2D  9E 1B              LDX   VARTAB                  * GET START OF VARIABLES AND USE IT
1865   AD2F  9F 1D              STX   ARYTAB                  * TO RESET START OF ARRAYS
1866   AD31  9F 1F              STX   ARYEND                  RESET END OF ARRAYS
1867   AD33  8E Ø1 A9   LAD33   LDX   #STRSTK                 * RESET STRING STACK POINTER TO
1868   AD36  9F ØB              STX   TEMPPT                  * BOTTOM OF STRING STACK
1869   AD38  AE E4              LDX   ,S                      GET RETURN ADDRESS OFF STACK
```

```
1870   AD3A  10 DE 21                  LDS   FRETOP                  RESTORE STACK POINTER
1871   AD3D  6F E2                     CLR   ,-S                     PUT A ZERO BYTE ON STACK - TO CLEAR ANY RETURN OF
1872                                  *                              FOR/NEXT DATA FROM THE STACK
1873   AD3F  0F 2D                     CLR   OLDPTR                  RESET  CONT  ADDRESS SO YOU
1874   AD41  0F 2E                     CLR   OLDPTR+1                 CAN T CONTINUE
1875   AD43  0F 08                     CLR   ARYDIS                  CLEAR THE ARRAY DISABLE FLAG
1876   AD45  6E 84                     JMP   ,X                      RETURN TO CALLING ROUTINE - THIS IS NECESSARY
1877                                  *                              SINCE THE STACK WAS RESET
1878                                  *
1879                                  * FOR
1880                                  *
1881                                  * THE FOR COMMAND WILL STORE 18 BYTES ON THE STACK FOR
1882                                  * EACH FOR-NEXT LOOP WHICH IS BEING PROCESSED. THESE
1883                                  * BYTES ARE DEFINED AS FOLLOWS: 0- $80 (FOR FLAG);
1884                                  *   1,2=INDEX VARIABLE DESCRIPTOR POINTER; 3-7=FP VALUE OF STEP;
1885                                  *     8=STEP DIRECTION: $FF IF NEGATIVE; 0 IF ZERO; 1 IF POSITIVE;
1886                                  *  9-13=FP VALUE OF  TO  PARAMETER;
1887                                  * 14,15=CURRENT LINE NUMBER; 16,17=RAM ADDRESS OF THE END
1888                                  *       OF THE LINE CONTAINING THE  FOR  STATEMENT
1889   AD47  86 80                  FOR      LDA   #$80                  * SAVE THE DISABLE ARRAY FLAG IN VO8
1890   AD49  97 08                     STA   ARYDIS                  * DO NOT ALLOW THE INDEX VARIABLE TO BE AN ARRAY
1891   AD4B  BD AF 89                  JSR   LET                     SET INDEX VARIABLE TO INITIAL VALUE
1892   AD4E  BD AB F9                  JSR   LABF9                   SEARCH THE STACK FOR  FOR/NEXT  DATA
1893   AD51  32 62                     LEAS  2,S                     PURGE RETURN ADDRESS OFF OF THE STACK
1894   AD53  26 04                     BNE   LAD59                   BRANCH IF INDEX VARIABLE NOT ALREADY BEING USED
1895   AD55  9E 0F                     LDX   TEMPTR                  GET (ADDRESS + 18) OF MATCHED  FOR/NEXT  DATA
1896   AD57  32 85                     LEAS  B,X                     MOVE THE STACK POINTER TO THE BEGINNING OF THE
1897                                  * MATCHED  FOR/NEXT  DATA SO THE NEW DATA WILL
1898                                  * OVERLAY THE OLD DATA. THIS WILL ALSO DESTROY
1899                                  * ALL OF THE  RETURN  AND  FOR/NEXT  DATA BELOW
1900                                  * THIS POINT ON THE STACK
1901   AD59  C6 09                  LAD59    LDB   #$09                  * CHECK FOR ROOM FOR 18 BYTES
1902   AD5B  BD AC 33                  JSR   LAC33                   * IN FREE RAM
1903   AD5E  BD AE E8                  JSR   LAEE8                   GET ADDR OF END OF SUBLINE IN X
1904   AD61  DC 68                     LDD   CURLIN                  GET CURRENT LINE NUMBER
1905   AD63  34 16                     PSHS  X,B,A                   SAVE LINE ADDR AND LINE NUMBER ON STACK
1906   AD65  C6 A5                     LDB   #$A5                    TOKEN FOR  TO
1907   AD67  BD B2 6F                  JSR   LB26F                   SYNTAX CHECK FOR  TO
1908   AD6A  BD B1 43                  JSR   LB143                    TM  ERROR IF INDEX VARIABLE SET TO STRING
1909   AD6D  BD B1 41                  JSR   LB141                   EVALUATE EXPRESSION
1910                                  *
1911   AD70  D6 54                     LDB   FP0SGN                  GET FPA0 MANTISSA SIGN
1912   AD72  CA 7F                     ORB   #$7F                    FORM A MASK TO SAVE DATA BITS OF HIGH ORDER MANTISSA
1913   AD74  D4 50                     ANDB  FPA0                    PUT THE MANTISSA SIGN IN BIT 7 OF HIGH ORDER MANTISSA
1914   AD76  D7 50                     STB   FPA0                    SAVE THE PACKED HIGH ORDER MANTISSA
1915   AD78  10 8E AD 7F               LDY   #LAD7F                 LOAD FOLLOWING ADDRESS INTO Y AS A RETURN
1916   AD7C  7E B1 EA                  JMP   LB1EA                   ADDRESS - PUSH FPA0 ONTO THE STACK
1917   AD7F  8E BA C5                LAD7F    LDX   #LBAC5                 POINT X TO FLOATING POINT NUMBER 1.0 (DEFAULT STEP VALUE)
1918   AD82  BD BC 14                  JSR   LBC14                   MOVE (X) TO FPA0
1919   AD85  9D A5                     JSR   GETCCH                  GET CURRENT INPUT CHARACTER
1920   AD87  81 A9                     CMPA  #$A9                    STEP TOKEN
1921   AD89  26 05                     BNE   LAD90                   BRANCH IF NO  STEP  VALUE
1922   AD8B  9D 9F                     JSR   GETNCH                  GET A CHARACTER FROM BASIC
1923   AD8D  BD B1 41                  JSR   LB141                   EVALUATE NUMERIC EXPRESSION
1924   AD90  BD BC 6D                LAD90    JSR   LBC6D                 CHECK STATUS OF FPA0
1925   AD93  BD B1 E6                  JSR   LB1E6                   SAVE STATUS AND FPA0 ON THE STACK
1926   AD96  DC 3B                     LDD   VARDES                  * GET DESCRIPTOR POINTER FOR THE  STEP
1927   AD98  34 06                     PSHS  B,A                     * VARIABLE AND SAVE IT ON THE STACK
1928   AD9A  86 80                     LDA   #$80                    = GET THE  FOR  FLAG AND
1929   AD9C  34 02                     PSHS  A                       = SAVE IT ON THE STACK
1930                                  *
1931                                  * MAIN COMMAND INTERPRETATION LOOP
1932   AD9E  BD 01 9A                LAD9E    JSR   RVEC20                  HOOK INTO RAM
1933   ADA1  1C AF                     ANDCC #$AF                    ENABLE IRQ,FIRQ
1934   ADA3  8D 46                     BSR   LADEB                   CHECK FOR KEYBOARD BREAK
1935   ADA5  9E A6                     LDX   CHARAD                  GET BASIC S INPUT POINTER
1936   ADA7  9F 2F                     STX   TINPTR                  SAVE IT
1937   ADA9  A6 80                     LDA   ,X+                     GET CURRENT INPUT CHAR & MOVE POINTER
1938   ADAB  27 07                     BEQ   LADB4                   BRANCH IF END OF LINE
1939   ADAD  81 3A                     CMPA  #':                     CHECK FOR LINE SEPARATOR
1940   ADAF  27 0F                     BEQ   LADC0                   BRANCH IF COLON
1941   ADB1  7E B2 77                LADB1    JMP   LB277                    SYNTAX ERROR -IF NOT LINE SEPARATOR
1942   ADB4  A6 81                   LADB4    LDA   ,X++                    GET MS BYTE OF ADDRESS OF NEXT BASIC LINE
1943   ADB6  97 00                     STA   ENDFLG                  SAVE IN STOP/END FLAG - CAUSE A STOP IF
1944                                  *                              NEXT LINE ADDRESS IS < $8000; CAUSE
1945                                  *                              AN END IF ADDRESS > $8000
1946   ADB8  27 5B                     BEQ   LAE15                   BRANCH TO  STOP  - END OF PROGRAM
1947   ADBA  EC 80                     LDD   ,X+                     GET CURRENT LINE NUMBER
1948   ADBC  DD 68                     STD   CURLIN                  SAVE IN CURLIN
1949   ADBE  9F A6                     STX   CHARAD                  SAVE ADDRESS OF FIRST BYTE OF LINE
1950   ADC0  9D 9F                   LADC0    JSR   GETNCH                  GET A CHARACTER FROM BASIC
1951   ADC2  8D 02                     BSR   LADC6                   GO PROCESS COMMAND
1952   ADC4  20 D8                     BRA   LAD9E                   GO BACK TO MAIN LOOP
1953   ADC6  27 78                   LADC6    BEQ   LAE40                  RETURN IF END OF LINE
1954   ADC8  4D                        TSTA                          CHECK FOR TOKEN - BIT 7 SET (NEGATIVE)
1955   ADC9  10 2A 01 BC               LBPL  LET                     BRANCH IF NOT A TOKEN - GO DO A  LET  WHICH
1956                                  *                              IS THE  DEFAULT  TOKEN FOR MICROSOFT BASIC
1957   ADCD  81 A3                     CMPA  #$A3                    SKIPF TOKEN - HIGHEST EXECUTABLE COMMAND IN BASIC
1958   ADCF  22 0B                     BHI   LADDC                   BRANCH IF > A BASIC COMMAND
```

```
1959   ADD1  BE Ø1 23              LDX   COMVEC+3              GET ADDRESS OF BASIC S COMMAND TABLE
196Ø   ADD4  48             LADD4  ASLA                        X2 (2 BYTE/JUMP ADDRESS) & DISCARD BIT 7
1961   ADD5  1F 89                 TFR   A,B                   SAVE COMMAND OFFSET IN ACCB
1962   ADD7  3A                    ABX                         NON X POINTS TO COMMAND JUMP ADDR
1963   ADD8  9D 9F                 JSR   GETNCH                GET AN INPUT CHAR
1964                        *
1965                        * HERE IS WHERE WE BRANCH TO DO A  COMMAND
1966   ADDA  6E 94                  JMP   [,X]                  GO DO A COMMAND
1967   ADDC  81 B4           LADDC  CMPA  #$B4                  $B4 IS HIGHEST BASIC TOKEN
1968   ADDE  23 D1                  BLS   LADB1                  SYNTAX ERROR  IF NON-EXECUTABLE TOKEN
1969   ADEØ  6E 9F Ø1 2D            JMP   [COMVEC+13]           JUMP TO AN EX BAS COMMAND
197Ø                        *
1971                        * RESTORE
1972   ADE4  9E 19           RESTOR LDX   TXTTAB                BEGINNING OF PROGRAM ADDRESS
1973   ADE6  3Ø 1F                  LEAX  -1,X                  MOVE TO ONE BYTE BEFORE PROGRAM
1974   ADE8  9F 33           LADE8  STX   DATPTR                SAVE NEW DATA POINTER
1975   ADEA  39                     RTS
1976                        *
1977                        * BREAK CHECK
1978   ADEB  BD A1 C1        LADEB  JSR   LA1C1                 GET A KEYSTROKE ENTRY
1979   ADEE  27 ØA                  BEQ   LADFA                 RETURN IF NO INPUT
198Ø   ADFØ  81 Ø3           LADFØ  CMPA  #3                    CONTROL C? (BREAK)
1981   ADF2  27 15                  BEQ   STOP                  YES
1982   ADF4  81 13                  CMPA  #$13                  CONTROL S? (PAUSE)
1983   ADF6  27 Ø3                  BEQ   LADFB                 YES
1984   ADF8  97 87                  STA   IKEYIM                SAVE KEYSTROKE IN INKEY IMAGE
1985   ADFA  39              LADFA  RTS
1986   ADFB  BD A1 CB        LADFB  JSR   KEYIN                 GET A KEY
1987   ADFE  27 FB                  BEQ   LADFB                 BRANCH IF NO KEY DOWN
1988   AEØØ  2Ø EE                  BRA   LADFØ                 CONTINUE - DO A BREAK CHECK
1989                        *
199Ø                        * END
1991   AE02  BD A4 26        END    JSR   LA426                 CLOSE FILES
1992   AE05  9D A5                  JSR   GETCCH                GET CURRENT INPUT CHAR
1993   AE07  2Ø Ø2                  BRA   LAEØB
1994                        *
1995                        * STOP
1996   AE09  1A Ø1           STOP   ORCC  #$Ø1                  SET CARRY FLAG
1997   AE0B  26 33           LAEØB  BNE   LAE4Ø                 BRANCH IF ARGUMENT EXISTS
1998   AE0D  9E A6                  LDX   CHARAD                * SAVE CURRENT POSITION OF
1999   AE0F  9F 2F                  STX   TINPTR                * BASIC S INPUT POINTER
2ØØØ   AE11  Ø6 ØØ           LAE11  ROR   ENDFLG                ROTATE CARRY INTO BIT 7 OF STOP/END FLAG
2ØØ1   AE13  32 62                  LEAS  2,S                   PURGE RETURN ADDRESS OFF STACK
2ØØ2   AE15  9E 68           LAE15  LDX   CURLIN                GET CURRENT LINE NUMBER
2ØØ3   AE17  8C FF FF                CMPX  #$FFFF                DIRECT MODE?
2ØØ4   AE1A  27 Ø6                   BEQ   LAE22                 YES
2ØØ5   AE1C  9F 29                   STX   OLDTXT                SAVE CURRENT LINE NUMBER
2ØØ6   AE1E  9E 2F                   LDX   TINPTR                * GET AND SAVE CURRENT POSITION
2ØØ7   AE2Ø  9F 2D                   STX   OLDPTR                * OF BASIC S INPUT POINTER
2ØØ8   AE22  ØF 6F           LAE22  CLR   DEVNUM                SET DEVICE NUMBER TO SCREEN
2ØØ9   AE24  8E AB F1                LDX   #LABF2-1              POINT TO CR,  BREAK  MESSAGE
2Ø1Ø   AE27  ØD ØØ                   TST   ENDFLG                CHECK STOP/END FLAG
2Ø11   AE29  1Ø 2A FE 46             LBPL  LAC73                 BRANCH TO MAIN LOOP OF BASIC IF END
2Ø12   AE2D  7E AC 68                JMP   LAC68                 PRINT  BREAK AT ####  AND GO TO
2Ø13                        *                                   BASIC S MAIN LOOP IF  STOP
2Ø14
2Ø15                        * CONT
2Ø16   AE3Ø  26 ØE           CONT   BNE   LAE4Ø                 RETURN IF ARGUMENT GIVEN
2Ø17   AE32  C6 2Ø                   LDB   #2*16                  CAN T CONTINUE  ERROR
2Ø18   AE34  9E 2D                   LDX   OLDPTR                GET CONTINUE ADDRESS (INPUT POINTER)
2Ø19   AE36  1Ø 27 FE ØC             LBEQ  LAC46                  CN  ERROR IF CONTINUE ADDRESS = Ø
2Ø2Ø   AE3A  9F A6                   STX   CHARAD                RESET BASIC S INPUT POINTER
2Ø21   AE3C  9E 29                   LDX   OLDTXT                GET LINE NUMBER
2Ø22   AE3E  9F 68                   STX   CURLIN                RESET CURRENT LINE NUMBER
2Ø23   AE4Ø  39              LAE4Ø  RTS
2Ø24                        *
2Ø25                        * CLEAR
2Ø26   AE41  27 2C           CLEAR  BEQ   LAE6F                 BRANCH IF NO ARGUMENT
2Ø27   AE43  BD B3 E6                JSR   LB3E6                 EVALUATE ARGUMENT
2Ø28   AE46  34 Ø6                   PSHS  B,A                   SAVE AMOUNT OF STRING SPACE ON STACK
2Ø29   AE48  9E 27                   LDX   MEMSIZ                GET CURRENT TOP OF CLEARED SPACE
2Ø3Ø   AE4A  9D A5                   JSR   GETCCH                GET CURRENT INPUT CHARACTER
2Ø31   AE4C  27 ØC                   BEQ   LAE5A                 BRANCH IF NO NEW TOP OF CLEARED SPACE
2Ø32   AE4E  BD B2 6D                JSR   LB26D                 SYNTAX CHECK FOR COMMA
2Ø33   AE51  BD B7 3D                JSR   LB73D                 EVALUATE EXPRESSION; RETURN VALUE IN X
2Ø34   AE54  3Ø 1F                   LEAX  -1,X                  X = TOP OF CLEARED SPACE
2Ø35   AE56  9C 74                   CMPX  TOPRAM                COMPARE TO TOP OF RAM
2Ø36   AE58  22 18                   BHI   LAE72                  OM  ERROR IF > TOP OF RAM
2Ø37   AE5A  1F 1Ø           LAE5A  TFR   X,D                   ACCD = TOP OF CLEARED SPACE
2Ø38   AE5C  A3 E1                   SUBD  ,S++                  SUBTRACT OUT AMOUNT OF CLEARED SPACE
2Ø39   AE5E  25 12                   BCS   LAE72                  OM  ERROR IF FREE MEM < Ø
2Ø4Ø   AE6Ø  1F Ø3                   TFR   D,U                   U = BOTTOM OF CLEARED SPACE
2Ø41   AE62  83 ØØ 3A                SUBD  #STKBUF               SUBTRACT OUT STACK BUFFER
2Ø42   AE65  25 ØB                   BCS   LAE72                  OM  ERROR IF FREE MEM < Ø
2Ø43   AE67  93 1B                   SUBD  VARTAB                SUBTRACT OUT START OF VARIABLES
2Ø44   AE69  25 Ø7                   BCS   LAE72                  OM  ERROR IF FREE MEM < Ø
2Ø45   AE6B  DF 21                   STU   FRETOP                SAVE NEW BOTTOM OF CLEARED SPACE
2Ø46   AE6D  9F 27                   STX   MEMSIZ                SAVE NEW TOP OF CLEARED SPACE
2Ø47   AE6F  7E AD 26        LAE6F  JMP   LAD26                 ERASE ALL VARIABLES, INITIALIZE POINTERS, ETC
```

```
2048   AE72  7E AC 44        LAE72    JMP   LAC44                    OM  ERROR
2049                         *
2050                         * RUN
2051   AE75  BD Ø1 94        RUN      JSR   RVEC18                   HOOK INTO RAM
2052   AE78  BD A4 26                  JSR   LA426                    CLOSE ANY OPEN FILES
2053   AE7B  9D A5                     JSR   GETCCH                   * GET CURRENT INPUT CHARACTER
2054   AE7D  1Ø 27 FE AØ               LBEQ  LAD21                    * IF NO LINE NUMBER
2055   AE81  BD AD 26                  JSR   LAD26                    ERASE ALL VARIABLES
2056   AE84  2Ø 19                     BRA   LAE9F                     GOTO  THE RUN ADDRESS
2057                         *
2058                         * GO
2059   AE86  1F 89            GO       TFR   A,B                      SAVE INPUT CHARACTER IN ACCB
2060   AE88  9D 9F            LAE88    JSR   GETNCH                   GET A CHARACTER FROM BASIC
2061   AE8A  C1 A5                     CMPB  #$A5                     TO  TOKEN
2062   AE8C  27 16                     BEQ   LAEA4                    BRANCH IF GOTO
2063   AE8E  C1 A6                     CMPB  #$A6                     SUB  TOKEN
2064   AE9Ø  26 45                     BNE   LAED7                    SYNTAX ERROR  IF NEITHER
2065   AE92  C6 Ø3                     LDB   #3                       =ROOM FOR 6
2066   AE94  BD AC 33                  JSR   LAC33                    =BYTES ON STACK?
2067   AE97  DE A6                     LDU   CHARAD                   * SAVE CURRENT BASIC INPUT POINTER, LINE
2068   AE99  9E 68                     LDX   CURLIN                   * NUMBER AND SUB TOKEN ON STACK
2069   AE9B  86 A6                     LDA   #$A6                     *
2070   AE9D  34 52                     PSHS  U,X,A                    *
2071   AE9F  8D Ø3            LAE9F    BSR   LAEA4                    GO DO A  GOTO
2072   AEA1  7E AD 9E                  JMP   LAD9E                    JUMP BACK TO BASIC S MAIN LOOP
2073                         * GOTO
2074   AEA4  9D A5            LAEA4    JSR   GETCCH                   GET CURRENT INPUT CHAR
2075   AEA6  BD AF 67                  JSR   LAF67                    GET LINE NUMBER TO BINARY IN BINVAL
2076   AEA9  8D 4Ø                     BSR   LAEEB                    ADVANCE BASIC S POINTER TO END OF LINE
2077   AEAB  3Ø Ø1                     LEAX  $Ø1,X                    POINT TO START OF NEXT LINE
2078   AEAD  DC 2B                     LDD   BINVAL                   GET THE LINE NUMBER TO RUN
2079   AEAF  1Ø 93 68                  CMPD  CURLIN                   COMPARE TO CURRENT LINE NUMBER
2080   AEB2  22 Ø2                     BHI   LAEB6                    IF REO D LINE NUMBER IS > CURRENT LINE NUMBER,
2081                         *                                       DON T START LOOKING FROM
2082                         *                                       START OF PROGRAM
2083   AEB4  9E 19                     LDX   TXTTAB                   BEGINNING OF PROGRAM
2084   AEB6  BD AD Ø5         LAEB6    JSR   LAD05                    GO FIND A LINE NUMBER
2085   AEB9  25 17                     BCS   LAED2                     UNDEFINED LINE NUMBER
2086   AEBB  3Ø 1F            LAEBB    LEAX  -1,X                     MOVE BACK TO JUST BEFORE START OF LINE
2087   AEBD  9F A6                     STX   CHARAD                   RESET BASIC S INPUT POINTER
2088   AEBF  39               LAEBF    RTS
2089                         *
2090                         * RETURN
2091   AEC0  26 FD            RETURN   BNE   LAEBF                    EXIT ROUTINE IF ARGUMENT GIVEN
2092   AEC2  86 FF                     LDA   #$FF                     * PUT AN ILLEGAL VARIABLE NAME IN FIRST BYTE OF
2093   AEC4  97 3B                     STA   VARDES                   * VARDES WHICH WILL CAUSE  FOR/NEXT  DATA ON THE
2094                         *                                       STACK TO BE IGNORED
2095   AEC6  BD AB F9                  JSR   LABF9                    CHECK FOR RETURN DATA ON THE STACK
2096   AEC9  1F 14                     TFR   X,S                      RESET STACK POINTER - PURGE TWO RETURN ADDRESSES
2097                         *                                       FROM THE STACK
2098   AECB  81 26                     CMPA  #$A6-$8Ø                 SUB TOKEN - $8Ø
2099   AECD  27 ØB                     BEQ   LAEDA                    BRANCH IF  RETURN  FROM SUBROUTINE
2100   AECF  C6 Ø4                     LDB   #2*2                     ERROR #2  RETURN WITHOUT GOSUB
2101   AED1  8C                        FCB   SKP2                     SKIP TWO BYTES
2102   AED2  C6 ØE            LAED2    LDB   #7*2                     ERROR #7  UNDEFINED LINE NUMBER
2103   AED4  7E AC 46                  JMP   LAC46                    JUMP TO ERROR HANDLER
2104   AED7  7E B2 77         LAED7    JMP   LB277                     SYNTAX ERROR
2105   AEDA  35 52            LAEDA    PULS  A,X,U                    * RESTORE VALUES OF CURRENT LINE NUMBER AND
2106   AEDC  9F 68                     STX   CURLIN                   * BASIC S INPUT POINTER FOR THIS SUBROUTINE
2107   AEDE  DF A6                     STU   CHARAD                   * AND LOAD ACCA WITH SUB TOKEN ($A6)
2108                         *
2109                         * DATA
2110   AEE0  8D Ø6            DATA     BSR   LAEE8                    MOVE INPUT POINTER TO END OF SUBLINE OR LINE
2111   AEE2  8C                        FCB   SKP2                     SKIP 2 BYTES
2112
2113                         * REM, ELSE
2114                         ELSE
2115   AEE3  8D Ø6            REM      BSR   LAEEB                    MOVE INPUT POINTER TO END OF LINE
2116   AEE5  9F A6                     STX   CHARAD                   RESET BASIC S INPUT POINTER
2117   AEE7  39               LAEE7    RTS
2118                         * ADVANCE INPUT POINTER TO END OF SUBLINE OR LINE
2119   AEE8  C6 3A            LAEE8    LDB   #':                      COLON = SUBLINE TERMINATOR CHARACTER
2120   AEEA  86               LAEEA    FCB   SKP1LD                   SKPILD SKIP ONE BYTE; LDA #$5F
2121                         * ADVANCE BASIC S INPUT POINTER TO END OF
2122                         * LINE - RETURN ADDRESS OF END OF LINE+1 IN X
2123   AEEB  5F               LAEEB    CLRB                           Ø = LINE TERMINATOR CHARACTER
2124   AEEC  D7 Ø1                     STB   CHARAC                   TEMP STORE PRIMARY TERMINATOR CHARACTER
2125   AEEE  5F                        CLRB                           Ø (END OF LINE) = ALTERNATE TERM. CHAR.
2126   AEEF  9E A6                     LDX   CHARAD                   LOAD X W/BASIC S INPUT POINTER
2127   AEF1  1F 98            LAEF1    TFR   B,A                      * CHANGE TERMINATOR CHARACTER
2128   AEF3  D6 Ø1                     LDB   CHARAC                   * FROM ACCB TO CHARAC - SAVE OLD TERMINATOR
2129                         *                                       IN CHARAC
2130   AEF5  97 Ø1                     STA   CHARAC                   SWAP PRIMARY AND SECONDARY TERMINATORS
2131   AEF7  A6 84            LAEF7    LDA   ,X                       GET NEXT INPUT CHARACTER
2132   AEF9  27 EC                     BEQ   LAEE7                    RETURN IF Ø (END OF LINE)
2133   AEFB  34 Ø4                     PSHS  B                        SAVE TERMINATOR ON STACK
2134   AEFD  A1 E0                     CMPA  ,S+                      COMPARE TO INPUT CHARACTER
2135   AEFF  27 E6                     BEQ   LAEE7                    RETURN IF EQUAL
2136   AF01  3Ø Ø1                     LEAX  1,X                      MOVE POINTER UP ONE
```

```
2137  AF03  81 22                    CMPA  #'"                  CHECK FOR DOUBLE QUOTES
2138  AF05  27 EA                    BEQ   LAEF1                BRANCH IF " - TOGGLE TERMINATOR CHARACTERS
2139  AF07  4C                       INCA                       * CHECK FOR $FF AND BRANCH IF
2140  AF08  26 02                    BNE   LAF0C                * NOT SECONDARY TOKEN
2141  AF0A  30 01                    LEAX  1,X                  MOVE INPUT POINTER 1 MORE IF SECONDARY
2142  AF0C  81 86             LAF0C  CMPA  #$85+1               TOKEN FOR IF?
2143  AF0E  26 E7                    BNE   LAEF7                NO - GET ANOTHER INPUT CHARACTER
2144  AF10  0C 04                    INC   IFCTR                INCREMENT IF COUNTER - KEEP TRACK OF HOW MANY
2145                          *                                  IF  STATEMENTS ARE NESTED IN ONE LINE
2146  AF12  20 E3                    BRA   LAEF7                GET ANOTHER INPUT CHARACTER
2147
2148                          * IF
2149  AF14  BD B1 41          IF     JSR   LB141                EVALUATE NUMERIC EXPRESSION
2150  AF17  9D A5                    JSR   GETCCH               GET CURRENT INPUT CHARACTER
2151  AF19  81 81                    CMPA  #$81                 TOKEN FOR GO
2152  AF1B  27 05                    BEQ   LAF22                TREAT  GO  THE SAME AS  THEN
2153  AF1D  C6 A7                    LDB   #$A7                 TOKEN FOR THEN
2154  AF1F  BD B2 6F                 JSR   LB26F                DO A SYNTAX CHECK ON ACCB
2155  AF22  96 4F             LAF22  LDA   FP0EXP               CHECK FOR TRUE/FALSE - FALSE IF FPA0 EXPONENT = ZERO
2156  AF24  26 13                    BNE   LAF39                BRANCH IF CONDITION TRUE
2157  AF26  0F 04                    CLR   IFCTR                CLEAR FLAG - KEEP TRACK OF WHICH NESTED ELSE STATEMENT
2158                          *                                  TO SEARCH FOR IN NESTED  IF  LOOPS
2159  AF28  8D B6             LAF28  BSR   DATA                 MOVE BASIC S POINTER TO END OF SUBLINE
2160  AF2A  4D                       TSTA                       * CHECK TO SEE IF END OF LINE OR SUBLINE
2161  AF2B  27 BA                    BEQ   LAEE7                * AND RETURN IF END OF LINE
2162  AF2D  9D 9F                    JSR   GETNCH               GET AN INPUT CHARACTER FROM BASIC
2163  AF2F  81 84                    CMPA  #$84                 TOKEN FOR ELSE
2164  AF31  26 F5                    BNE   LAF28                IGNORE ALL DATA EXCEPT  ELSE  UNTIL
2165                          *                                  END OF LINE (ZERO BYTE)
2166  AF33  0A 04                    DEC   IFCTR                CHECK TO SEE IF YOU MUST SEARCH ANOTHER SUBLINE
2167  AF35  2A F1                    BPL   LAF28                BRANCH TO SEARCH ANOTHER SUBLINE FOR  ELSE
2168  AF37  9D 9F                    JSR   GETNCH               GET AN INPUT CHARACTER FROM BASIC
2169  AF39  9D A5             LAF39  JSR   GETCCH               GET CURRENT INPUT CHARACTER
2170  AF3B  10 25 FF 65              LBCS  LAEA4                BRANCH TO  GOTO  IF NUMERIC CHARACTER
2171  AF3F  7E AD C6                 JMP   LADC6                RETURN TO MAIN INTERPRETATION LOOP
2172
2173                          * ON
2174  AF42  BD B7 0B          ON     JSR   LB70B                EVALUATE EXPRESSION
2175  AF45  C6 81                    LDB   #$81                 TOKEN FOR GO
2176  AF47  BD B2 6F                 JSR   LB26F                SYNTAX CHECK FOR GO
2177  AF4A  34 02                    PSHS  A                    SAVE NEW TOKEN (TO,SUB)
2178  AF4C  81 A6                    CMPA  #$A6                 TOKEN FOR SUB?
2179  AF4E  27 04                    BEQ   LAF54                YES
2180  AF50  81 A5                    CMPA  #$A5                 TOKEN FOR TO?
2181  AF52  26 83             LAF52  BNE   LAED7                 SYNTAX  ERROR IF NOT  SUB  OR  TO
2182  AF54  0A 53             LAF54  DEC   FPA0+3               DECREMENT IS BYTE OF MANTISSA OF FPA0 - THIS
2183                          *                                  IS THE ARGUMENT OF THE  ON  STATEMENT
2184  AF56  26 05                    BNE   LAF5D                BRANCH IF NOT AT THE PROPER GOTO OR GOSUB LINE NUMBER
2185  AF58  35 04                    PULS  B                    GET BACK THE TOKEN FOLLOWING  GO
2186  AF5A  7E AE 88                 JMP   LAE88                GO DO A  GOTO  OR  GOSUB
2187  AF5D  9D 9F             LAF5D  JSR   GETNCH               GET A CHARACTER FROM BASIC
2188  AF5F  8D 06                    BSR   LAF67                CONVERT BASIC LINE NUMBER TO BINARY
2189  AF61  81 2C                    CMPA  #',                  IS CHARACTER FOLLOWING LINE NUMBER A COMMA?
2190  AF63  27 EF                    BEQ   LAF54                YES
2191  AF65  35 84                    PULS  B,PC                 IF NOT, FALL THROUGH TO NEXT COMMAND
2192  AF67  9E 8A             LAF67  LDX   ZERO                 DEFAULT LINE NUMBER OF ZERO
2193  AF69  9F 2B                    STX   BINVAL               SAVE IT IN BINVAL
2194                          *
2195                          * CONVERT LINE NUMBER TO BINARY - RETURN VALUE IN BINVAL
2196                          *
2197  AF6B  24 61             LAF6B  BCC   LAFCE                RETURN IF NOT NUMERIC CHARACTER
2198  AF6D  80 30                    SUBA  #'0                  MASK OFF ASCII
2199  AF6F  97 01                    STA   CHARAC               SAVE DIGIT IN V01
2200  AF71  DC 2B                    LDD   BINVAL               GET ACCUMULATED LINE NUMBER VALUE
2201  AF73  81 18                    CMPA  #24                  LARGEST LINE NUMBER IS $F9FF (63999) -
2202                          *                                  (24*256+255)*10+9
2203  AF75  22 DB                    BHI   LAF52                 SYNTAX  ERROR IF TOO BIG
2204                          * MULT ACCD X 10
2205  AF77  58                       ASLB                       *
2206  AF78  49                       ROLA                       * TIMES 2
2207  AF79  58                       ASLB                       =
2208  AF7A  49                       ROLA                       = TIMES 4
2209  AF7B  D3 2B                    ADDD  BINVAL               ADD 1 = TIMES 5
2210  AF7D  58                       ASLB                       *
2211  AF7E  49                       ROLA                       * TIMES 10
2212  AF7F  DB 01                    ADDB  CHARAC               ADD NEXT DIGIT
2213  AF81  89 00                    ADCA  #0                   PROPAGATE CARRY
2214  AF83  DD 2B                    STD   BINVAL               SAVE NEW ACCUMULATED LINE NUMBER
2215  AF85  9D 9F                    JSR   GETNCH               GET NEXT CHARACTER FROM BASIC
2216  AF87  20 E2                    BRA   LAF6B                LOOP- PROCESS NEXT DIGIT
2217                          *
2218                          * LET (EXBAS)
2219                          * EVALUATE A NON-TOKEN EXPRESSION
2220                          * TARGET = REPLACEMENT
2221  AF89  BD B3 57          LET    JSR   LB357                FIND TARGET VARIABLE DESCRIPTOR
2222  AF8C  9F 3B                    STX   VARDES               SAVE DESCRIPTOR ADDRESS OF 1ST EXPRESSION
2223  AF8E  C6 B3                    LDB   #$B3                 TOKEN FOR "="
2224  AF90  BD B2 6F                 JSR   LB26F                DO A SYNTAX CHECK FOR  =
2225  AF93  96 06                    LDA   VALTYP               * GET VARIABLE TYPE AND
```

```
2226    AF95  34 02                    PSHS   A                       * SAVE ON THE STACK
2227    AF97  BD B1 56                 JSR    LB156                   EVALUATE EXPRESSION
2228    AF9A  35 02                    PULS   A                       * REGET VARIABLE TYPE OF 1ST EXPRESSION AND
2229    AF9C  46                       RORA                           * SET CARRY IF STRING
2230    AF9D  BD B1 48                 JSR    LB148                   TYPE CHECK-TM ERROR IF VARIABLE TYPES ON
2231                                *                                  BOTH SIDES OF EQUALS SIGN NOT THE SAME
2232    AFA0  10 27 0C 8F              LBEQ   LBC33                   GO PUT FPA0 INTO VARIABLE DESCRIPTOR IF NUMERIC
2233                                * MOVE A STRING WHOSE DESCRIPTOR IS LOCATED AT
2234                                * FPA0+2 INTO THE STRING SPACE. TRANSFER THE
2235                                * DESCRIPTOR ADDRESS TO THE ADDRESS IN VARDES
2236                                * DON T MOVE THE STRING IF IT IS ALREADY IN THE
2237                                * STRING SPACE. REMOVE DESCRIPTOR FROM STRING
2238                                * STACK IF IT IS LAST ONE ON THE STACK
2239    AFA4  9E 52                 LAFA4  LDX    FPA0+2                  POINT X TO DESCRIPTOR OF REPLACEMENT STRING
2240    AFA6  DC 21                    LDD    FRETOP                  LOAD ACCD WITH START OF STRING SPACE
2241    AFA8  10 A3 02                 CMPD   2,X                     IS THE STRING IN STRING SPACE?
2242    AFAB  24 11                    BCC    LAFBE                   BRANCH IF IT S NOT IN THE STRING SPACE
2243    AFAD  9C 1B                    CMPX   VARTAB                  COMPARE DESCRIPTOR ADDRESS TO START OF VARIABLES
2244    AFAF  25 0D                    BCS    LAFBE                   BRANCH IF DESCRIPTOR ADDRESS NOT IN VARIABLES
2245    AFB1  E6 84                 LAFB1  LDB    ,X                      GET LENGTH OF REPLACEMENT STRING
2246    AFB3  BD B5 0D                 JSR    LB50D                   RESERVE ACCB BYTES OF STRING SPACE
2247    AFB6  9E 4D                    LDX    V4D                     GET DESCRIPTOR ADDRESS BACK
2248    AFB8  BD B6 43                 JSR    LB643                   MOVE STRING INTO STRING SPACE
2249    AFBB  8E 00 56                 LDX    #STRDES                 POINT X TO TEMP STRING DESCRIPTOR ADDRESS
2250    AFBE  9F 4D                 LAFBE  STX    V4D                     SAVE STRING DESCRIPTOR ADDRESS IN V4D
2251    AFC0  BD B6 75                 JSR    LB675                   REMOVE STRING DESCRIPTOR IF LAST ONE
2252                                *                                  ON STRING STACK
2253    AFC3  DE 4D                    LDU    V4D                     POINT U TO REPLACEMENT DESCRIPTOR ADDRESS
2254    AFC5  9E 3B                    LDX    VARDES                  GET TARGET DESCRIPTOR ADDRESS
2255    AFC7  37 26                    PULU   A,B,Y                   GET LENGTH AND START OF REPLACEMENT STRING
2256    AFC9  A7 84                    STA    ,X                      * SAVE STRING LENGTH AND START IN
2257    AFCB  10 AF 02                 STY    2,X                     * TARGET DESCRIPTOR LOCATION
2258    AFCE  39                    LAFCE  RTS
2259
2260    AFCF  3F 52 45 44 4F        LAFCF  FCC    '?REDO'                 ?REDO MESSAGE
2261    AFD4  0D 00                    FCB    CR,$00
2262
2263    AFD6  C6 22                 LAFD6  LDB    #2*17                    BAD FILE DATA  ERROR
2264    AFD8  0D 6F                    TST    DEVNUM                  CHECK DEVICE NUMBER AND BRANCH
2265    AFDA  27 03                    BEQ    LAFDF                   IF SET TO SCREEN
2266    AFDC  7E AC 46              LAFDC  JMP    LAC46                   JMP TO ERROR HANDLER
2267    AFDF  96 09                 LAFDF  LDA    INPFLG                  = GET THE INPUT FLAG AND BRANCH
2268    AFE1  27 07                    BEQ    LAFEA                   = IF  INPUT
2269    AFE3  9E 31                    LDX    DATTXT                  * GET LINE NUMBER WHERE THE ERROR OCCURRED
2270    AFE5  9F 68                    STX    CURLIN                  * AND USE IT AS THE CURRENT LINE NUMBER
2271    AFE7  7E B2 77                 JMP    LB277                    SYNTAX ERROR
2272    AFEA  8E AF CE              LAFEA  LDX    #LAFCF-1                * POINT X TO  ?REDO  AND PRINT
2273    AFED  BD B9 9C                 JSR    LB99C                   * IT ON THE SCREEN
2274    AFF0  9E 2F                    LDX    TINPTR                  = GET THE SAVED ABSOLUTE ADDRESS OF
2275    AFF2  9F A6                    STX    CHARAD                  = INPUT POINTER AND RESTORE IT
2276    AFF4  39                       RTS
2277                                *
2278                                * INPUT
2279    AFF5  C6 16                 INPUT  LDB    #11*2                    ID  ERROR
2280    AFF7  9E 68                    LDX    CURLIN                  GET CURRENT LINE NUMBER
2281    AFF9  30 01                    LEAX   1,X                     ADD ONE
2282    AFFB  27 DF                    BEQ    LAFDC                    ID  ERROR BRANCH IF DIRECT MODE
2283    AFFD  8D 03                    BSR    LB002                   GET SOME INPUT DATA
2284    AFFF  0F 6F                    CLR    DEVNUM                  SET DEVICE NUMBER TO SCREEN
2285    B001  39                       RTS
2286    B002  81 23                 LB002  CMPA   #'#                      CHECK FOR DEVICE NUMBER
2287    B004  26 09                    BNE    LB00F                   NO DEVICE NUMBER GIVEN
2288    B006  BD A5 A5                 JSR    LA5A5                   CHECK SYNTAX AND GET DEVICE NUMBER
2289    B009  BD A3 ED                 JSR    LA3ED                   CHECK FOR VALID INPUT FILE
2290    B00C  BD B2 6D                 JSR    LB26D                   SYNTAX CHECK FOR COMMA
2291    B00F  81 22                 LB00F  CMPA   #'"                      CHECK FOR PROMPT STRING DELIMITER
2292    B011  26 0B                    BNE    LB01E                   BRANCH IF NO PROMPT STRING
2293    B013  BD B2 44                 JSR    LB244                   PUT PROMPT STRING ON STRING STACK
2294    B016  C6 3B                    LDB    #';                     *
2295    B018  BD B2 6F                 JSR    LB26F                   * DO A SYNTAX CHECK FOR SEMICOLON
2296    B01B  BD B9 9F                 JSR    LB99F                   PRINT MESSAGE TO CONSOLE OUT
2297    B01E  8E 02 DC              LB01E  LDX    #LINBUF                 POINT TO BASIC S LINE BUFFER
2298    B021  6F 84                    CLR    ,X                      CLEAR 1ST BYTE - FLAG TO INDICATE NO DATA
2299                                *                                  IN LINE BUFFER
2300    B023  0D 6F                    TST    DEVNUM                  CHECK DEVICE NUMBER
2301    B025  26 22                    BNE    LB049                   BRANCH IF NOT SET TO SCREEN
2302    B027  8D 06                    BSR    LB02F                   INPUT A STRING TO LINE BUFFER
2303    B029  C6 2C                    LDB    #',                     * INSERT A COMMA AT THE END
2304    B02B  E7 84                    STB    ,X                      * OF THE LINE INPUT BUFFER
2305    B02D  20 1A                    BRA    LB049
2306                                * FILL BASIC S LINE INPUT BUFFER CONSOLE IN
2307    B02F  BD B9 AF              LB02F  JSR    LB9AF                   SEND A "?" TO CONSOLE OUT
2308    B032  BD B9 AC                 JSR    LB9AC                   SEND A  SPACE  TO CONSOLE OUT
2309    B035  BD A3 90              LB035  JSR    LA390                   GO READ IN A BASIC LINE
2310    B038  24 05                    BCC    LB03F                   BRANCH IF ENTER KEY ENDED ENTRY
2311    B03A  32 64                    LEAS   4,S                     PURGE TWO RETURN ADDRESSES OFF THE STACK
2312    B03C  7E AE 11                 JMP    LAE11                   GO DO A  STOP  IF BREAK KEY ENDED LINE ENTRY
2313    B03F  C6 2E                 LB03F  LDB    #2*23                    INPUT PAST END OF FILE  ERROR
2314    B041  0D 70                    TST    CINBFL                  CHECK FOR MORE CHARACTERS IN CONSOLE IN BUFFER
```

```
2315    B043  26 97                    BNE   LAFDC                    IE  ERROR IF EMPTY
2316    B045  39                       RTS
2317                            *
2318                            * READ
2319    B046  9E 33             READ    LDX   DATPTR                   GET  READ  START ADDRESS
2320    B048  86                        FCB   SKP1LD                   SKIP ONE BYTE - LDA #*$4F
2321    B0049 4F                LB049   CLRA                           INPUT  ENTRY POINT: INPUT FLAG = Ø
2322    B04A  97 Ø9                     STA   INPFLG                   SET INPUT FLAG; Ø = INPUT: <> Ø = READ
2323    B04C  9F 35                     STX   DATTMP                   SAVE  READ  START ADDRESS/ INPUT  BUFFER START
2324    B04E  BD B3 57          LB04E   JSR   LB357                    EVALUATE A VARIABLE
2325    B051  9F 3B                     STX   VARDES                   SAVE DESCRIPTOR ADDRESS
2326    B053  9E A6                     LDX   CHARAD                   * GET BASIC S INPUT POINTER
2327    B055  9F 2B                     STX   BINVAL                   * AND SAVE IT
2328    B057  9E 35                     LDX   DATTMP                   GET  READ  ADDRESS START/ INPUT  BUFFER POINTER
2329    B059  A6 84                     LDA   ,X                       GET A CHARACTER FROM THE BASIC PROGRAM
2330    B05B  26 ØC                     BNE   LBØ69                    BRANCH IF NOT END OF LINE
2331    B05D  96 Ø9                     LDA   INPFLG                   * CHECK INPUT FLAG AND BRANCH
2332    B05F  26 58                     BNE   LBØB9                    * IF LOOKING FOR DATA (READ)
2333                            * NO DATA IN  INPUT  LINE BUFFER AND/OR INPUT
2334                            * NOT COMING FROM SCREEN
2335    BØ61  BD Ø1 7C                  JSR   RVEC1Ø                   HOOK INTO RAM IF  INPUT
2336    BØ64  BD B9 AF                  JSR   LB9AF                    SEND A '?' TO CONSOLE OUT
2337    BØ67  8D C6                     BSR   LBØ2F                    FILL INPUT BUFFER FROM CONSOLE IN
2338    BØ69  9F A6             LBØ69   STX   CHARAD                   RESET BASIC S INPUT POINTER
2339    BØ6B  9D 9F                     JSR   GETNCH                   GET A CHARACTER FROM BASIC
2340    BØ6D  D6 Ø6                     LDB   VALTYP                   * CHECK VARIABLE TYPE AND
2341    BØ6F  27 27                     BEQ   LBØ98                    * BRANCH IF NUMERIC
2342                            * READ/INPUT A STRING VARIABLE
2343    BØ71  9E A6                     LDX   CHARAD                   LOAD X WITH CURRENT BASIC INPUT POINTER
2344    BØ73  97 Ø1                     STA   CHARAC                   SAVE CURRENT INPUT CHARACTER
2345    BØ75  81 22                     CMPA  #'"                      CHECK FOR STRING DELIMITER
2346    BØ77  27 12                     BEQ   LBØ8B                    BRANCH IF STRING DELIMITER
2347    BØ79  3Ø 1F                     LEAX  -1,X                     BACK UP POINTER
2348    BØ7B  4F                        CLRA                           * ZERO = END OF LINE CHARACTER
2349    BØ7C  97 Ø1                     STA   CHARAC                   * SAVE AS TERMINATOR
2350    BØ7E  BD A3 5F                  JSR   LA35F                    SET UP PRINT PARAMETERS
2351    BØ81  ØD 6E                     TST   PRTDEV                   CHECK PRINT DEVICE NUMBER
2352    BØ83  26 Ø6                     BNE   LBØ8B                    BRANCH IF CASSETTE - USE TWO ZEROS AS TERMINATOR
2353                            *                                      CHARACTERS FOR CASSETTE
2354    BØ85  86 3A                     LDA   #':                      END OF SUBLINE CHARACTER
2355    BØ87  97 Ø1                     STA   CHARAC                   SAVE AS TERMINATOR I
2356    BØ89  86 2C                     LDA   #',                      COMMA
2357    BØ8B  97 Ø2             LBØ8B   STA   ENDCHR                   SAVE AS TERMINATOR 2
2358    BØ8D  BD B5 1E                  JSR   LB51E                    STRIP A STRING FROM THE INPUT BUFFER
2359    BØ9Ø  BD B2 49                  JSR   LB249                    MOVE INPUT POINTER TO END OF STRING
2360    BØ93  BD AF A4                  JSR   LAFA4                    PUT A STRING INTO THE STRING SPACE IF NECESSARY
2361    BØ96  2Ø Ø6                     BRA   LBØ9E                    CHECK FOR ANOTHER DATA ITEM
2362                            * SAVE A NUMERIC VALUE IN A READ OR INPUT DATA ITEM
2363    BØ98  BD BD 12          LBØ98   JSR   LBD12                    CONVERT AN ASCII STRING TO FP NUMBER
2364    BØ9B  BD BC 33                  JSR   LBC33                    PACK FPAØ AND STORE IT IN ADDRESS IN VARDES -
2365                            *                                      INPUT OR READ DATA ITEM
2366    BØ9E  9D A5             LBØ9E   JSR   GETCCH                   GET CURRENT INPUT CHARACTER
2367    BØAØ  27 Ø6                     BEQ   LBØA8                    BRANCH IF END OF LINE
2368    BØA2  81 2C                     CMPA  #',                      CHECK FOR A COMMA
2369    BØA4  1Ø 26 FF 2E               LBNE  LAFD6                    'BAD FILE DATA' ERROR OR RETRY
2370    BØA8  9E A6             LBØA8   LDX   CHARAD                   * GET CURRENT INPUT
2371    BØAA  9F 35                     STX   DATTMP                   * POINTER (USED AS A DATA POINTER) AND SAVE IT
2372    BØAC  9E 2B                     LDX   BINVAL                   * RESET INPUT POINTER TO INPUT OR
2373    BØAE  9F A6                     STX   CHARAD                   * READ STATEMENT
2374    BØBØ  9D A5                     JSR   GETCCH                   GET CURRENT CHARACTER FROM BASIC
2375    BØB2  27 21                     BEQ   LBØD5                    BRANCH IF END OF LINE - EXIT COMMAND
2376    BØB4  BD B2 6D                  JSR   LB26D                    SYNTAX CHECK FOR COMMA
2377    BØB7  2Ø 95                     BRA   LBØ4E                    GET ANOTHER INPUT OR READ ITEM
2378                            * SEARCH FROM ADDRESS IN X FOR
2379                            * 1ST OCCURENCE OF THE TOKEN FOR DATA
2380    BØB9  9F A6             LBØB9   STX   CHARAD                   RESET BASIC S INPUT POINTER
2381    BØBB  BD AE E8                  JSR   LAEE8                    SEARCH FOR END OF CURRENT LINE OR SUBLINE
2382    BØBE  3Ø Ø1                     LEAX  1,X                      MOVE X ONE PAST END OF LINE
2383    BØCØ  4D                        TSTA                           CHECK FOR END OF LINE
2384    BØC1  26 ØA                     BNE   LBØCD                    BRANCH IF END OF SUBLINE
2385    BØC3  C6 Ø6                     LDB   #2*3                      OUT OF DATA  ERROR
2386    BØC5  EE 81                     LDU   ,X++                     GET NEXT 2 CHARACTERS
2387    BØC7  27 41                     BEQ   LB1ØA                     OD  ERROR IF END OF PROGRAM
2388    BØC9  EC 81                     LDD   ,X++                     GET BASIC LINE NUMBER AND
2389    BØCB  DD 31                     STD   DATTXT                   SAVE IT IN DATTXT
2390    BØCD  A6 84             LBØCD   LDA   ,X                       GET AN INPUT CHARACTER
2391    BØCF  81 86                     CMPA  #$86                     DATA TOKEN?
2392    BØD1  26 E6                     BNE   LBØB9                    NO   KEEP LOOKING
2393    BØD3  2Ø 94                     BRA   LBØ69                    YES
2394                            * EXIT READ AND INPUT COMMANDS
2395    BØD5  9E 35             LBØD5   LDX   DATTMP                   GET DATA POINTER
2396    BØD7  D6 Ø9                     LDB   INPFLG                   * CHECK INPUT FLAG
2397    BØD9  1Ø 26 FD ØB               LBNE  LADE8                    * SAVE NEW DATA POINTER IF READ
2398    BØDD  A6 84                     LDA   ,X                       = CHECK NEXT CHARACTER IN  INPUT  BUFFER
2399    BØDF  27 Ø6                     BEQ   LBØE7                    =  RETURN IF NO MORE DATA FOR INPUT
2400    BØE1  8E B0 E7                  LDX   #LBØE8-1                 POINT X TO  ?EXTRA IGNORED
2401    BØE4  7E B9 9C                  JMP   LB99C                    PRINT THE MESSAGE
2402    BØE7  39                LBØE7   RTS
2403
```

```
2404   B0E8  3F 45 58 54 52 41 LB0E8     FCC    '?EXTRA IGNORED'          ?EXTRA IGNORED MESSAGE
2405   B0EE  20 49 47 4E 4F 52
2406   B0F4  45 44
2407   B0F6  0D 00                       FCB    CR,$00
2408
2409                         * NEXT
2410   B0F8  26 04              NEXT     BNE    LB0FE                     BRANCH IF ARGUMENT GIVEN
2411   B0FA  9E 8A                       LDX    ZERO                      X = 0: DEFAULT FOR NO ARGUMENT
2412   B0FC  20 03                       BRA    LB101
2413   B0FE  BD B3 57           LB0FE    JSR    LB357                     EVALUATE AN ALPHA EXPRESSION
2414   B101  9F 3B             LB101    STX    VARDES                    SAVE VARIABLE DESCRIPTOR POINTER
2415   B103  BD AB F9                    JSR    LABF9                     GO SCAN FOR  FOR/NEXT  DATA ON STACK
2416   B106  27 04                       BEQ    LB10C                     BRANCH IF DATA FOUND
2417   B108  C6 00                       LDB    #0                         NEXT WITHOUT FOR  ERROR (SHOULD BE CLRB)
2418   B10A  20 47              LB10A    BRA    LB153                     PROCESS ERROR
2419   B10C  1F 14              LB10C    TFR    X,S                       POINT S TO START OF  FOR/NEXT  DATA
2420   B10E  30 03                       LEAX   3,X                       POINT X TO FP VALUE OF STEP
2421   B110  BD BC 14                    JSR    LBC14                     COPY A FP NUMBER FROM (X) TO FPA0
2422   B113  A6 68                       LDA    8,S                       GET THE DIRECTION OF STEP
2423   B115  97 54                       STA    FP0SGN                    SAVE IT AS THE SIGN OF FPA0
2424   B117  9E 3B                       LDX    VARDES                    POINT (X) TO INDEX VARIABLE DESCRIPTOR
2425   B119  BD B9 C2                    JSR    LB9C2                     ADD (X) TO FPA0 (STEP TO INDEX)
2426   B11C  BD BC 33                    JSR    LBC33                     PACK FPA0 AND STORE IT IN ADDRESS
2427                        *                                             CONTAINED IN VARDES
2428   B11F  30 69                       LEAX   9,S                       POINT (X) TO TERMINAL VALUE OF INDEX
2429   B121  BD BC 96                    JSR    LBC96                     COMPARE CURRENT INDEX VALUE TO TERMINAL VALUE OF INDEX
2430   B124  E0 68                       SUBB   8,S                       ACCB = 0 IF TERMINAL VALUE=CURRENT VALUE AND STEP=0 OR IF
2431                        *                                             STEP IS POSITIVE AND CURRENT VALUE>TERMINAL VALUE OR
2432                        *                                             STEP IS NEGATIVE AND CURRENT VALUE<TERMINAL VALUE
2433   B126  27 0C                       BEQ    LB134                     BRANCH IF  FOR/NEXT  LOOP DONE
2434   B128  AE 6E                       LDX    14,S                      * GET LINE NUMBER AND
2435   B12A  9F 68                       STX    CURLIN                    * BASIC POINTER OF
2436   B12C  AE E8 10                    LDX    16,S                      * STATEMENT FOLLOWING THE
2437   B12F  9F A6                       STX    CHARAD                    * PROPER FOR STATEMENT
2438   B131  7E AD 9E           LB131    JMP    LAD9E                     JUMP BACK TO COMMAND INTEPR. LOOP
2439   B134  32 E8 12           LB134    LEAS   18,S                      PULL THE  FOR-NEXT  DATA OFF THE STACK
2440   B137  9D A5                       JSR    GETCCH                    GET CURRENT INPUT CHARACTER
2441   B139  81 2C                       CMPA   #',                       CHECK FOR ANOTHER ARGUMENT
2442   B13B  26 F4                       BNE    LB131                     RETURN IF NONE
2443   B13D  9D 9F                       JSR    GETNCH                    GET NEXT CHARACTER FROM BASIC
2444   B13F  8D BD                       BSR    LB0FE                     BSR SIMULATES A CALL TO  NEXT  FROM COMMAND LOOP
2445
2446                         * EVALUATE A NUMERIC EXPRESSION
2447   B141  8D 13              LB141    BSR    LB156                     EVALUATE EXPRESSION AND DO A TYPE CHECK FOR NUMERIC
2448   B143  1C FE              LB143    ANDCC  #$FE                      CLEAR CARRY FLAG
2449   B145  7D                 LB145    FCB    $7D                       OP CODE OF TST $1A01 - SKIP TWO BYTES (DO
2450                        *                                             NOT CHANGE CARRY FLAG)
2451   B146  1A 01              LB146    ORCC   #1                        SET CARRY
2452
2453                         * STRING TYPE MODE CHECK - IF ENTERED AT LB146 THEN VALTYP PLUS IS 'TM' ERROR
2454                         * NUMERIC TYPE MODE CHECK - IF ENTERED AT LB143 THEN VALTYP MINUS IS 'TM' ERROR
2455                         * IF ENTERED AT LB148, A TYPE CHECK IS DONE ON VALTYP
2456                         * IF ENTERED WITH CARRY SET, THEN 'TM' ERROR IF NUMERIC
2457                         * IF ENTERED WITH CARRY CLEAR, THEN 'TM' ERROR IF STRING.
2458   B148  0D 06              LB148    TST    VALTYP                    TEST TYPE FLAG; DO NOT CHANGE CARRY
2459   B14A  25 03                       BCS    LB14F                     BRANCH IF STRING
2460   B14C  2A 99                       BPL    LB0E7                     RETURN ON PLUS
2461   B14E  8C                          FCB    SKP2                      SKIP 2 BYTES -  TM  ERROR
2462   B14F  2B 96              LB14F    BMI    LB0E7                     RETURN ON MINUS
2463   B151  C6 18                       LDB    #12*2                      TYPE M1SMATCH  ERROR
2464   B153  7E AC 46           LB153    JMP    LAC46                     PROCESS ERROR
2465                         * EVALUATE EXPRESSION
2466   B156  8D 6E              LB156    BSR    LB1C6                     BACK UP INPUT POINTER
2467   B158  4F                          CLRA                             END OF OPERATION PRECEDENCE FLAG
2468   B159  8C                          FCB    SKP2                      SKIP TWO BYTES
2469   B15A  34 04              LB15A    PSHS   B                         SAVE FLAG (RELATIONAL OPERATOR FLAG)
2470   B15C  34 02                       PSHS   A                         SAVE FLAG (PRECEDENCE FLAG)
2471   B15E  C6 01                       LDB    #1                        *
2472   B160  BD AC 33                    JSR    LAC33                     * SEE IF ROOM IN FREE RAM FOR (B) WORDS
2473   B163  BD B2 23                    JSR    LB223                     GO EVALUATE AN EXPRESSION
2474   B166  0F 3F                       CLR    TRELFL                    RESET RELATIONAL OPERATOR FLAG
2475   B168  9D A5              LB168    JSR    GETCCH                    GET CURRENT INPUT CHARACTER
2476                         * CHECK FOR RELATIONAL OPERATORS
2477   B16A  80 B2              LB16A    SUBA   #$B2                      TOKEN FOR >
2478   B16C  25 13                       BCS    LB181                     BRANCH IF LESS THAN RELATIONAL OPERATORS
2479   B16E  81 03                       CMPA   #3                        *
2480   B170  24 0F                       BCC    LB181                     * BRANCH IF GREATER THAN RELATIONAL OPERATORS
2481   B172  81 01                       CMPA   #1                        SET CARRY IF  >
2482   B174  49                          ROLA                             CARRY TO BIT 0
2483   B175  98 3F                       EORA   TRELFL                    * CARRY SET IF
2484   B177  91 3F                       CMPA   TRELFL                    * TRELFL = ACCA
2485   B179  25 64                       BCS    LB1DF                     BRANCH IF SYNTAX ERROR : == << OR >>
2486   B17B  97 3F                       STA    TRELFL                    BIT 0: >, BIT 1 =, BIT 2: <  SAVE DESIRED RELATIONAL COMPARISON
2487   B17D  9D 9F                       JSR    GETNCH                    GET AN INPUT CHARACTER
2488   B17F  20 E9                       BRA    LB16A                     CHECK FOR ANOTHER RELATIONAL OPERATOR
2489                        *
2490   B181  D6 3F              LB181    LDB    TRELFL                    GET RELATIONAL OPERATOR FLAG
2491   B183  26 33                       BNE    LB1B8                     BRANCH IF RELATIONAL COMPARISON
2492   B185  10 24 00 6B                 LBCC   LB1F4                     BRANCH IF > RELATIONAL OPERATOR
```

```
2493    B189  8B Ø7                    ADDA  #7                    SEVEN ARITHMETIC/LOGICAL OPERATORS
2494    B18B  24 67                    BCC   LB1F4                 BRANCH IF NOT ARITHMETIC/LOGICAL OPERATOR
2495    B18D  99 Ø6                    ADCA  VALTYP                ADD CARRY, NUMERIC FLAG AND MODIFIED TOKEN NUMBER
2496    B18F  1Ø 27 Ø4 7C              LBEQ  LB6ØF                 BRANCH IF VALTYP = FF, AND ACCA =  +  TOKEN -
2497                                                               CONCATENATE TWO STRINGS
2498    B193  89 FF                    ADCA  #-1                   RESTORE ARITHMETIC/LOGICAL OPERATOR NUMBER
2499    B195  34 Ø2                    PSHS  A                     * STORE OPERATOR NUMBER ON STACK; MULTIPLY IT BY 2
2500    B197  48                       ASLA                        * THEN ADD THE STORED STACK DATA = MULTIPLY
2501    B198  AB EØ                    ADDA  ,S+                   * X 3; 3 BYTE/TABLE ENTRY
2502    B19A  8E AA 51                 LDX   #LAA51                JUMP TABLE FOR ARITHMETIC & LOGICAL OPERATORS
2503    B19D  3Ø 86                    LEAX  A,X                   POINT X TO PROPER TABLE
2504    B19F  35 Ø2            LB19F   PULS  A                     GET PRECEDENCE FLAG FROM STACK
2505    B1A1  A1 84                    CMPA  ,X                    COMPARE TO CURRENT OPERATOR
2506    B1A3  24 55                    BCC   LB1FA                 BRANCH IF STACK OPERATOR > CURRENT OPERATOR
2507    B1A5  8D 9C                    BSR   LB143                  TM  ERROR IF VARIABLE TYPE = STRING
2508
2509                           * OPERATION BEING PROCESSED IS OF HIGHER PRECEDENCE THAN THE PREVIOUS OPERATION.
2510    B1A7  34 Ø2            LB1A7   PSHS  A                     SAVE PRECEDENCE FLAG
2511    B1A9  8D 29                    BSR   LB1D4                 PUSH OPERATOR ROUTINE ADDRESS AND FPAØ ONTO STACK
2512    B1AB  9E 3D                    LDX   RELPTR                GET POINTER TO ARITHMETIC/LOGICAL TABLE ENTRY FOR
2513                           *                                   LAST CALCULATED OPERATION
2514    B1AD  35 Ø2                    PULS  A                     GET PRECEDENCE FLAG OF PREVIOUS OPERATION
2515    B1AF  26 1D                    BNE   LB1CE                 BRANCH IF NOT END OF OPERATION
2516    B1B1  4D                       TSTA                        CHECK TYPE OF PRECEDENCE FLAG
2517  > B1B2  1Ø 27 ØØ 6A              LBEQ  LB22Ø                 BRANCH IF END OF EXPRESSION OR SUB-EXPRESSION
2518    B1B6  2Ø 4B                    BRA   LB2Ø3                 EVALUATE AN OPERATION
2519                           * DO A RELATIONAL COMPARISON HERE
2520    B1B8  Ø8 Ø6            LB1B8   ASL   VALTYP                BIT 7 OF TYPE FLAG TO CARRY
2521    B1BA  59                       ROLB                        SHIFT RELATIONAL FLAG LEFT - VALTYP TO BIT Ø
2522    B1BB  8D Ø9                    BSR   LB1C6                 MOVE THE INPUT POINTER BACK ONE
2523    B1BD  8E B1 CB                 LDX   #LB1CB                POINT X TO RELATIONAL COMPARISON JUMP TABLE
2524    B1CØ  D7 3F                    STB   TRELFL                SAVE RELATIONAL COMPARISON DATA
2525    B1C2  ØF Ø6                    CLR   VALTYP                SET VARIABLE TYPE TO NUMERIC
2526    B1C4  2Ø D9                    BRA   LB19F                 PERFORM OPERATION OR SAVE ON STACK
2527
2528    B1C6  9E A6            LB1C6   LDX   CHARAD                * GET BASIC S INPUT POINTER AND
2529    B1C8  7E AE BB                 JMP   LAEBB                 * MOVE IT BACK ONE
2530                           * RELATIONAL COMPARISON JUMP TABLE
2531    B1CB  64               LB1CB   FCB   $64                   RELATIONAL COMPARISON FLAG
2532    B1CC  B2 F4            LB1CC   FDB   LB2F4                 JUMP ADDRESS
2533
2534    B1CE  A1 84            LB1CE   CMPA  ,X                    COMPARE PRECEDENCE OF LAST DONE OPERATION TO
2535                           *                                   NEXT TO BE DONE OPERATION
2536    B1DØ  24 31                    BCC   LB2Ø3                 EVALUATE OPERATION IF LOWER PRECEDENCE
2537    B1D2  2Ø D3                    BRA   LB1A7                 PUSH OPERATION DATA ON STACK IF HIGHER PRECEDENCE
2538
2539                           * PUSH OPERATOR EVALUATION ADDRESS AND FPAØ ONTO STACK AND EVALUATE ANOTHER EXPR
2540    B1D4  EC Ø1            LB1D4   LDD   1,X                   GET ADDRESS OF OPERATOR ROUTINE
2541    B1D6  34 Ø6                    PSHS  B,A                   SAVE IT ON THE STACK
2542    B1D8  8D Ø8                    BSR   LB1E2                 PUSH FPAØ ONTO STACK
2543    B1DA  D6 3F                    LDB   TRELFL                GET BACK RELATIONAL OPERATOR FLAG
2544    B1DC  16 FF 7B                 LBRA  LB15A                 EVALUATE ANOTHER EXPRESSION
2545    B1DF  7E B2 77         LB1DF   JMP   LB277                  SYNTAX ERROR
2546                           * PUSH FPAØ ONTO THE STACK.  ,S = EXPONENT
2547                           * 1-2,S =HIGH ORDER MANTISSA  3-4,S = LOW ORDER MANTISSA
2548                           * 5,S = SIGN  RETURN WITH PRECEDENCE CODE IN ACCA
2549    B1E2  D6 54            LB1E2   LDB   FPØSGN                GET SIGN OF FPAØ MANTISSA
2550    B1E4  A6 84                    LDA   ,X                    GET PRECEDENCE CODE TO ACCA
2551    B1E6  35 2Ø            LB1E6   PULS  Y                     GET RETURN ADDRESS FROM STACK & PUT IT IN Y
2552    B1E8  34 Ø4                    PSHS  B                     SAVE ACCB ON STACK
2553    B1EA  D6 4F            LB1EA   LDB   FPØEXP                * PUSH FPAØ ONTO THE STACK
2554    B1EC  9E 5Ø                    LDX   FPAØ                  *
2555    B1EE  DE 52                    LDU   FPAØ+2                *
2556    B1FØ  34 54                    PSHS  U,X,B                 *
2557    B1F2  6E A4                    JMP   ,Y                    JUMP TO ADDRESS IN Y
2558
2559                           * BRANCH HERE IF NON-OPERATOR CHARACTER FOUND - USUALLY  )  OR END OF LINE
2560    B1F4  9E 8A            LB1F4   LDX   ZERO                  POINT X TO DUMMY VALUE (ZERO)
2561    B1F6  A6 EØ                    LDA   ,S+                   GET PRECEDENCE FLAG FROM STACK
2562    B1F8  27 26                    BEQ   LB22Ø                 BRANCH IF END OF EXPRESSION
2563    B1FA  81 64            LB1FA   CMPA  #$64                  * CHECK FOR RELATIONAL COMPARISON FLAG
2564    B1FC  27 Ø3                    BEQ   LB2Ø1                 * AND BRANCH IF RELATIONAL COMPARISON
2565    B1FE  BD B1 43                 JSR   LB143                  TM  ERROR IF VARIABLE TYPE = STRING
2566    B2Ø1  9F 3D            LB2Ø1   STX   RELPTR                SAVE POINTER TO OPERATOR ROUTINE
2567    B2Ø3  35 Ø4            LB2Ø3   PULS  B                     GET RELATIONAL OPERATOR FLAG FROM STACK
2568    B2Ø5  81 5A                    CMPA  #$5A                  CHECK FOR  NOT  OPERATOR
2569    B2Ø7  27 19                    BEQ   LB222                 RETURN IF  NOT  - NO RELATIONAL COMPARISON
2570    B2Ø9  81 7D                    CMPA  #$7D                  CHECK FOR NEGATION (UNARY) FLAG
2571    B2ØB  27 15                    BEQ   LB222                 RETURN IF NEGATION - NO RELATIONAL COMPARISON
2572
2573                           * EVALUATE AN OPERATION. EIGHT BYTES WILL BE STORED ON STACK, FIRST SIX BYTES
2574                           * ARE A TEMPORARY FLOATING POINT RESULT THEN THE ADDRESS OF ROUTINE WHICH
2575                           * WILL EVALUATE THE OPERATION. THE RTS AT END OF ROUTINE WILL VECTOR
2576                           * TO EVALUATING ROUTINE.
2577    B2ØD  54                       LSRB                        = ROTATE VALTYP BIT INTO CARRY
2578    B2ØE  D7 ØA                    STB   RELFLG                = FLAG AND SAVE NEW RELFLG
2579    B21Ø  35 52                    PULS  A,X,U                 * PULL A FP VALUE OFF OF THE STACK
2580    B212  97 5C                    STA   FP1EXP                * AND SAVE IT IN FPA1
2581    B214  9F 5D                    STX   FPA1                  *
```

```
2582   B216  DF 5F                    STU   FPA1+2                  *
2583   B218  35 Ø4                    PULS  B                       = GET MANTISSA SIGN AND
2584   B21A  D7 61                    STB   FP1SGN                  = SAVE IT IN FPA1
2585   B21C  D8 54                    EORB  FPØSGN                  EOR IT WITH FPA1 MANTISSA SIGN
2586   B21E  D7 62                    STB   RESSGN                  SAVE IT IN RESULT SIGN BYTE
2587   B22Ø  D6 4F             LB22Ø  LDB   FPØEXP                  GET EXPONENT OF FPAØ
2588   B222  39                LB222  RTS
2589
2590   B223  BD Ø1 8B          LB223  JSR   RVEC15                  HOOK INTO RAM
2591   B226  ØF Ø6                    CLR   VALTYP                  INITIALIZE TYPE FLAG TO NUMERIC
2592   B228  9D 9F                    JSR   GETNCH                  GET AN INPUT CHAR
2593   B22A  24 Ø3                    BCC   LB22F                   BRANCH IF NOT NUMERIC
2594   B22C  7E BD 12          LB22C  JMP   LBD12                   CONVERT ASCII STRING TO FLOATING POINT -
2595                                  *                             RETURN RESULT IN FPAØ
2596                                  * PROCESS A NON NUMERIC FIRST CHARACTER
2597   B22F  BD B3 A2          LB22F  JSR   LB3A2                   SET CARRY IF NOT ALPHA
2598   B232  24 5Ø                    BCC   LB284                   BRANCH IF ALPHA CHARACTER
2599   B234  81 2E                    CMPA  #'.                     IS IT  .  (DECIMAL POINT)?
2600   B236  27 F4                    BEQ   LB22C                   CONVERT ASCII STRING TO FLOATING POINT
2601   B238  81 AC                    CMPA  #$AC                    MINUS TOKEN
2602   B23A  27 4Ø                    BEQ   LB27C                   YES - GO PROCESS THE MINUS OPERATOR
2603   B23C  81 AB                    CMPA  #$AB                    PLUS TOKEN
2604   B23E  27 E3                    BEQ   LB223                   YES - GET ANOTHER CHARACTER
2605   B24Ø  81 22                    CMPA  #'"                     STRING DELIMITER?
2606   B242  26 ØA                    BNE   LB24E                   NO
2607   B244  9E A6             LB244  LDX   CHARAD                  CURRENT BASIC POINTER TO X
2608   B246  BD B5 18                 JSR   LB518                   SAVE STRING ON STRING STACK
2609   B249  9E 64             LB249  LDX   COEFPT                  * GET ADDRESS OF END OF STRING AND
2610   B24B  9F A6                    STX   CHARAD                  * PUT BASIC S INPUT POINTER THERE
2611   B24D  39                       RTS
2612   B24E  81 A8             LB24E  CMPA  #$A8                    NOT TOKEN?
2613   B25Ø  26 ØD                    BNE   LB25F                   NO
2614                                  * PROCESS THE NOT OPERATOR
2615   B252  86 5A                    LDA   #$5A                     NOT  PRECEDENCE FLAG
2616   B254  BD B1 5A                 JSR   LB15A                   PROCESS OPERATION FOLLOWING  NOT
2617   B257  BD B3 ED                 JSR   INTCNV                  CONVERT FPAØ TO INTEGER IN ACCD
2618   B25A  43                       COMA                          *  NOT  THE INTEGER
2619   B25B  53                       COMB                          *
2620   B25C  7E B4 F4                 JMP   GIVABF                  CONVERT ACCD TO FLOATING POINT (FPAØ)
2621   B25F  4C                LB25F  INCA                          CHECK FOR TOKENS PRECEEDED BY 5FF
2622   B26Ø  27 2E                    BEQ   LB29Ø                   IT WAS PRECEEDED BY 5FF
2623   B262  8D Ø6             LB262  BSR   LB26A                   SYNTAX CHECK FOR A  (
2624   B264  BD B1 56                 JSR   LB156                   EVALUATE EXPRESSIONS WITHIN PARENTHESES AT
2625                                  *                             HIGHEST PRECEDENCE
2626   B267  C6 29             LB267  LDB   #')                     SYNTAX CHECK FOR  )
2627   B269  8C                       FCB   SKP2                    SKIP 2 BYTES
2628   B26A  C6 28             LB26A  LDB   #'(                     SYNTAX CHECK FOR  (
2629   B26C  8C                       FCB   SKP2                    SKIP 2 BYTES
2630   B26D  C6 2C             LB26D  LDB   #',                     SYNTAX CHECK FOR COMMA
2631   B26F  E1 9F ØØ A6       LB26F  CMPB  [CHARAD]                * COMPARE ACCB TO CURRENT INPUT
2632   B273  26 Ø2                    BNE   LB277                   * CHARACTER - SYNTAX ERROR IF NO MATCH
2633   B275  ØE 9F                    JMP   GETNCH                  GET A CHARACTER FROM BASIC
2634   B277  C6 Ø2             LB277  LDB   #2*1                    SYNTAX ERROR
2635   B279  7E AC 46                 JMP   LAC46                   JUMP TO ERROR HANDLER
2636
2637                                  * PROCESS THE MINUS (UNARY) OPERATOR
2638   B27C  86 7D             LB27C  LDA   #$7D                    MINUS (UNARY) PRECEDENCE FLAG
2639   B27E  BD B1 5A                 JSR   LB15A                   PROCESS OPERATION FOLLOWING  UNARY  NEGATION
2640   B281  7E BE E9                 JMP   LBEE9                   CHANGE SIGN OF FPAØ MANTISSA
2641
2642                                  * EVALUATE ALPHA EXPRESSION
2643   B284  BD B3 57          LB284  JSR   LB357                   FIND THE DESCRIPTOR ADDRESS OF A VARIABLE
2644   B287  9F 52                    STX   FPAØ+2                  SAVE DESCRIPTOR ADDRESS IN FPAØ
2645   B289  96 Ø6                    LDA   VALTYP                  TEST VARIABLE TYPE
2646   B28B  26 95                    BNE   LB222                   RETURN IF STRING
2647   B28D  7E BC 14                 JMP   LBC14                   COPY A FP NUMBER FROM (X) TO FPAØ
2648
2649                                  * EVALUATING A SECONDARY TOKEN
2650   B29Ø  9D 9F             LB29Ø  JSR   GETNCH                  GET AN INPUT CHARACTER (SECONDARY TOKEN)
2651   B292  1F 89                    TFR   A,B                     SAVE IT IN ACCB
2652   B294  58                       ASLB                          X2 & BET RID OF BIT 7
2653   B295  9D 9F                    JSR   GETNCH                  GET ANOTHER INPUT CHARACTER
2654   B297  C1 26                    CMPB  #2*19                   19 SECONDARY FUNCTIONS IN BASIC
2655   B299  23 Ø4                    BLS   LB29F                   BRANCH IF COLOR BASIC TOKEN
2656   B29B  6E 9F Ø1 32              JMP   [COMVEC+18]             JUMP TO EXBAS SECONDARY TOKEN HANDLER
2657   B29F  34 Ø4             LB29F  PSHS  B                       SAVE TOKEN OFFSET ON STACK
2658   B2A1  C1 1C                    CMPB  #2*14                   CHECK FOR NUMERIC ARGUMENT TOKEN
2659   B2A3  25 22                    BCS   LB2C7                   DO SECONDARIES $8D (JOYSTK) OR LESS
2660   B2A5  C1 24                    CMPB  #2*18                   *
2661   B2A7  24 2Ø                    BCC   LB2C9                   * DO SECONDARIES $92 (INKEY$) OR >
2662   B2A9  8D BF                    BSR   LB26A                   SYNTAX CHECK FOR A  (
2663   B2AB  A6 E4                    LDA   ,S                      GET TOKEN NUMBER
2664   B2AD  81 22                    CMPA  #2*17                   CHECK FOR  POINT  COMMAND
2665   B2AF  24 18                    BCC   LB2C9                   DO POINT COMMAND ($91)
2666                                  * DO SECONDARIES $8E, $8F, $9Ø (LEFT$, RIGHT$, MID$)
2667   B2B1  BD B1 56                 JSR   LB156                   EVALUATE FIRST STRING IN ARGUMENT
2668   B2B4  8D B7                    BSR   LB26D                   SYNTAX CHECK FOR A COMMA
2669   B2B6  BD B1 46                 JSR   LB146                    TM  ERROR IF NUMERIC VARiABLE
2670   B2B9  35 Ø2                    PULS  A                       GET TOKEN OFFSET FROM STACK
```

```
2671    B2BB   DE 52                    LDU    FPA0+2                  POINT U TO STRING DESCRIPTOR
2672    B2BD   34 42                    PSHS   U,A                     SAVE TOKEN OFFSET AND DESCRIPTOR ADDRESS
2673    B2BF   BD B7 0B                 JSR    LB70B                   EVALUATE FIRST NUMERIC ARGUMENT
2674    B2C2   35 02                    PULS   A                       GET TOKEN OFFSET FROM STACK
2675    B2C4   34 06                    PSHS   B,A                     SAVE TOKEN OFFSET AND NUMERIC ARGUMENT
2676    B2C6   8E                       FCB    $8E                     OP CODE OF LDX# - SKIP 2 BYTES
2677    B2C7   8D 99            LB2C7   BSR    LB262                   SYNTAX CHECK FOR A  (
2678    B2C9   35 04            LB2C9   PULS   B                       GET TOKEN OFFSET
2679    B2CB   BE 01 28                 LDX    COMVEC+8                GET SECONDARY FUNCTION JUMP TABLE ADDRESS
2680    B2CE   3A               LB2CE   ABX                            ADD IN COMMAND OFFSET
2681                           *
2682                           * HERE IS WHERE WE BRANCH TO A SECONDARY FUNCTION
2683    B2CF   AD 94                    JSR    [,X]                    GO DO AN SECONDARY FUNCTION
2684    B2D1   7E B1 43                 JMP    LB143                    TM  ERROR IF VARIABLE TYPE = STRING
2685
2686                           * LOGICAL OPERATOR  OR  JUMPS HERE
2687    B2D4   86               LB2D4   FCB    SKP1LD                  SKIP ONE BYTE -  OR  FLAG = $4F
2688
2689                           * LOGICAL OPERATOR  AND  JUMPS HERE
2690    B2D5   4F               LB2D5   CLRA                           AND FLAG = 0
2691    B2D6   97 03                    STA    TMPLOC                  AND/OR FLAG
2692    B2D8   BD B3 ED                 JSR    INTCNV                  CONVERT FPA0 INTO AN INTEGER IN ACCD
2693    B2DB   DD 01                    STD    CHARAC                  TEMP SAVE ACCD
2694    B2DD   BD BC 4A                 JSR    LBC4A                   MOVE FPA1 TO FPA0
2695    B2E0   BD B3 ED                 JSR    INTCNV                  CONVERT FPA0 INTO AN INTEGER IN ACCD
2696    B2E3   0D 03                    TST    TMPLOC                  CHECK AND/OR FLAG
2697    B2E5   26 06                    BNE    LB2ED                   BRANCH IF OR
2698    B2E7   94 01                    ANDA   CHARAC                  *  AND  ACCD WITH FPA0 INTEGER
2699    B2E9   D4 02                    ANDB   ENDCHR                  * STORED IN ENDCHR
2700    B2EB   20 04                    BRA    LB2F1                   CONVERT TO FP
2701    B2ED   9A 01            LB2ED   ORA    CHARAC                  *  OR  ACCD WITH FPA0 INTEGER
2702    B2EF   DA 02                    ORB    ENDCHR                  * STORED IN CHARAC
2703    B2F1   7E B4 F4         LB2F1   JMP    GIVABF                  CONVERT THE VALUE IN ACCD INTO A FP NUMBER
2704
2705                           * RELATIONAL COMPARISON PROCESS HANDLER
2706    B2F4   BD B1 48                 JSR    LB148                    TM  ERROR IF TYPE MISMATCH
2707    B2F7   26 10                    BNE    LB309                   BRANCH IF STRING VARIABLE
2708    B2F9   96 61                    LDA    FP1SGN                  *  PACK  THE MANTISSA
2709    B2FB   8A 7F                    ORA    #$7F                    * SIGN OF FPA1 INTO
2710    B2FD   94 5D                    ANDA   FPA1                    * BIT 7 OF THE
2711    B2FF   97 5D                    STA    FPA1                    * MANTISSA MS BYTE
2712    B301   8E 00 5C                 LDX    #FP1EXP                 POINT X TO FPA1
2713    B304   BD BC 96                 JSR    LBC96                   COMPARE FPA0 TO FPA1
2714    B307   20 36                    BRA    LB33F                   CHECK TRUTH OF RELATIONAL COMPARISON
2715
2716                           * RELATIONAL COMPARISON OF STRINGS
2717    B309   0F 06            LB309  CLR    VALTYP                  SET VARIABLE TYPE TO NUMERIC
2718    B30B   0A 3F                    DEC    TRELFL                  REMOVE STRING TYPE FLAG (BIT0=1 FOR STRINGS) FROM THE
2719                           *                                      DESIRED RELATIONAL COMPARISON DATA
2720    B30D   BD B6 57                 JSR    LB657                   GET LENGTH AND ADDRESS OF STRING WHOSE
2721                           *                                      DESCRIPTOR ADDRESS IS IN THE BOTTOM OF FPA0
2722    B310   D7 56                    STB    STRDES                  * SAVE LENGTH AND ADDRESS IN TEMPORARY
2723    B312   9F 58                    STX    STRDES+2                * DESCRIPTOR (STRING B)
2724    B314   9E 5F                    LDX    FPA1+2                  = RETURN LENGTH AND ADDRESS OF STRING
2725    B316   BD B6 59                 JSR    LB659                   = WHOSE DESCRIPTOR ADDRESS IS STORED IN FPA1+2
2726    B319   96 56                    LDA    STRDES                  LOAD ACCA WITH LENGTH OF STRING B
2727    B31B   34 04                    PSHS   B                       SAVE LENGTH A ON STACK
2728    B31D   A0 E0                    SUBA   ,S+                     SUBTRACT LENGTH A FROM LENGTH B
2729    B31F   27 07                    BEQ    LB328                   BRANCH IF STRINGS OF EQUAL LENGTH
2730    B321   86 01                    LDA    #1                      TRUE FLAG
2731    B323   24 03                    BCC    LB328                   TRUE IF LENGTH B > LENGTH A
2732    B325   D6 56                    LDB    STRDES                  LOAD ACCB WITH LENGTH B
2733    B327   40                       NEGA                           SET FLAG = FALSE (1FF)
2734    B328   97 54            LB328   STA    FP0SGN                  SAVE TRUE/FALSE FLAG
2735    B32A   DE 58                    LDU    STRDES+2                POINT U TO START OF STRING
2736    B32C   5C                       INCB                           COMPENSATE FOR THE DECB BELOW
2737                           * ENTER WITH ACCB CONTAINING LENGTH OF SHORTER STRING
2738    B32D   5A               LB32D   DECB                           DECREMENT SHORTER STRING LENGTH
2739    B32E   26 04                    BNE    LB334                   BRANCH IF ALL OF STRING NOT COMPARED
2740    B330   D6 54                    LDB    FP0SGN                  GET TRUE/FALSE FLAB
2741    B332   20 0B                    BRA    LB33F                   CHECK TRUTH OF RELATIONAL COMPARISON
2742    B334   A6 80            LB334   LDA    ,X+                     GET A BYTE FROM STRING A
2743    B336   A1 C0                    CMPA   ,U+                     COMPARE TO STRING B
2744    B338   27 F3                    BEQ    LB32D                   CHECK ANOTHER CHARACTER IF =
2745    B33A   C6 FF                    LDB    #$FF                    FALSE FLAG IF STRING A > B
2746    B33C   24 01                    BCC    LB33F                   BRANCH IF STRING A > STRING B
2747    B33E   50                       NEGB                           SET FLAG = TRUE
2748
2749                           * DETERMINE TRUTH OF COMPARISON - RETURN RESULT IN FPA0
2750    B33F   CB 01            LB33F   ADDB   #1                      CONVERT $FF,0,1 TO 0,1,2
2751    B341   59                       ROLB                           NOW IT S 1,2,4 FOR > = <
2752    B342   D4 0A                    ANDB   RELFLG                   AND  THE ACTUAL COMPARISON WITH THE DESIRED -
2753                                                                   COMPARISON
2754    B344   27 02                    BEQ    LB348                   BRANCH IF FALSE (NO MATCHING BITS)
2755    B346   C6 FF                    LDB    #$FF                    TRUE FLAG
2756    B348   7E BC 7C         LB348   JMP    LBC7C                   CONVERT ACCB INTO FP NUMBER IN FPA0
2757
2758                           * DIM
2759    B34B   BD B2 6D         LB34B   JSR    LB26D                   SYNTAX CHECK FOR COMMA
```

```
2760   B34E  C6 01           DIM      LDB   #1                    DIMENSION FLAG
2761   B350  8D 08                    BSR   LB35A                 SAVE ARRAY SPACE FOR THIS VARIABLE
2762   B352  9D A5                    JSR   GETCCH                GET CURRENT INPUT CHARACTER
2763   B354  26 F5                    BNE   LB34B                 KEEP DIMENSIONING IF NOT END OF LINE
2764   B356  39                       RTS
2765                          * EVALUATE A VARIABLE - RETURN X AND
2766                          * VARPTR POINTING TO VARIABLE DESCRIPTOR
2767                          * EACH VARIABLE REQUIRES 7 BYTES - THE FIRST TWO
2768                          * BYTES ARE THE VARIABLE NAME AND THE NEXT 5
2769                          * BYTES ARE THE DESCRIPTOR. IF BIT 7 OF THE
2770                          * FIRST BYTE OF VARIABLE NAME IS SET, THE
2771                          * VARIABLE IS A DEF FN VARIABLE. IF BIT 7 OF
2772                          * THE SECOND BYTE OF VARIABLE NAME IS SET, THE
2773                          * VARIABLE IS A STRING, OTHERWISE THE VARIABLE
2774                          * IS NUMERIC.
2775                          * IF THE VARIABLE IS NOT FOUND, A ZERO VARIABLE IS
2776                          * INSERTED INTO THE VARIABLE SPACE
2777   B357  5F               LB357    CLRB                        DIMENSION FLAG = 0; DO NOT SET UP AN ARRAY
2778   B358  9D A5                    JSR   GETCCH                GET CURRENT INPUT CHARACTER
2779   B35A  D7 05            LB35A    STB   DIMFLG               SAVE ARRAY FLAG
2780   B35C  97 37                     STA   VARNAM                SAVE INPUT CHARACTER
2781                          * ENTRY POINT FOR DEF FN VARIABLE SEARCH
2782   B35E  9D A5            LB35C    JSR   GETCCH                GET CURRENT INPUT CHARACTER
2783   B360  8D 40                     BSR   LB3A2                 SET CARRY IF NOT ALPHA
2784   B362  10 25 FF 11               LBCS  LB277                 SYNTAX ERROR IF NOT ALPHA
2785   B366  5F                        CLRB                        DEFAULT 2ND VARIABLE CHARACTER TO ZERO
2786   B367  D7 06                     STB   VALTYP                SET VARIABLE TYPE TO NUMERIC
2787   B369  9D 9F                     JSR   GETNCH                GET ANOTHER CHARACTER FROM BASIC
2788   B36B  25 04                     BCS   LB371                 BRANCH IF NUMERIC (2ND CHARACTER IN
2789                          *                                    VARIABLE MAY BE NUMERIC)
2790   B36D  8D 33                     BSR   LB3A2                 SET CARRY IF NOT ALPHA
2791   B36F  25 0A                     BCS   LB37B                 BRANCH IF NOT ALPHA
2792   B371  1F 89            LB371    TFR   A,B                   SAVE 2ND CHARACTER IN ACCB
2793                          * READ INPUT CHARACTERS UNTIL A NON ALPHA OR
2794                          * NON NUMERIC IS FOUND - IGNORE ALL CHARACTERS
2795                          * IN VARIABLE NAME AFTER THE 1ST TWO
2796   B373  9D 9F            LB373    JSR   GETNCH                GET AN INPUT CHARACTER
2797   B375  25 FC                     BCS   LB373                 BRANCH IF NUMERIC
2798   B377  8D 29                     BSR   LB3A2                 SET CARRY IF NOT ALPHA
2799   B379  24 F8                     BCC   LB373                 BRANCH IF ALPHA
2800   B37B  81 24            LB37B    CMPA  #'$                   CHECK FOR A STRING VARIABLE
2801   B37D  26 06                     BNE   LB385                 BRANCH IF IT IS NOT A STRING
2802   B37F  03 06                     COM   VALTYP                SET VARIABLE TYPE TO STRING
2803   B381  CB 80                     ADDB  #$80                  SET BIT 7 OF 2ND CHARACTER (STRING)
2804   B383  9D 9F                     JSR   GETNCH                GET AN INPUT CHARACTER
2805   B385  D7 38            LB385    STB   VARNAM+1              SAVE 2ND CHARACTER IN VARNAM+1
2806   B387  9A 08                     ORA   ARYDIS                OR IN THE ARRAY DISABLE FLAG - IF = $80,
2807                          *                                    DON T SEARCH FOR VARIABLES IN THE ARRAYS
2808   B389  80 28                     SUBA  #'(                   IS THIS AN ARRAY VARIABLE?
2809 > B38B  10 27 00 75               LBEQ  LB404                 BRANCH IF IT IS
2810   B38F  0F 08                     CLR   ARYDIS                RESET THE ARRAY DISABLE FLAG
2811   B391  9E 1B                     LDX   VARTAB                POINT X TO THE START OF VARIABLES
2812   B393  DC 37                     LDD   VARNAM                GET VARIABLE IN QUESTION
2813   B395  9C 1D            LB395    CMPX  ARYTAB                COMPARE X TO THE END OF VARIABLES
2814   B397  27 12                     BEQ   LB3AB                 BRANCH IF END OF VARIABLES
2815   B399  10 A3 81                  CMPD  ,X++                  * COMPARE VARIABLE IN QUESTION TO CURRENT
2816   B39C  27 3E                     BEQ   LB3DC                 * VARIABLE AND BRANCH IF MATCH
2817   B39E  30 05                     LEAX  5,X                   = MOVE POINTER TO NEXT VARIABLE AND
2818   B3A0  20 F3                     BRA   LB395                 = KEEP LOOKING
2819
2820                          * SET CARRY IF NOT UPPER CASE ALPHA
2821   B3A2  81 41            LB3A2    CMPA  #'A                   * CARRY SET IF <  A
2822   B3A4  25 04                     BCS   LB3AA                 *
2823   B3A6  80 5B                     SUBA  #'Z+1                 =
2824   B3A8  80 A5                     SUBA  #-('Z+1)              = CARRY CLEAR IF <= 'Z'
2825   B3AA  39               LB3AA    RTS
2826                          * PUT A NEW VARIABLE IN TABLE OF VARIABLES
2827   B3AB  8E 00 8A         LB3AB    LDX   #ZERO                 POINT X TO ZERO LOCATION
2828   B3AE  EE E4                     LDU   ,S                    GET CURRENT RETURN ADDRESS
2829   B3B0  11 83 B2 87               CMPU  #LB287                DID WE COME FROM  EVALUATE ALPHA EXPR ?
2830   B3B4  27 28                     BEQ   LB3DE                 YES - RETURN A ZERO VALUE
2831   B3B6  DC 1F                     LDD   ARYEND                * GET END OF ARRAYS ADDRESS AND
2832   B3B8  DD 43                     STD   V43                   * SAVE IT AT V43
2833   B3BA  C3 00 07                  ADDD  #7                    = ADD 7 TO END OF ARRAYS (EACH
2834   B3BD  DD 41                     STD   V41                   = VARIABLE = 7 BYTES) AND SAVE AT V41
2835   B3BF  9E 1D                     LDX   ARYTAB                * GET END OF VARIABLES AND SAVE AT V47
2836   B3C1  9F 47                     STX   V47                   *
2837   B3C3  BD AC 1E                  JSR   LAC1E                 MAKE A SEVEN BYTE SLOT FOR NEW VARIABLE AT
2838                          *                                    TOP OF VARIABLES
2839   B3C6  9E 41                     LDX   V41                   = GET NEW END OF ARRAYS AND SAVE IT
2840   B3C8  9F 1F                     STX   ARYEND                =
2841   B3CA  9E 45                     LDX   V45                   * GET NEW END OF VARIABLES AND SAVE IT
2842   B3CC  9F 1D                     STX   ARYTAB                *
2843   B3CE  9E 47                     LDX   V47                   GET OLD END OF VARIABLES
2844   B3D0  DC 37                     LDD   VARNAM                GET NEW VARIABLE NAME
2845   B3D2  ED 81                     STD   ,X++                  SAVE VARIABLE NAME
2846   B3D4  4F                        CLRA                        * ZERO OUT THE FP VALUE OF THE NUMERIC
2847   B3D5  5F                        CLRB                        * VARIABLE OR THE LENGTH AND ADDRESS
2848   B3D6  ED 84                     STD   ,X                    * OF A STRING VARIABLE
```

```
2849    B3D8  ED Ø2                  STD   2,X                    *
2850    B3DA  A7 Ø4                  STA   4,X                    *
2851    B3DC  9F 39           LB3DC  STX   VARPTR                 STORE ADDRESS OF VARIABLE VALUE
2852    B3DE  39              LB3DE  RTS
2853                          *
2854    B3DF  9Ø 8Ø ØØ ØØ ØØ  LB3DF  FCB   $9Ø,$8Ø,$ØØ,$ØØ,$ØØ     * FLOATING POINT -32768
2855                          *                                    SMALLEST SIGNED TWO BYTE INTEGER
2856                          *
2857    B3E4  9D 9F           LB3E4  JSR   GETNCH                  GET AN INPUT CHARACTER FROM BASIC
2858    B3E6  BD B1 41        LB3E6  JSR   LB141                   GO EVALUATE NUMERIC EXPRESSION
2859    B3E9  96 54           LB3E9  LDA   FPØSGN                  GET FPAØ MANTISSA SIGN
2860    B3EB  2B 5D                  BMI   LB44A                   FC  ERROR IF NEGATIVE NUMBER
2861
2862                          * CONVERT FPAØ TO A SIGNED TWO BYTE INTEGER; RETURN VALUE IN ACCD
2863    B3ED  BD B1 43        INTCNV JSR   LB143                   TM  ERROR IF STRING VARIABLE
2864    B3FØ  96 4F                  LDA   FPØEXP                  GET FPAØ EXPONENT
2865    B3F2  81 9Ø                  CMPA  #$9Ø                    * COMPARE TO 32768 - LARGEST INTEGER EXPONENT AND
2866    B3F4  25 Ø8                  BCS   LB3FE                   * BRANCH IF FPAØ < 32768
2867    B3F6  8E B3 DF               LDX   #LB3DF                  POINT X TO FP VALUE OF -32768
2868    B3F9  BD BC 96               JSR   LBC96                   COMPARE -32768 TO FPAØ
2869    B3FC  26 4C                  BNE   LB44A                   FC  ERROR IF NOT =
2870    B3FE  BD BC C8        LB3FE  JSR   LBCC8                   CONVERT FPAØ TO A TWO BYTE INTEGER
2871    B4Ø1  DC 52                  LDD   FPA Ø+2                 GET THE INTEGER
2872    B4Ø3  39                     RTS
2873                          * EVALUATE AN ARRAY VARIABLE
2874    B4Ø4  DC Ø5           LB4Ø4  LDD   DIMFLG                  GET ARRAY FLAG AND VARIABLE TYPE
2875    B4Ø6  34 Ø6                  PSHS  B,A                     SAVE THEM ON STACK
2876    B4Ø8  12                     NOP                           DEAD SPACE CAUSED BY 1.2 REVISION
2877    B4Ø9  5F                     CLRB                          RESET DIMENSION COUNTER
2878    B4ØA  9E 37           LB4ØA  LDX   VARNAM                  GET VARIABLE NAME
2879    B4ØC  34 14                  PSHS  X,B                     SAVE VARIABLE NAME AND DIMENSION COUNTER
2880    B4ØE  8D D4                  BSR   LB3E4                   EVALUATE EXPRESSION (DIMENSION LENGTH)
2881    B41Ø  35 34                  PULS  B,X,Y                   PULL OFF VARIABLE NAME, DIMENSION COUNTER,
2882                          *                                    ARRAY FLAG
2883    B412  9F 37                  STX   VARNAM                  SAVE VARIABLE NAME AND VARIABLE TYPE
2884    B414  DE 52                  LDU   FPA Ø+2                 GET DIMENSION LENGTH
2885    B416  34 6Ø                  PSHS  U,Y                     SAVE DIMENSION LENGTH, ARRAY FLAG, VARIABLE TYPE
2886    B418  5C                     INCB                          INCREASE DIMENSION COUNTER
2887    B419  9D A5                  JSR   GETCCH                  GET CURRENT INPUT CHARACTER
2888    B41B  81 2C                  CMPA  #',                     CHECK FOR ANOTHER DIMENSION
2889    B41D  27 EB                  BEQ   LB4ØA                   BRANCH IF MORE
2890    B41F  D7 Ø3                  STB   TMPLOC                  SAVE DIMENSION COUNTER
2891    B421  BD B2 67               JSR   LB267                   SYNTAX CHECK FOR A  )
2892    B424  35 Ø6                  PULS  A,B                     * RESTORE VARIABLE TYPE AND ARRAY
2893    B426  DD Ø5                  STD   DIMFLG                  * FLAG - LEAVE DIMENSION LENGTH ON STACK
2894    B428  9E 1D                  LDX   ARYTAB                  GET START OF ARRAYS
2895    B42A  9C 1F           LB42A  CMPX  ARYEND                  COMPARE TO END OF ARRAYS
2896    B42C  27 21                  BEQ   LB44F                   BRANCH IF NO MATCH FOUND
2897    B42E  DC 37                  LDD   VARNAM                  GET VARIABLE IN QUESTION
2898    B43Ø  1Ø A3 84               CMPD  ,X                      COMPARE TO CURRENT VARIABLE
2899    B433  27 Ø6                  BEQ   LB43B                   BRANCH IF =
2900    B435  EC Ø2                  LDD   2,X                     GET OFFSET TO NEXT ARRAY VARIABLE
2901    B437  3Ø 8B                  LEAX  D,X                     ADD TO CURRENT POINTER
2902    B439  2Ø EF                  BRA   LB42A                   KEEP SEARCHING
2903    B43B  C6 12           LB43B  LDB   #2*9                    REDIMENSIONED ARRAY  ERROR
2904    B43D  96 Ø5                  LDA   DIMFLG                  * TEST ARRAY FLAG - IF <>Ø YOU ARE TRYING
2905    B43F  26 ØB                  BNE   LB44C                   * TO REDIMENSION AN ARRAY
2906    B441  D6 Ø3                  LDB   TMPLOC                  GET NUMBER OF DIMENSIONS IN ARRAY
2907    B443  E1 Ø4                  CMPB  4,X                     COMPARE TO THIS ARRAYS DIMENSIONS
2908    B445  27 59                  BEQ   LB4AØ                   BRANCH IF =
2909    B447  C6 1Ø           LB447  LDB   #8*2                    BAD SUBSCRIPT
2910    B449  8C                     FCB   SKP2                    SKIP TWO BYTES
2911    B44A  C6 Ø8           LB44A  LDB   #4*2                    ILLEGAL FUNCTION CALL
2912    B44C  7E AC 46        LB44C  JMP   LAC46                   JUMP TO ERROR SERVICING ROUTINE
2913
2914                          * INSERT A NEW ARRAY INTO ARRAY VARIABLES
2915                          * EACH SET OF ARRAY VARIABLES IS PRECEEDED BY A DE-
2916                          * SCRIPTOR BLOCK COMPOSED OF 5+2*N BYTES WHERE N IS THE
2917                          * NUMBER OF DIMENSIONS IN THE ARRAY. THE BLOCK IS DEFINED
2918                          * AS FOLLOWS: BYTES Ø,1:VARIABLE S NAME; 2,3:TOTAL LENGTH
2919                          * OF ARRAY ITEMS AND DESCRIPTOR BLOCK; 4:NUMBER OF DIMEN-
2920                          * ISIONS; 5,6:LENGTH OF DIMENSION 1; 7,8:LENGTH OF DIMEN-
2921                          * SION 2;  4+N,5+N:LENGTH OF DIMENSION N.
2922
2923    B44F  CC ØØ Ø5        LB44F  LDD   #5                      * 5 BYTES/ARRAY ENTRY SAVE AT COEFPT
2924    B452  DD 64                  STD   COEFPT                  *
2925    B454  DC 37                  LDD   VARNAM                  = GET NAME OF ARRAY AND SAVE IN
2926    B456  ED 84                  STD   ,X                      = FIRST 2 BYTES OF DESCRIPTOR
2927    B458  D6 Ø3                  LDB   TMPLOC                  GET NUMBER OF DIMENSIONS AND SAVE IN
2928    B45A  E7 Ø4                  STB   4,X                     * 5TH BYTE OF DESCRIPTOR
2929    B45C  BD AC 33               JSR   LAC33                   CHECK FOR ROOM FOR DESCRIPTOR IN FREE RAM
2930    B45F  9F 41                  STX   V41                     TEMPORARILY SAVE DESCRIPTOR ADDRESS
2931    B461  C6 ØB           LB461  LDB   #11                     * DEFAULT DIMENSION VALUE:X(1Ø)
2932    B463  4F                     CLRA                          *
2933    B464  ØD Ø5                  TST   DIMFLG                  = CHECK ARRAY FLAG AND BRANCH IF
2934    B466  27 Ø5                  BEQ   LB46D                   = NOT DIMENSIONING AN ARRAY
2935    B468  35 Ø6                  PULS  A,B                     GET DIMENSION LENGTH
2936    B46A  C3 ØØ Ø1               ADDD  #1                      ADD ONE (X(Ø) HAS A LENGTH OF ONE)
2937    B46D  ED Ø5           LB46D  STD   5,X                     SAVE LENGTH OF ARRAY DIMENSION
```

```
2938   B46F  8D 5D                  BSR   LB4CE                    MULTIPLY ACCUM ARRAY SIZE NUMBER LENGTH
2939                               *                                OF NEW DIMENSION
2940   B471  DD 64                  STD   COEFPT                   TEMP STORE NEW CURRENT ACCUMULATED ARRAY SIZE
2941   B473  30 02                  LEAX  2,X                      BUMP POINTER UP TWO
2942   B475  0A 03                  DEC   TMPLOC                   * DECREMENT DIMENSION COUNTER AND BRANCH IF
2943   B477  26 E8                  BNE   LB461                    * NOT DONE WITH ALL DIMENSIONS
2944   B479  9F 0F                  STX   TEMPTR                   SAVE ADDRESS OF (END OF ARRAY DESCRIPTOR - 5)
2945   B47B  D3 0F                  ADDD  TEMPTR                   ADD TOTAL SIZE OF NEW ARRAY
2946   B47D  10 25 F7 C3            LBCS  LAC44                     OM  ERROR IF > $FFFF
2947   B481  1F 01                  TFR   D,X                      SAVE END OF ARRAY IN X
2948   B483  BD AC 37               JSR   LAC37                    MAKE SURE THERE IS ENOUGH FREE RAM FOR ARRAY
2949   B486  83 00 35               SUBD  #STKBUF-5                SUBTRACT OUT THE (STACK BUFFER - 5)
2950   B489  DD 1F                  STD   ARYEND                   SAVE NEW END OF ARRAYS
2951   B48B  4F                     CLRA                           ZERO = TERMINATOR BYTE
2952   B48C  30 1F                  LB48C LEAX  -1,X               * STORE TWO TERMINATOR BYTES AT
2953   B48E  A7 05                  STA   5,X                      * THE END OF THE ARRAY DESCRIPTOR
2954   B490  9C 0F                  CMPX  TEMPTR                   *
2955   B492  26 F8                  BNE   LB48C                    *
2956   B494  9E 41                  LDX   V41                      GET ADDRESS OF START OF DESCRIPTOR
2957   B496  96 1F                  LDA   ARYEND                   GET MSB OF END OF ARRAYS; LSB ALREADY THERE
2958   B498  93 41                  SUBD  V41                      SUBTRACT OUT ADDRESS OF START OF DESCRIPTOR
2959   B49A  ED 02                  STD   2,X                      SAVE LENGTH OF (ARRAY AND DESCRIPTOR)
2960   B49C  96 05                  LDA   DIMFLG                   * GET ARRAY FLAG AND BRANCH
2961   B49E  26 2D                  BNE   LB4CD                    * BACK IF DIMENSIONING
2962                               * CALCULATE POINTER TO CORRECT ELEMENT
2963   B4A0  E6 04                  LB4A0 LDB   4,X                GET THE NUMBER OF DIMENSIONS
2964   B4A2  D7 03                  STB   TMPLOC                   TEMPORARILY SAVE
2965   B4A4  4F                     CLRA                           * INITIALIZE POINTER
2966   B4A5  5F                     CLRB                           * TO ZERO
2967   B4A6  DD 64                  LB4A6 STD   COEFPT             SAVE ACCUMULATED POINTER
2968   B4A8  35 06                  PULS  A,B                      * PULL DIMENSION ARGUMENT OFF THE
2969   B4AA  DD 52                  STD   FPA0+2                   * STACK AND SAVE IT
2970   B4AC  10 A3 05               CMPD  5,X                      COMPARE TO STORED  DIM  ARGUMENT
2971   B4AF  24 3A                  BCC   LB4EB                     BS  ERROR IF > = "DIM" ARGUMENT
2972   B4B1  DE 64                  LDU   COEFPT                   * GET ACCUMULATED POINTER AND
2973   B4B3  27 04                  BEQ   LB4B9                    * BRANCH IF 1ST DIMENSION
2974   B4B5  8D 17                  BSR   LB4CE                    = MULTIPLY ACCUMULATED POINTER AND DIMENSION
2975   B4B7  D3 52                  ADDD  FPA0+2                   = LENGTH AND ADD TO CURRENT ARGUMENT
2976   B4B9  30 02                  LB4B9 LEAX  2,X                MOVE POINTER TO NEXT DIMENSION
2977   B4BB  0A 03                  DEC   TMPLOC                   * DECREMENT DIMENSION COUNTER AND
2978   B4BD  26 E7                  BNE   LB4A6                    * BRANCH IF ANY DIMENSIONS LEFT
2979                               * MULTIPLY ACCD BY 5 - 5 BYTES/ARRAY VALUE
2980   B4BF  ED E3                  STD   ,--S
2981   B4C1  58                     ASLB
2982   B4C2  49                     ROLA                           TIMES 2
2983   B4C3  58                     ASLB
2984   B4C4  49                     ROLA                           TIMES 4
2985   B4C5  E3 E1                  ADDD  ,S++                     TIMES 5
2986   B4C7  30 8B                  LEAX  D,X                      ADD OFFSET TO START OF ARRAY
2987   B4C9  30 05                  LEAX  5,X                      ADJUST POINTER FOR SIZE OF DESCRIPTOR
2988   B4CB  9F 39                  STX   VARPTR                   SAVE POINTER TO ARRAY VALUE
2989   B4CD  39                     LB4CD RTS
2990                               * MULTIPLY 2 BYTE NUMBER IN 5,X BY THE 2 BYTE NUMBER
2991                               IN COEFPT. RETURN RESULT IN ACCD, BS ERROR IF > $FFFF
2992   B4CE  86 10                  LB4CE LDA   #16                16 SHIFTS TO DO A MULTIPLY
2993   B4D0  97 45                  STA   V45                      SHIFT COUNTER
2994   B4D2  EC 05                  LDD   5,X                      * GET SIZE OF DIMENSION
2995   B4D4  DD 17                  STD   BOTSTK                   * AND SAVE IT
2996   B4D6  4F                     CLRA                           * ZERO
2997   B4D7  5F                     CLRB                           * ACCD
2998   B4D8  58                     LB4D8 ASLB                     = SHIFT ACCB LEFT
2999   B4D9  49                     ROLA                           = ONE BIT
3000   B4DA  25 0F                  BCS   LB4EB                    'BS' ERROR IF CARRY
3001   B4DC  08 65                  ASL   COEFPT+1                 * SHIFT MULTIPLICAND LEFT ONE
3002   B4DE  09 64                  ROL   COEFPT                   * BIT - ADD MULTIPLIER TO ACCUMULATOR
3003   B4E0  24 04                  BCC   LB4E6                    * IF CARRY <> 0
3004   B4E2  D3 17                  ADDD  BOTSTK                   ADD MULTIPLIER TO ACCD
3005   B4E4  25 05                  BCS   LB4EB                    'BS' ERROR IF CARRY (>$FFFF)
3006   B4E6  0A 45                  LB4E6 DEC   V45                * DECREMENT SHIFT COUNTER
3007   B4E8  26 EE                  BNE   LB4D8                    * IF NOT DONE
3008   B4EA  39                     RTS
3009   B4EB  7E B4 47               LB4EB JMP   LB447              'BS' ERROR
3010                               *
3011                               * MEM
3012                               * THIS IS NOT A TRUE INDICATOR OF FREE MEMORY BECAUSE
3013                               * BASIC REQUIRES A STKBUF SIZE BUFFER FOR THE STACK
3014                               * FOR WHICH MEM DOES NOT ALLOW.
3015                               *
3016   B4EE  1F 40                  MEM   TFR   S,D                PUT STACK POINTER INTO ACCD
3017   B4F0  93 1F                  SUBD  ARYEND                   SUBTRACT END OF ARRAYS
3018   B4F2  21                     FCB   SKP1                     SKIP ONE BYTE
3019                               *CONVERT THE VALUE IN ACCB INTO A FP NUMBER IN FPA0
3020   B4F3  4F                     LB4F3 CLRA                     CLEAR MS BYTE OF ACCD
3021                               * CONVERT THE VALUE IN ACCD INTO A FLOATING POINT NUMBER IN FPA0
3022   B4F4  0F 06                  GIVABF CLR  VALTYP             SET VARIABLE TYPE TO NUMERIC
3023   B4F6  DD 50                  STD   FPA0                     SAVE ACCD IN TOP OF FACA
3024   B4F8  C6 90                  LDB   #$90                     EXPONENT REQUIRED IF THE TOP TWO BYTES
3025                               *                              OF FPA0 ARE TO BE TREATED AS AN INTEGER IN FPA0
3026   B4FA  7E BC 82               JMP   LBC82                    CONVERT THE REST OF FPA0 TO AN INTEGER
```

```
3027
3028                            * STR$
3029   B4FD  BD B1 43           STR       JSR   LB143                     'TM' ERROR IF STRING VARIABLE
3030   B500  CE 03 D9                     LDU   #STRBUF+2                 *CONVERT FP NUMBER TO ASCII STRING IN
3031   B503  BD BD DC                     JSR   LBDDC                     *THE STRING BUFFER
3032   B506  32 62                        LEAS  2,S                       PURGE THE RETURN ADDRESS FROM THE STACK
3033   B508  8E 03 D8                     LDX   #STRBUF+1                 *POINT X TO STRING BUFFER AND SAVE
3034   B50B  20 0B                        BRA   LB518                     *THE STRING IN THE STRING SPACE
3035                            * RESERVE ACCB BYTES OF STRING SPACE. RETURN START
3036                            * ADDRESS IN (X) AND FRESPC
3037   B50D  9F 4D              LB50D     STX   V4D                       SAVE X IN V4D
3038   B50F  8D 5C              LB50F     BSR   LB56D                     RESERVE ACCB BYTES IN STRING SPACE
3039   B511  9F 58              LB511     STX   STRDES+2                  SAVE NEW STRING ADDRESS
3040   B513  D7 56                        STB   STRDES                    SAVE LENGTH OF RESERVED BLOCK
3041   B515  39                           RTS
3042   B516  30 1F              LB516     LEAX  -1,X                      MOVE POINTER BACK ONE
3043                            * SCAN A LINE FROM (X) UNTIL AN END OF LINE FLAG (ZERO) OR
3044                            * EITHER OF THE TWO TERMINATORS STORED IN CHARAC OR ENDCHR IS MATCHED.
3045                            * THE RESULTING STRING IS STORED IN THE STRING SPACE
3046                            * ONLY IF THE START OF THE STRING IS <= STRBUF+2
3047   B518  86 22              LB518     LDA   #'"                       * INITIALIZE
3048   B51A  97 01                        STA   CHARAC                    * TERMINATORS
3049   B51C  97 02              LB51A     STA   ENDCHR                    * TO "
3050   B51E  30 01              LB51E     LEAX  1,X                       MOVE POINTER UP ONE
3051   B520  9F 62                        STX   RESSGN                    TEMPORARILY SAVE START OF STRING
3052   B522  9F 58                        STX   STRDES+2                  SAVE START OF STRING IN TEMP DESCRIPTOR
3053   B524  C6 FF                        LDB   #-1                       INITIALIZE CHARACTER COUNTER TO - 1
3054   B526  5C                 LB526     INCB                            INCREMENT CHARACTER COUNTER
3055   B527  A6 80                        LDA   ,X+                       GET CHARACTER
3056   B529  27 0C                        BEQ   LB537                     BRANCH IF END OF LINE
3057   B52B  91 01                        CMPA  CHARAC                    * CHECK FOR TERMINATORS
3058   B52D  27 04                        BEQ   LB533                     * IN CHARAC AND ENDCHR
3059   B52F  91 02                        CMPA  ENDCHR                    * DON T MOVE POINTER BACK
3060   B531  26 F3                        BNE   LB526                     * ONE IF TERMINATOR IS "MATCHED"
3061   B533  81 22              LB533     CMPA  #'"                       = COMPARE CHARACTER TO STRING DELIMITER
3062   B535  27 02                        BEQ   LB539                     = & DON T MOVE POINTER BACK IF SO
3063   B537  30 1F              LB537     LEAX  -1,X                      MOVE POINTER BACK ONE
3064   B539  9F 64              LB539     STX   COEFPT                    SAVE END OF STRING ADDRESS
3065   B53B  D7 56                        STB   STRDES                    SAVE STRING LENGTH IN TEMP DESCRIPTOR
3066   B53D  DE 62                        LDU   RESSGN                    GET INITIAL STRING START
3067   B53F  11 83 03 D9                  CMPU  #STRBUF+2                 COMPARE TO START OF STRING BUFFER
3068   B543  22 07              LB543     BHI   LB54C                     BRANCH IF > START OF STRING BUFFER
3069   B545  8D C6                        BSR   LB50D                     GO RESERVE SPACE FOR THE STRING
3070   B547  9E 62                        LDX   RESSGN                    POINT X TO THE BEGINNING OF THE STRING
3071   B549  BD B6 45                     JSR   LB645                     MOVE (B) BYTES FROM (X) TO
3072                            *                                         [FRESPC] - MOVE STRING DATA
3073                            * PUT DIRECT PAGE STRING DESCRIPTOR BUFFER DATA
3074                            * ON THE STRING STACK. SET VARIABLE TYPE TO STRING
3075   B54C  9E 0B              LB54C     LDX   TEMPPT                    GET NEXT AVAILABLE STRING STACK DESCRIPTOR
3076   B54E  8C 01 D1                     CMPX  #CFNBUF                   COMPARE TO TOP OF STRING DESCRIPTOR STACK
3077   B551  26 05                        BNE   LB558                     FORMULA O.K.
3078   B553  C6 1E                        LDB   #15*2                     'STRING FORMULA TOO COMPLEX' ERROR
3079   B555  7E AC 46           LB555     JMP   LAC46                     JUMP TO ERROR SERVICING ROUTINE
3080   B558  96 56              LB558     LDA   STRDES                    * GET LENGTH OF STRING AND SAVE IT
3081   B55A  A7 00                        STA   ,X                        * IN BYTE 0 OF DESCRIPTOR
3082   B55C  DC 58                        LDD   STRDES+2                  = GET START ADDRESS OF ACTUAL STRING
3083   B55E  ED 02                        STD   2,X                       = AND SAVE IN BYTES 2,3 OF DESCRIPTOR
3084   B560  86 FF                        LDA   #$FF                      * VARIABLE TYPE = STRING
3085   B562  97 06                        STA   VALTYP                    * SAVE IN VARIABLE TYPE FLAG
3086   B564  9F 0D                        STX   LASTPT                    = SAVE START OF DESCRIPTOR
3087   B566  9F 52                        STX   FPA0+2                    = ADDRESS IN LASTPT AND FPA0
3088   B568  30 05                        LEAX  5,X                       5 BYTES/STRING DESCRIPTOR
3089   B56A  9F 0B                        STX   TEMPPT                    NEXT AVAILABLE STRING VARIABLE DESCRIPTOR
3090   B56C  39                           RTS
3091                            * RESERVE ACCB BYTES IN STRING STORAGE SPACE
3092                            * RETURN WITH THE STARTING ADDRESS OF THE
3093                            * RESERVED STRING SPACE IN (X) AND FRESPC
3094   B56D  0F 07              LB56D     CLR   GARBFL                    CLEAR STRING REORGANIZATION FLAG
3095   B56F  4F                 LB56F     CLRA                            * PUSH THE LENGTH OF THE
3096   B570  34 06                        PSHS  B,A                       * STRING ONTO THE STACK
3097   B572  DC 23                        LDD   STRTAB                    GET START OF STRING VARIABLES
3098   B574  A3 E0                        SUBD  ,S+                       SUBTRACT STRING LENGTH
3099   B576  10 93 21                     CMPD  FRETOP                    COMPARE TO START OF STRING STORAGE
3100   B579  25 0A                        BCS   LB585                     IF BELOW START, THEN REORGANIZE
3101   B57B  DD 23                        STD   STRTAB                    SAVE NEW START OF STRING VARIABLES
3102   B57D  9E 23                        LDX   STRTAB                    GET START OF STRING VARIABLES
3103   B57F  30 01                        LEAX  1,X                       ADD ONE
3104   B581  9F 25                        STX   FRESPC                    SAVE START ADDRESS OF NEWLY RESERVED SPACE
3105   B583  35 84                        PULS  B,PC                      RESTORE NUMBER OF BYTES RESERVED AND RETURN
3106   B585  C6 1A              LB585     LDB   #2*13                     'OUT OF STRING SPACE' ERROR
3107   B587  03 07                        COM   GARBFL                    TOGGLE REORGANIZATION FLAG
3108   B589  27 CA                        BEQ   LB555                     ERROR IF FRESHLY REORGANIZED
3109   B58B  8D 04                        BSR   LB591                     GO REORGANIZE STRING SPACE
3110   B58D  35 04                        PULS  B                         GET BACK THE NUMBER OF BYTES TO RESERVE
3111   B58F  20 DE                        BRA   LB56F                     TRY TO RESERVE ACCB BYTES AGAIN
3112                            * REORGANIZE THE STRING SPACE
3113   B591  9E 27              LB591     LDX   MEMSIZ                    GET THE TOP OF STRING SPACE
3114   B593  9F 23              LB593     STX   STRTAB                    SAVE TOP OF UNORGANIZED STRING SPACE
3115   B595  4F                           CLRA                            * ZERO OUT ACCD
```

```
3116    B596  5F                        CLRB                        * AND RESET VARIABLE
3117    B597  DD 4B                     STD   V4B                   * POINTER TO Ø
3118    B599  9E 21                     LDX   FRETOP                POINT X TO START OF STRING SPACE
3119    B59B  9F 47                     STX   V47                   SAVE POINTER IN V47
3120    B59D  8E Ø1 A9                  LDX   #STRSTK               POINT X TO START OF STRING DESCRIPTOR STACK
3121    B5A0  9C ØB             LB5AØ   CMPX  TEMPPT                COMPARE TO ADDRESS OF NEXT AVAILABLE DESCRIPTOR
3122    B5A2  27 Ø4                     BEQ   LB5A8                 BRANCH IF TOP OF STRING STACK
3123    B5A4  8D 32                     BSR   LB5D8                 CHECK FOR STRING IN UNORGANIZED STRING SPACE
3124    B5A6  2Ø F8                     BRA   LB5AØ                 KEEP CHECKING
3125    B5A8  9E 1B             LB5A8   LDX   VARTAB                GET THE END OF BASIC PROGRAM
3126    B5AA  9C 1D             LB5AA   CMPX  ARYTAB                COMPARE TO END OF VARIABLES
3127    B5AC  27 Ø4                     BEQ   LB5B2                 BRANCH IF AT TOP OF VARIABLES
3128    B5AE  8D 22                     BSR   LB5D2                 CHECK FOR STRING IN UNORGANIZED STRING SPACE
3129    B5BØ  2Ø F8                     BRA   LB5AA                 KEEP CHECKING VARIABLES
3130    B5B2  9F 41             LB5B2   STX   V41                   SAVE ADDRESS OF THE END OF VARIABLES
3131    B5B4  9E 41             LB5B4   LDX   V41                   GET CURRENT ARRAY POINTER
3132    B5B6  9C 1F             LB5B6   CMPX  ARYEND                COMPARE TO THE END OF ARRAYS
3133    B5B8  27 35                     BEQ   LB5EF                 BRANCH IF AT END OF ARRAYS
3134    B5BA  EC Ø2                     LDD   2,X                   GET LENGTH OF ARRAY AND DESCRIPTOR
3135    B5BC  D3 41                     ADDD  V41                   * ADD TO CURRENT ARRAY POINTER
3136    B5BE  DD 41                     STD   V41                   * AND SAVE IT
3137    B5CØ  A6 Ø1                     LDA   1,X                   GET 1ST CHARACTER OF VARIABLE NAME
3138    B5C2  2A FØ                     BPL   LB5B4                 BRANCH IF NUMERIC ARRAY
3139    B5C4  E6 Ø4                     LDB   4,X                   GET THE NUMBER OF DIMENSIONS IN THIS ARRAY
3140    B5C6  58                        ASLB                        MULTIPLY BY 2
3141    B5C7  CB Ø5                     ADDB  #5                    ADD FIVE BYTES (VARIABLE NAME, ARRAY
3142                            *                                   LENGTH, NUMBER DIMENSIONS)
3143    B5C9  3A                        ABX                         X NOW POINTS TO START OF ARRAY ELEMENTS
3144    B5CA  9C 41             LB5CA   CMPX  V41                   AT END OF THIS ARRAY?
3145    B5CC  27 E8                     BEQ   LB5B6                 YES - CHECK FOR ANOTHER
3146    B5CE  8D Ø8                     BSR   LB5D8                 CHECK FOR STRING LOCATED IN
3147                            *                                   UNORGANIZED STRING SPACE
3148    B5DØ  2Ø F8                     BRA   LB5CA                 KEEP CHECKING ELEMENTS IN THIS ARRAY
3149    B5D2  A6 Ø1             LB5D2   LDA   1,X                   GET F1RST BYTE OF VARIABLE NAME
3150    B5D4  3Ø Ø2                     LEAX  2,X                   MOVE POINTER TO DESCRIPTOR
3151    B5D6  2A 14                     BPL   LB5EC                 BRANCH IF VARIABLE IS NUMERIC
3152                            * SEARCH FOR STRING - ENTER WITH X POINTING TO
3153                            * THE STRING DESCRIPTOR. IF STRING IS STORED
3154                            * BETWEEN V47 AND STRTAB, SAVE DESCRIPTOR POINTER
3155                            * IN V4B AND RESET V47 TO STRING ADDRESS
3156    B5D8  E6 84             LB5D8   LDB   ,X                    GET THE LENGTH OF THE STRING
3157    B5DA  27 1Ø                     BEQ   LB5EC                 BRANCH IF NULL - NO STRING
3158    B5DC  EC Ø2                     LDD   2,X                   GET STARTING ADDRESS OF THE STRING
3159    B5DE  1Ø 93 23                  CMPD  STRTAB                COMPARE TO THE START OF STRING VARIABLES
3160    B5E1  22 Ø9                     BHI   LB5EC                 BRANCH IF THIS STRING IS STORED IN
3161                            *                                   THE STRING VARIABLES
3162    B5E3  1Ø 93 47                  CMPD  V47                   COMPARE TO START OF STRING SPACE
3163    B5E6  23 Ø4                     BLS   LB5EC                 BRANCH IF NOT STORED IN THE STRING SPACE
3164    B5E8  9F 4B                     STX   V4B                   SAVE VARIABLE POINTER IF STORED IN STRING SPACE
3165    B5EA  DD 47                     STD   V47                   SAVE STRING STARTING ADDRESS
3166    B5EC  3Ø Ø5             LB5EC   LEAX  5,X                   MOVE TO NEXT VARIABLE DESCRIPTOR
3167    B5EE  39                LB5EE   RTS
3168    B5EF  9E 4B             LB5EF   LDX   V4B                   GET ADDRESS OF THE DESCRIPTOR FOR THE
3169                            *                                   STRING WHICH IS STORED IN THE HIGHEST RAM ADDRESS IN
3170                            *                                   THE UNORGANIZED STRING SPACE
3171    B5F1  27 FB                     BEQ   LB5EE                 BRANCH IF NONE FOUND AND REORGANIZATION DONE
3172    B5F3  4F                        CLRA                        CLEAR MS BYTE OF LENGTH
3173    B5F4  E6 84                     LDB   ,X                    GET LENGTH OF STRING
3174    B5F6  5A                        DECB                        SUBTRACT ONE
3175    B5F7  D3 47                     ADDD  V47                   ADD LENGTH OF STRING TO ITS STARTING ADDRESS
3176    B5F9  DD 43                     STD   V43                   SAVE AS MOVE STARTING ADDRESS
3177    B5FB  9E 23                     LDX   STRTAB                POINT X TO THE START OF ORGANIZED STRING VARIABLES
3178    B5FD  9F 41                     STX   V41                   SAVE AS MOVE ENDING ADDRESS
3179    B5FF  BD AC 2Ø                  JSR   LAC2Ø                 MOVE STRING FROM CURRENT POSITION TO THE
3180                            *                                   TOP OF UNORGANIZED STRING SPACE
3181    B602  9E 4B                     LDX   V4B                   POINT X TO STRING DESCRIPTOR
3182    B604  DC 45                     LDD   V45                   * GET NEW STARTING ADDRESS OF STRING AND
3183    B606  ED Ø2                     STD   2,X                   * SAVE IT IN DESCRIPTOR
3184    B608  9E 45                     LDX   V45                   GET NEW TOP OF UNORGANIZED STRING SPACE
3185    B60A  3Ø 1F                     LEAX  -1,X                  MOVE POINTER BACK ONE
3186  > B60C  7E B5 93                  JMP   LB593                 JUMP BACK AND REORGANIZE SOME MORE
3187
3188                            * CONCATENATE TWO STRINGS
3189    B60F  DC 52             LB60F   LDD   FPA0+2                * GET DESCRIPTOR ADDRESS OF STRING A
3190    B611  34 Ø6                     PSHS  B,A                   * AND SAVE IT ON THE STACK
3191    B613  BD B2 23                  JSR   LB223                 GET DESCRIPTOR ADDRESS OF STRING B
3192    B616  BD B1 46                  JSR   LB146                 'TM' ERROR IF NUMERIC VARIABLE
3193    B619  35 1Ø                     PULS  X                     * POINT X TO STRING A DESCRIPTOR
3194    B61B  9F 62                     STX   RESSGN                * ADDRESS AND SAVE IT IN RESSGN
3195    B61D  E6 84                     LDB   ,X                    GET LENGTH OF STRING A
3196    B61F  9E 52                     LDX   FPA0+2                POINT X TO DESCRIPTOR OF STRING B
3197    B621  EB 84                     ADDB  ,X                    ADD LENGTH OF STRING B TO STR1NG A
3198    B623  24 Ø5                     BCC   LB62A                 BRANCH IF LENGTH < 256
3199    B625  C6 1C                     LDB   #2*14                 'STRING TOO LONG' ERROR IF LENGTH > 255
3200    B627  7E AC 46                  JMP   LAC46                 JUMP TO ERROR SERVICING ROUTINE
3201    B62A  BD B5 ØD          LB62A   JSR   LB50D                 RESERVE ROOM IN STRING SPACE FOR NEW STRING
3202    B62D  9E 62                     LDX   RESSGN                GET DESCRIPTOR ADDRESS OF STRING A
3203    B62F  E6 84                     LDB   ,X                    GET LENGTH OF STRING A
3204    B631  8D 1Ø                     BSR   LB643                 MOVE STRING A INTO RESERVED BUFFER IN STRING SPACE
```

```
3205   B633  9E 4D                    LDX   V4D                    GET DESCRIPTOR ADDRESS OF STRING B
3206   B635  8D 22                    BSR   LB659                  GET LENGTH AND ADDRESS OF STRING B
3207   B637  8D 0C                    BSR   LB645                  MOVE STRING B INTO REST OF RESERVED BUFFER
3208   B639  9E 62                    LDX   RESSGN                 POINT X TO DESCRIPTOR OF STRING A
3209   B63B  8D 1C                    BSR   LB659                  DELETE STRING A IF LAST STRING ON STRING STACK
3210   B63D  BD B5 4C                 JSR   LB54C                  PUT STRING DESCRIPTOR ON THE STRING STACK
3211   B640  7E B1 68                 JMP   LB168                  BRANCH BACK TO EXPRESSION EVALUATION
3212
3213                         * MOVE (B) BYTES FROM 2,X TO FRESPC
3214   B643  AE 02            LB643   LDX   2,X                    POINT X TO SOURCE ADDRESS
3215   B645  DE 25            LB645   LDU   FRESPC                 POINT U TO DESTINATION ADDRESS
3216   B647  5C                       INCB                         COMPENSATION FOR THE DECB BELOW
3217   B648  20 04                    BRA   LB64E                  GO MOVE THE BYTES
3218                         * MOVE B BYTES FROM (X) TO (U)
3219   B64A  A6 80            LB64A   LDA   ,X+                   * GET A SOURCE BYTE AND MOVE IT
3220   B64C  A7 C0                    STA   ,U+                    * TO THE DESTINATION
3221   B64E  5A               LB64E   DECB                         DECREMENT BYTE COUNTER
3222   B64F  26 F9                    BNE   LB64A                  BRANCH IF ALL BYTES NOT MOVED
3223   B651  DF 25                    STU   FRESPC                 SAVE ENDING ADDRESS IN FRESPC
3224   B653  39                       RTS
3225                         * RETURN LENGTH (ACCB) AND ADDRESS (X) OF
3226                         * STRING WHOSE DESCRIPTOR IS IN FPA0+2
3227                         * DELETE THE STRING IF IT IS THE LAST ONE
3228                         * PUT ON THE STRING STACK. REMOVE STRING FROM STRING
3229                         * SPACE IF IT IS AT THE BOTTOM OF STRING VARIABLES.
3230   B654  BD B1 46         LB654   JSR   LB146                  'TM' ERROR IF VARIABLE TYPE = NUMERIC
3231   B657  9E 52            LB657   LDX   FPA0+2                 GET ADDRESS OF SELECTED STRING DESCRIPTOR
3232   B659  E6 84            LB659   LDB   ,X                     GET LENGTH OF STRING
3233   B65B  8D 18                    BSR   LB675                  * CHECK TO SEE IF THIS STRING DESCRIPTOR WAS
3234   B65D  26 13                    BNE   LB672                  * THE LAST ONE PUT ON THE STRING STACK AND
3235                         *                                     * BRANCH IF NOT
3236   B65F  AE 07                    LDX   5+2,X                  GET START ADDRESS OF STRING JUST REMOVED
3237   B661  30 1F                    LEAX  -1,X                   MOVE POINTER DOWN ONE
3238   B663  9C 23                    CMPX  STRTAB                 COMPARE TO START OF STRING VARIABLES
3239   B665  26 08                    BNE   LB66F                  BRANCH IF THIS STRING IS NOT AT THE BOTTOM
3240                         *                                     OF STRING VARIABLES
3241   B667  34 04                    PSHS  B                      SAVE LENGTH; ACCA WAS CLEARED
3242   B669  D3 23                    ADDD  STRTAB                 * ADD THE LENGTH OF THE JUST REMOVED STRING
3243   B66B  DD 23                    STD   STRTAB                 * TO THE START OF STRING VARIABLES - THIS WILL
3244                         *                                     * REMOVE THE STRING FROM THE STRING SPACE
3245   B66D  35 04                    PULS  B                      RESTORE LENGTH
3246   B66F  30 01            LB66F   LEAX  1,X                    ADD ONE TO POINTER
3247   B671  39                       RTS
3248   B672  AE 02            LB672   LDX   2,X                    *POINT X TO ADDRESS OF STRING NOT
3249   B674  39                       RTS                          *ON THE STRING STACK
3250                         * REMOVE STRING FROM STRING STACK. ENTER WITH X
3251                         * POINTING TO A STRING DESCRIPTOR - DELETE THE
3252                         * STRING FROM STACK IF IT IS ON TOP OF THE
3253                         * STACK. IF THE STRING IS DELETED, SET THE ZERO FLAG
3254   B675  9C 0D            LB675   CMPX  LASTPT                 *COMPARE TO LAST USED DESCRIPTOR ADDRESS
3255   B677  26 07                    BNE   LB680                  *ON THE STRING STACK, RETURN IF DESCRIPTOR
3256                         *                                     *ADDRESS NOT ON THE STRING STACK
3257   B679  9F 0B                    STX   TEMPPT                 SAVE LAST USED DESCRIPTOR AS NEXT AVAILABLE
3258   B67B  30 1B                    LEAX  -5,X                   * MOVE LAST USED DESCRIPTOR BACK 5 BYTES
3259   B67D  9F 0D                    STX   LASTPT                 * AND SAVE AS THE LAST USED DESCRIPTOR ADDR
3260   B67F  4F                       CLRA                         SET ZERO FLAG
3261   B680  39               LB680   RTS
3262
3263                         * LEN
3264   B681  8D 03            LEN     BSR   LB686                  POINT X TO PROPER STRING AND GET LENGTH
3265   B683  7E B4 F3         LB683   JMP   LB4F3                  CONVERT ACCB TO FP NUMBER IN FPA0
3266                         * POINT X TO STRING ADDRESS LOAD LENGTH INTO
3267                         * ACCB. ENTER WITH THE STRING DESCRIPTOR IN
3268                         * BOTTOM TWO BYTES OF FPA0
3269   B686  8D CC            LB686   BSR   LB654                  GET LENGTH AND ADDRESS OF STRING
3270   B688  0F 06                    CLR   VALTYP                 SET VARIABLE TYPE TO NUMERIC
3271   B68A  5D                       TSTB                         SET FLAGS ACCORDING TO LENGTH
3272   B68B  39                       RTS
3273
3274                         * CHR$
3275 > B68C  BD B7 0E         CHR     JSR   LB70E                  CONVERT FPA0 TO AN INTEGER IN ACCD
3276   B68F  C6 01            LB68F   LDB   #1                     * RESERVE ONE BYTE IN
3277   B691  BD B5 6D                 JSR   LB56D                  * THE STRING SPACE
3278   B694  96 53                    LDA   FPA0+3                 GET ASCII STRING VALUE
3279   B696  BD B5 11                 JSR   LB511                  SAVE RESERVED STRING DESCRIPTOR IN TEMP DESCRIPTOR
3280   B699  A7 84                    STA   ,X                     SAVE THE STRING (IT S ONLY ONE BYTE)
3281   B69B  32 62            LB69B   LEAS  2,X                    PURGE THE RETURN ADDRESS OFF OF THE STACK
3282   B69D  7E B5 4C         LB69D   JMP   LB54C                  PUT TEMP DESCRIPTOR DATA ONTO STRING STACK
3283
3284                         * ASC$
3285   B6A0  8D 02            ASC     BSR   LB6A4                  PUT 1ST CHARACTER OF STRING INTO ACCB
3286   B6A2  20 DF                    BRA   LB683                  CONVERT ACCB INTO FP NUMBER IN FPA0
3287   B6A4  8D E0            LB6A4   BSR   LB686                  POINT X TO STRING DESCRIPTOR
3288   B6A6  27 5E                    BEQ   LB706                  'FC' ERROR IF NULL STRING
3289   B6A8  E6 84                    LDB   ,X                     GET FIRST BYTE OF STRING
3290   B6AA  39                       RTS
3291
3292                         * LEFT$
3293   B6AB  8D 48            LEFT    BSR   LB6F5                  GET ARGUMENTS FROM STACK
```

```
3294   B6AD  4F                        CLRA                        CLEAR STRING POINTER OFFSET - OFFSET = Ø FOR LEFT$
3295   B6AE  E1 84             LB6AE   CMPB  ,X                    * COMPARE LENGTH PARAMETER TO LENGTH OF
3296   B6Ø  23 Ø3                      BLS   LB6B5                 * STRING AND BRANCH IF LENGTH OF STRING
3297                           *                                   >= LENGTH PARAMETER
3298   B6B2  E6 84                     LDB   ,X                    USE LENGTH OF STRING OTHERWISE
3299   B6B4  4F                        CLRA                        CLEAR STRING POINTER OFFSET (Ø FOR LEFT$)
33ØØ   B6B5  34 Ø6             LB6B5   PSHS  B,A                   PUSH PARAMETERS ONTO STACK
33Ø1   B6B7  BD B5 ØF                  JSR   LB5ØF                 RESERVE ACCB BYTES IN THE STRING SPACE
33Ø2   B6BA  9E 4D                      LDX   V4D                   POINT X TO STRING DESCRIPTOR
33Ø3   B6BC  8D 9B                      BSR   LB659                 GET ADDRESS OF OLD STRING (X=ADDRESS)
33Ø4   B6BE  35 Ø4                      PULS  B                     * PULL STRING POINTER OFFSET OFF OF THE STACK
33Ø5   B6CØ  3A                         ABX                         * AND ADD IT TO STRING ADDRESS
33Ø6   B6C1  35 Ø4                      PULS  B                     PULL LENGTH PARAMETER OFF OF THE STACK
33Ø7   B6C3  BD B6 45                   JSR   LB645                 MOVE ACCB BYTES FROM (X) TO [FRESPC]
33Ø8   B6C6  2Ø D5                      BRA   LB69D                 PUT TEMP STRING DESCRIPTOR ONTO THE STRING STACK
3309
3310                           * RIGHT$
3311   B6C8  8D 2B             RIGHT   BSR   LB6F5                 GET ARGUMENTS FROM STACK
3312   B6CA  AØ 84                     SUBA  ,X                    ACCA=LENGTH PARAMETER - LENGTH OF OLD STRING
3313   B6CC  4Ø                        NEGA                        NOW ACCA = LENGTH OF OLD STRING
3314   B6CD  2Ø DF                     BRA   LB6AE                 PUT NEW STRING IN THE STRING SPACE
3315
3316                           * MID$
3317   B6CF  C6 FF             MID     LDB   #$FF                  * GET DEFAULT VALUE OF LENGTH AND
3318   B6D1  D7 53                     STB   FPAØ+3                * SAVE IT IN FPAØ
3319   B6D3  9D A5                     JSR   GETCCH                GET CURRENT CHARACTER FROM BASIC
3320   B6D5  81 29                     CMPA  #')                   ARGUMENT DELIMITER?
3321   B6D7  27 Ø5                     BEQ   LB6DE                 YES - NO LENGTH PARAMETER GIVEN
3322   B6D9  BD B2 6D                  JSR   LB26D                 SYNTAX CHECK FOR COMMA
3323   B6DC  8D 2D                     BSR   LB7ØB                 EVALUATE NUMERIC EXPRESSION (LENGTH)
3324   B6DE  8D 15             LB6DE   BSR   LB6F5                 GET ARGUMENTS FROM STACK
3325   B6EØ  27 24                     BEQ   LB7Ø6                 'FC' ERROR IF NULL STRING
3326   B6E2  5F                        CLRB                        CLEAR LENGTH COUNTER (DEFAULT VALUE)
3327   B6E3  4A                        DECA                        *SUOTRACT ONE FROM POSITION PARAMETER (THESE
3328   B6E4  A1 84                     CMPA  ,X                    *ROUTINES EXPECT 1ST POSITION TO BE ZERO, NOT ONE)
3329                           *                                   *AND COMPARE IT TO LENGTH OF OLD STRING
3330   B6E6  24 CD                     BCC   LB6B5                 IF POSITION > LENGTH OF OLD STRING, THEN NEW
3331                           *                                   STRING WILL BE A NULL STRING
3332   B6E8  1F 89                     TFR   A,B                   SAVE ABSOLUTE POSITION PARAMETER IN ACCB
3333   B6EA  EØ 84                     SUBB  ,X                    ACCB=POSITION-LENGTH OF OLD STRING
3334   B6EC  5Ø                        NEGB                        NOW ACCB=LENGTH OF OLDSTRING-POSITION
3335   B6ED  D1 53                     CMPB  FPAØ+3                *IF THE AMOUNT OF OLD STRING TO THE RIGHT OF
3336   B6EF  23 C4                     BLS   LB6B5                 *POSITION IS <= THE LENGTH PARAMETER, BRANCH AND
3337                           *USE ALL OF THE STRING TO THE RIGHT OF THE POSITION
3338                           *INSTEAD OF THE LENGTH PARAMETER
3339   B6F1  D6 53                     LDB   FPAØ+3                GET LENGTH OF NEW STRING
3340   B6F3  2Ø CØ                     BRA   LB6B5                 PUT NEW STRING IN STRING SPACE
3341                           * DO A SYNTAX CHECK FOR ")", THEN PULL THE PREVIOUSLY CALCULATED NUMERIC
3342                           * ARGUMENT (ACCD) AND STRING ARGUMENT DESCRIPTOR ADDR OFF OF THE STACK
3343   B6F5  BD B2 67          LB6F5   JSR   LB267                 SYNTAX CHECK FOR A ")"
3344   B6F8  EE E4                     LDU   ,S                    LOAD THE RETURN ADDRESS INTO U REGISTER
3345   B6FA  AE 65                     LDX   5,S                   * GET ADDRESS OF STRING AND
3346   B6FC  9F 4D                     STX   V4D                   * SAVE IT IN V4D
3347   B6FE  A6 64                     LDA   4,S                   = PUT LENGTH OF STRING IN
3348   B7ØØ  E6 64                     LDB   4,S                   = BOTH ACCA AND ACCB
3349   B7Ø2  32 67                     LEAS  7,S                   REMOVE DESCRIPTOR AND RETURN ADDRESS FROM STACK
3350   B7Ø4  1F 35                     TFR   U,PC                  JUMP TO ADDRESS IN U REGISTER
3351   B7Ø6  7E B4 4A          LB7Ø6   JMP   LB44A                 'ILLEGAL FUNCTION CALL'
3352                           * EVALUATE AN EXPRESSION - RETURN AN INTEGER IN
3353                           * ACCB - 'FC' ERROR IF EXPRESSION > 255
3354   B7Ø9  9D 9F             LB7Ø9   JSR   GETNCH                GET NEXT BASIC INPUT CHARACTER
3355   B7ØB  BD B1 41          LB7ØB   JSR   LB141                 EVALUATE A NUMERIC EXPRESSION
3356   B7ØE  BD B3 E9          LB7ØE   JSR   LB3E9                 CONVERT FPAØ TO INTEGER IN ACCD
3357   B711  4D                        TSTA                        TEST MS BYTE OF INTEGER
3358   B712  26 F2                     BNE   LB7Ø6                 'FC' ERROR IF EXPRESSION > 255
3359   B714  ØE A5                     JMP   GETCCH                GET CURRENT INPUT CHARACTER FROM BASIC
3360
3361                           * VAL
3362   B716  BD B6 86          VAL     JSR   LB686                 POINT X TO STRING ADDRESS
3363   B719  1Ø 27 Ø3 1C               LBEQ  LBA39                 IF NULL STRING SET FPAØ
3364   B71D  DE A6                     LDU   CHARAD                SAVE INPUT POINTER IN REGISTER U
3365   B71F  9F A6                     STX   CHARAD                POINT INPUT POINTER TO ADDRESS OF STRING
3366   B721  3A                        ABX                         MOVE POINTER TO END OF STRING TERMINATOR
3367   B722  A6 84                     LDA   ,X                    GET LAST BYTE OF STRING
3368   B724  34 52                     PSHS  U,X,A                 SAVE INPUT POINTER, STRING TERMINATOR
3369                           *                                   ADDRESS AND CHARACTER
3370   B726  6F 84                     CLR   ,X                    CLEAR STRING TERMINATOR : FOR ASCII - FP CONVERSION
3371   B728  9D A5                     JSR   GETCCH                GET CURRENT CHARACTER FROM BASIC
3372   B72A  BD BD 12                  JSR   LBD12                 CONVERT AN ASCII STRING TO FLOATING POINT
3373   B72D  35 52                     PULS  A,X,U                 RESTORE CHARACTERS AND POINTERS
3374   B72F  A7 84                     STA   ,X                    REPLACE STRING TERMINATOR
3375   B731  DF A6                     STU   CHARAD                RESTORE INPUT CHARACTER
3376   B733  39                        RTS
3377
3378   B734  8D Ø7             LB734   BSR   LB73D                 * EVALUATE AN EXPRESSION, RETURN
3379   B736  9F 2B                     STX   BINVAL                * THE VALUE IN X; STORE IT IN BINVAL
3380   B738  BD B2 6D          LB738   JSR   LB26D                 SYNTAX CHECK FOR A COMMA
3381   B73B  2Ø CE                     BRA   LB7ØB                 EVALUATE EXPRESSION IN RANGE Ø <= X < 256
3382                           * EVALUATE EXPRESSION : RETURN INTEGER PORTION IN X - 'FC' ERROR IF
```

```
3383                            * EXPRESSION IS NEGATIVE OR > 32767, I.E. NOT A LEGAL POSITIVE INTEGER.
3384   B73D  BD B1 41           LB73D     JSR   LB141                     EVALUATE NUMERIC EXPRESSION
3385   B740  96 54              LB740     LDA   FP0SGN                    GET SIGN OF FPA0 MANTISSA
3386   B742  2B C2                       BMI   LB706                     'ILLEGAL FUNCTION CALL' IF NEGATIVE
3387   B744  96 4F                       LDA   FP0EXP                    GET EXPONENT OF FPA0
3388   B746  81 90                       CMPA  #$90                      COMPARE TO LARGEST POSITIVE INTEGER
3389   B748  22 BC                       BHI   LB706                     'ILLEGAL FUNCTION CALL' IF TOO LARGE
3390   B74A  BD BC C8                    JSR   LBCC8                     SHIFT BINARY POINT TO EXTREME RIGHT OF FPA0
3391   B74D  9E 52                       LDX   FPA0+2                    LOAD X WITH LOWER TWO BYTES OF FPA0
3392   B74F  39                          RTS
3393
3394                            * PEEK
3395   B750  8D EE              PEEK      BSR   LB740                     CONVERT FPA0 TO INTEGER IN REGISTER X
3396   B752  E6 84                       LDB   ,X                        GET THE VALUE BEING 'PEEK'ED
3397   B754  7E B4 F3                    JMP   LB4F3                     CONVERT ACCB INTO A FP NUMBER
3398
3399                            * POKE
3400   B757  8D DB              POKE      BSR   LB734                     EVALUATE 2 EXPRESSIONS
3401   B759  9E 2B                       LDX   BINVAL                    GET THE ADDRESS TO BE 'POKE'ED
3402   B75B  E7 84                       STB   ,X                        STORE THE DATA IN THAT ADDRESS
3403   B75D  39                          RTS
3404
3405                            * LLIST
3406   B75E  C6 FE              LLIST     LDB   #-2                       * SET DEVICE NUMBER TO
3407   B760  D7 6F                       STB   DEVNUM                    * PRINTER
3408   B762  9D A5                       JSR   GETCCH                    GET CURRENT CHARACTER FROM BASIC
3409
3410                            * LIST
3411   B764  34 01              LIST      PSHS  CC                        SAVE ZERO FLAG ON STACK
3412   B766  BD AF 67                    JSR   LAF67                     CONVERT DECIMAL LINE NUMBER TO BINARY
3413   B769  BD AD 01                    JSR   LAD01                     * FIND RAM ADDRESS OF THAT LINE NUMBER AND
3414   B76C  9F 66                       STX   LSTTXT                    * SAVE IT IN LSTTXT
3415   B76E  35 01                       PULS  CC                        GET ZERO FLAG FROM STACK
3416   B770  27 12                       BEQ   LB784                     BRANCH IF END OF LINE
3417   B772  9D A5                       JSR   GETCCH                    GET CURRENT CHARACTER FROM BASIC
3418   B774  27 13                       BEQ   LB789                     BRANCH IF END OF LINE
3419   B776  81 AC                       CMPA  #$AC                      MINUS TOKEN (IS IT A RANGE OF LINE NUMBERS?)
3420   B778  26 09                       BNE   LB783                     NO - RETURN
3421   B77A  9D 9F                       JSR   GETNCH                    GET NEXT CHARACTER FROM BASIC
3422   B77C  27 06                       BEQ   LB784                     BRANCH IF END OF LINE
3423   B77E  BD AF 67                    JSR   LAF67                     GET ENDING LINE NUMBER
3424   B781  27 06                       BEQ   LB789                     BRANCH IF LEGAL LINE NUMBER
3425   B783  39                 LB783     RTS
3426                            * LIST THE ENTIRE PROGRAM
3427   B784  CE FF FF           LB784     LDU   #$FFFF                    * SET THE DEFAULT ENDING LINE NUMBER
3428   B787  DF 2B                       STU   BINVAL                    * TO $FFFF
3429   B789  32 62              LB789     LEAS  2,S                       PURGE RETURN ADDRESS FROM THE STACK
3430   B78B  9E 66                       LDX   LSTTXT                    POINT X TO STARTING LINE ADDRESS
3431   B78D  BD B9 5C           LB78D     JSR   LB95C                     MOVE CURSOR TO START OF A NEW LINE
3432   B790  BD A5 49                    JSR   LA549                     CHECK FOR A BREAK OR PAUSE
3433   B793  EC 84                       LDD   ,X                        GET ADDRESS OF NEXT BASIC LINE
3434   B795  26 08                       BNE   LB79F                     BRANCH IF NOT END OF PROGRAM
3435   B797  BD A4 2D           LB797     JSR   LA42D                     CHECK CLOSE FILE HANDLER
3436   B79A  0F 6F                       CLR   DEVNUM                    SET DEVICE NUMBER TO SCREEN
3437   B79C  7E AC 73                    JMP   LAC73                     RETURN TO BASIC S MAIN INPUT LOOP
3438   B79F  9F 66              LB79F     STX   LSTTXT                    SAVE NEW STARTING LINE ADDRESS
3439   B7A1  EC 02                       LDD   2,X                       * GET THE LINE NUMBER OF THIS LINE AND
3440   B7A3  10 93 2B                    CMPD  BINVAL                    * COMPARE IT TO ENDING LINE NUMBER
3441   B7A6  22 EF                       BHI   LB797                     EXIT IF LINE NUMBER > ENDING LINE NUMBER
3442   B7A8  BD BD CC                    JSR   LBDCC                     PRINT THE NUMBER IN ACCD ON SCREEN IN DECIMAL
3443   B7AB  BD B9 AC                    JSR   LB9AC                     SEND A SPACE TO CONSOLE OUT
3444   B7AE  9E 66                       LDX   LSTTXT                    GET RAM ADDRESS OF THIS LINE
3445   B7B0  8D 10                       BSR   LB7C2                     UNCRUNCH A LINE
3446   B7B2  AE 9F 00 66                 LDX   [LSTTXT]                  POINT X TO START OF NEXT LINE
3447   B7B6  CE 02 DD                    LDU   #LINBUF+1                 POINT U TO BUFFER FULL OF UNCRUNCHED LINE
3448   B7B9  A6 C0              LB7B9     LDA   ,U+                       GET A BYTE FROM THE BUFFER
3449   B7BB  27 D0                       BEQ   LB78D                     BRANCH IF END OF BUFFER
3450   B7BD  BD B9 B1                    JSR   LB9B1                     SEND CHARACTER TO CONSOLE OUT
3451   B7C0  20 F7                       BRA   LB7B9                     GET ANOTHER CHARACTER
3452
3453                            * UNCRUNCH A LINE INTO BASIC S LINE INPUT BUFFER
3454   B7C2  BD 01 A6           LB7C2     JSR   RVEC24                    HOOK INTO RAM
3455   B7C5  30 04                       LEAX  4,X                       MOVE POINTER PAST ADDRESS OF NEXT LINE AND LINE NUMBER
3456   B7C7  10 8E 02 DD                 LDY   #LINBUF+1                 UNCRUNCH LINE INTO LINE INPUT BUFFER
3457   B7CB  A6 80              LB7CB     LDA   ,X+                       GET A CHARACTER
3458   B7CD  27 51                       BEQ   LB820                     BRANCH IF END OF LINE
3459   B7CF  2B 15                       BMI   LB7E6                     BRANCH IF IT S A TOKEN
3460   B7D1  81 3A                       CMPA  #':                       CHECK FOR END OF SUB LINE
3461   B7D3  26 0D                       BNE   LB7E2                     BRNCH IF NOT END OF SUB LINE
3462   B7D5  E6 84                       LDB   ,X                        GET CHARACTER FOLLOWING COLON
3463   B7D7  C1 84                       CMPB  #$84                      TOKEN FOR ELSE?
3464   B7D9  27 F0                       BEQ   LB7CB                     YES - DON T PUT IT IN BUFFER
3465   B7DB  C1 83                       CMPB  #$83                      TOKEN FOR REMARK?
3466   B7DD  27 EC                       BEQ   LB7CB                     YES - DON T PUT IT IN BUFFER
3467   B7DF  8C                          FCB   SKP2                      SKIP TWO BYTES
3468   B7E0  86 21              LB7E0     LDA   #'!                       EXCLAMATION POINT
3469   B7E2  8D 30              LB7E2     BSR   LB814                     PUT CHARACTER IN BUFFER
3470   B7E4  20 E5                       BRA   LB7CB                     GET ANOTHER CHARACTER
3471                            * UNCRUNCH A TOKEN
```

```
3472   B7E6  CE 01 16          LB7E6    LDU   #COMVEC-10               FIRST DO COMMANDS
3473   B7E9  81 FF                     CMPA  #$FF                      CHECK FOR SECONDARY TOKEN
3474   B7EB  26 04                     BNE   LB7F1                     BRANCH IF NON SECONDARY TOKEN
3475   B7ED  A6 80                     LDA   ,X+                       GET SECONDARY TOKEN
3476   B7EF  33 45                     LEAU  5,U                       BUMP IT UP TO SECONDARY FUNCTIONS
3477   B7F1  84 7F              LB7F1  ANDA  #$7F                      MASK OFF BIT 7 OF TOKEN
3478   B7F3  33 4A              LB7F3  LEAU  10,U                      MOVE TO NEXT COMMAND TABLE
3479   B7F5  6D C4                     TST   ,U                        IS THIS TABLE ENABLED?
3480   B7F7  27 E7                     BEQ   LB7E0                     NO - ILLEGAL TOKEN
3481   B7F9  A0 C4                     SUBA  ,U                        SUBTRACT THE NUMBER OF TOKENS FROM THE CURRENT TOKEN NUMBER
3482   B7FB  2A F6                     BPL   LB7F3                     BRANCH IF TOKEN NOT IN THIS TABLE
3483   B7FD  AB C4                     ADDA  ,U                        RESTORE TOKEN NUMBER RELATIVE TO THIS TABLE
3484   B7FF  EE 41                     LDU   1,U                       POINT U TO COMMAND DICTIONARY TABLE
3485   B801  4A                LB801  DECA                            DECREMENT TOKEN NUMBER
3486   B802  2B 06                     BMI   LB80A                     BRANCH IF THIS IS THE CORRECT TOKEN
3487                           * SKIP THROUGH DICTIONARY TABLE TO START OF NEXT TOKEN
3488   B804  6D C0              LB804  TST   ,U+                       GRAB A BYTE
3489   B806  2A FC                     BPL   LB804                     BRANCH IF BIT 7 NOT SET
3490   B808  20 F7                     BRA   LB801                     GO SEE IF THIS IS THE CORRECT TOKEN
3491   B80A  A6 C4              LB80A  LDA   ,U                        GET A CHARACTER FROM DICTIONARY TABLE
3492   B80C  8D 06                     BSR   LB814                     PUT CHARACTER IN BUFFER
3493   B80E  6D C0                     TST   ,U+                       CHECK FOR START OF NEXT TOKEN
3494   B810  2A F8                     BPL   LB80A                     BRANCH IF NOT DONE WITH THIS TOKEN
3495   B812  20 B7                     BRA   LB7CB                     GO GET ANOTHER CHARACTER
3496   B814  10 8C 03 D6       LB814  CMPY  #LINBUF+LBUFMX            TEST FOR END OF LINE INPUT BUFFER
3497   B818  24 06                     BCC   LB820                     BRANCH IF AT END OF BUFFER
3498   B81A  84 7F                     ANDA  #$7F                      MASK OFF BIT 7
3499   B81C  A7 A0                     STA   ,Y+                       * SAVE CHARACTER IN BUFFER AND
3500   B81E  6F A4                     CLR   ,Y                        * CLEAR NEXT CHARACTER SLOT IN BUFFER
3501   B820  39                LB820  RTS
3502                           *
3503                           * CRUNCH THE LINE THAT THE INPUT POINTER IS
3504                           * POINTING TO INTO THE LINE INPUT BUFFER
3505                           * RETURN LENGTH OF CRUNCHED LINE IN ACCD
3506                           *
3507   B821  BD 01 A3           LB821    JSR   RVEC23                   HOOK INTO RAM
3508   B824  9E A6                     LDX   CHARAD                    GET BASIC'S INPUT POINTER ADDRESS
3509   B826  CE 02 DC                  LDU   #LINBUF                   POINT X TO LINE INPUT BUFFER
3510   B829  0F 43              LB829  CLR   V43                       CLEAR ILLEGAL TOKEN FLAG
3511   B82B  0F 44                     CLR   V44                       CLEAR DATA FLAG
3512   B82D  A6 80              LB82D  LDA   ,X+                       GET INPUT CHAR
3513   B82F  27 21                     BEQ   LB852                     BRANCH IF END OF LINE
3514   B831  0D 43                     TST   V43                       * CHECK ILLEGAL TOKEN FLAG & BRANCH IF NOT
3515   B833  27 0F                     BEQ   LB844                     * PROCESSING AN ILLEGAL TOKEN
3516   B835  BD B3 A2                  JSR   LB3A2                     SET CARRY IF NOT UPPER CASE ALPHA
3517   B838  24 18                     BCC   LB852                     BRANCH IF UPPER CASE ALPHA
3518   B83A  81 30                     CMPA  #'0                       * DON T CRUNCH ASCII NUMERIC CHARACTERS
3519   B83C  25 04                     BLO   LB842                     * BRANCH IF NOT NUMERIC
3520   B83E  81 39                     CMPA  #'9                       *
3521   B840  23 10                     BLS   LB852                     * BRANCH IF NUMERIC
3522                           * END UP HERE IF NOT UPPER CASE ALPHA OR NUMERIC
3523   B842  0F 43              LB842  CLR   V43                       CLEAR ILLEGAL TOKEN FLAG
3524   B844  81 20              LB844  CMPA  #SPACE                    SPACE?
3525   B846  27 0A                     BEQ   LB852                     DO NOT REMOVE SPACES
3526   B848  97 42                     STA   V42                       SAVE INPUT CHARACTER AS SCAN DELIMITER
3527   B84A  81 22                     CMPA  #'"                       CHECK FOR STRING DELIMITER
3528   B84C  27 38                     BEQ   LB886                     BRANCH IF STRING
3529   B84E  0D 44                     TST   V44                       * CHECK DATA FLAG AND BRANCH IF CLEAR
3530   B850  27 19                     BEQ   LB86B                     * DO NOT CRUNCH DATA
3531   B852  A7 C0              LB852  STA   ,U+                       SAVE CHARACTER IN BUFFER
3532   B854  27 06                     BEQ   LB85C                     BRANCH IF END OF LINE
3533   B856  81 3A                     CMPA  #':                       * CHECK FOR END OF SUBLINE
3534   B858  27 CF                     BEQ   LB829                     * AND RESET FLAGS IF END OF SUBLINE
3535   B85A  20 D1              LB85A  BRA   LB82D                     GO GET ANOTHER CHARACTER
3536   B85C  6F C0              LB85C  CLR   ,U+                       * DOUBLE ZERO AT END OF LINE
3537   B85E  6F C0                     CLR   ,U+                       *
3538   B860  1F 30                     TFR   U,D                       SAVE ADDRESS OF END OF LINE IN ACCD
3539   B862  83 02 DA                  SUBD  #LINHDR                   LENGTH OF LINE IN ACCD
3540   B865  8E 02 DB                  LDX   #LINBUF-1                 * SET THE INPUT POINTER TO ONE BEFORE
3541   B868  9F A6                     STX   CHARAD                    * THE START OF THE CRUNCHED LINE
3542   B86A  39                        RTS                             EXIT 'CRUNCH'
3543   B86B  81 3F              LB86B  CMPA  #'?                       CHECK FOR "?" - PRINT ABBREVIATION
3544   B86D  26 04                     BNE   LB873                     BRANCH IF NOT PRINT ABBREVIATION
3545   B86F  86 87                     LDA   #$87                      * GET THE PRINT TOKEN AND SAVE IT
3546   B871  20 DF                     BRA   LB852                     * IN BUFFER
3547   B873  81 27              LB873  CMPA  #''                       APOSTROPHE IS SAME AS REM
3548   B875  26 13                     BNE   LB88A                     BRANCH IF NOT REMARK
3549   B877  CC 3A 83                  LDD   #$3A83                    COLON, REM TOKEN
3550   B87A  ED C1                     STD   ,U++                      SAVE IN BUFFER
3551   B87C  0F 42              LB87C  CLR   V42                       SET DELIMITER = 0 (END OF LINE)
3552   B87E  A6 80              LB87E  LDA   ,X+                       SCAN TILL WE MATCH [V42]
3553   B880  27 D0                     BEQ   LB852                     BRANCH IF END OF LINE
3554   B882  91 42                     CMPA  V42                       DELIMITER?
3555   B884  27 CC                     BEQ   LB852                     BRANCH OUT IF SO
3556   B886  A7 C0              LB886  STA   ,U+                       DON T CRUNCH REMARKS OR STRINGS
3557   B888  20 F4                     BRA   LB87E                     GO GET MORE STRING OR REMARK
3558   B88A  81 30              LB88A  CMPA  #'0                       * LESS THAN ASCII ZERO?
3559   B88C  25 04                     BCS   LB892                     * BRANCH IF SO
3560   B88E  81 3C                     CMPA  #';+1                     = CHECK FOR NUMERIC VALUE, COLON OR SEMICOLON
```

```
3561   B89Ø  25 CØ                    BCS   LB852                  = AND INSERT IN BUFFER IF SO
3562   B892  3Ø 1F             LB892  LEAX  -1,X                   MOVE INPUT POINTER BACK ONE
3563   B894  34 5Ø                    PSHS  U,X                     SAVE POINTERS TO INPUT STRING, OUTPUT STRING
3564   B896  ØF 41                    CLR   V41                     TOKEN FLAG Ø = COMMAND, FF = SECONDARY
3565   B898  CE Ø1 16                 LDU   #COMVEC-1Ø              POINT U TO COMMAND INTERPRETATION
3566                           *                                     TABLE FOR BASIC - 1Ø
3567   B89B  ØF 42             LB89B  CLR   V42                     INITIALIZE V42 AS TOKEN COUNTER
3568   B89D  33 4A             LB89D  LEAU  1Ø,U                    MOVE TO NEXT COMMAND INTERPRETATION TABLE
3569   B89F  A6 C4                    LDA   ,U                      GET NUMBER OF COMMANDS
3570   B8A1  27 31                    BEQ   LB8D4                   GO DO SECONDARY FUNCTIONS IF NO COMMAND TABLE
3571   B8A3  1Ø AE 41                 LDY   1,U                     POINT Y TO COMMAND DICTIONARY TABLE
3572   B8A6  AE E4             LB8A6  LDX   ,S                      GET POINTER TO INPUT STRING
3573   B8A8  E6 A0             LB8A8  LDB   ,Y+                     GET A BYTE FROM DICTIONARY TABLE
3574   B8AA  E0 80                    SUBB  ,X+                     SUBTRACT INPUT CHARACTER
3575   B8AC  27 FA                    BEQ   LB8A8                   LOOP IF SAME
3576   B8AE  C1 80                    CMPB  #$8Ø                    LAST CHAR IN RESERVED WORD TABLE HAD
3577                           *                                     BIT 7 SET, SO IF WE HAVE $8Ø HERE
3578                           *                                     THEN IT IS A GOOD COMPARE
3579   B8B0  26 38                    BNE   LB8EA                   BRANCH IF NO MATCH - CHECK ANOTHER COMMAND
3580   B8B2  32 62                    LEAS  2,S                     DELETE OLD INPUT POINTER FROM STACK
3581   B8B4  35 4Ø                    PULS  U                       GET POINTER TO OUTPUT STRING
3582   B8B6  DA 42                    ORB   V42                     OR IN THE TABLE POSITION TO MAKE THE TOKEN
3583                           *                                     - NOTE THAT B ALREADY HAD $8Ø IN IT -
3584   B8B8  96 41                    LDA   V41                     * CHECK TOKEN FLAG AND BRANCH
3585   B8BA  26 Ø6                    BNE   LB8C2                   * IF SECONDARY
3586   B8BC  C1 84                    CMPB  #$84                    IS IT ELSE TOKEN?
3587   B8BE  26 Ø6                    BNE   LB8C6                   NO
3588   B8CØ  86 3A                    LDA   #':                     PUT A COLON (SUBLINE) BEFORE ELSE TOKEN
3589   B8C2  ED C1             LB8C2  STD   ,U++                    SECONDARY TOKENS PRECEEDED BY $FF
3590   B8C4  2Ø 94                    BRA   LB85A                   GO PROCESS MORE INPUT CHARACTERS
3591   B8C6  E7 CØ             LB8C6  STB   ,U+                     SAVE THIS TOKEN
3592   B8C8  C1 86                    CMPB  #$86                    DATA TOKEN?
3593   B8CA  26 Ø2                    BNE   LB8CE                   NO
3594   B8CC  ØC 44                    INC   V44                     SET DATA FLAG
3595   B8CE  C1 82             LB8CE  CMPB  #$82                    REM TOKEN?
3596   B8DØ  27 AA                    BEQ   LB87C                   YES
3597   B8D2  2Ø 86             LB8D2  BRA   LB85A                   GO PROCESS MORE INPUT CHARACTERS
3598                           * CHECK FOR A SECONDARY TOKEN
3599   B8D4  CE Ø1 1B          LB8D4  LDU   #COMVEC-5               NOW DO SECONDARY FUNCTIONS
3600   B8D7  Ø3 41                    COM   V41                     TOGGLE THE TOKEN FLAG
3601   B8D9  26 CØ                    BNE   LB89B                   BRANCH IF NOW CHECKING SECONDARY COMMANDS
3602
3603                           * THIS CODE WILL PROCESS INPUT DATA WHICH CANNOT BE CRUNCHED AND SO
3604                           * IS ASSUMED TO BE ILLEGAL DATA OR AN ILLEGAL TOKEN
3605   B8DB  35 5Ø                    PULS  X,U                     RESTORE INPUT AND OUTPUT POINTERS
3606   B8DD  A6 80                    LDA   ,X+                     * MOVE THE FIRST CHARACTER OF AN
3607   B8DF  A7 CØ                    STA   ,U+                     * ILLEGAL TOKEN
3608   B8E1  BD B3 A2                 JSR   LB3A2                   SET CARRY IF NOT ALPHA
3609   B8E4  25 EC                    BCS   LB8D2                   BRANCH IF NOT ALPHA
3610   B8E6  Ø3 43                    COM   V43                     SET ILLEGAL TOKEN FLAG IF UPPER CASE ALPHA
3611   B8E8  2Ø E8                    BRA   LB8D2                   PROCESS MORE INPUT CHARACTERS
3612   B8EA  ØC 42             LB8EA  INC   V42                     INCREMENT TOKEN COUNTER
3613   B8EC  4A                       DECA                         DECR COMMAND COUNTER
3614   B8ED  27 AE                    BEQ   LB89D                   GET ANOTHER COMMAND TABLE IF DONE W/THIS ONE
3615   B8EF  31 3F                    LEAY  -1,Y                    MOVE POINTER BACK ONE
3616   B8F1  E6 A0             LB8F1  LDB   ,Y+                     * GET TO NEXT
3617   B8F3  2A FC                    BPL   LB8F1                   * RESERVED WORD
3618   B8F5  2Ø AF                    BRA   LB8A6                   GO SEE IF THIS WORD IS A MATCH
3619
3620                           * PRINT
3621   B8F7  27 5F             PRINT  BEQ   LB958                   BRANCH IF NO ARGUMENT
3622   B8F9  8D Ø3                    BSR   LB8FE                   CHECK FOR ALL PRINT OPTIONS
3623   B8FB  ØF 6F                    CLR   DEVNUM                  SET DEVICE NUMBER TO SCREEN
3624   B8FD  39                       RTS
3625   B8FE  81 4Ø             LB8FE  CMPA  #'@                     CHECK FOR PRINT @
3626   B9ØØ  26 Ø5                    BNE   LB9Ø7                   NOT PRINT @
3627   B9Ø2  BD A5 54                 JSR   LA554                   MOVE CURSOR TO PROPER PRINT LOCATION
3628   B9Ø5  2Ø ØA                    BRA   LB911                   GO PRINT THE DATA
3629   B9Ø7  81 23             LB9Ø7  CMPA  #'#                     CHECK FOR PRINT NUMBER
3630   B9Ø9  26 ØD                    BNE   LB918                   NOT PRINT#
3631   B9ØB  BD A5 A5                 JSR   LA5A5                   CHECK FOR A VALID DEVICE NUMBER
3632   B9ØE  BD A4 Ø6                 JSR   LA4Ø6                   CHECK FOR A VALID OUTPUT FILE
3633   B911  9D A5             LB911  JSR   GETCCH                  GET CURRENT INPUT CHARACTER
3634   B913  27 43                    BEQ   LB958                   BRANCH IF END OF LINE
3635   B915  BD B2 6D                 JSR   LB26D                   SYNTAX CHECK FOR COMMA
3636   B918  BD Ø1 79          LB918  JSR   RVEC9                   HOOK INTO RAM
3637   B91B  27 48             LB91B  BEQ   LB965                   RETURN IF END OF LINE
3638   B91D  81 A4             LB91D  CMPA  #$A4                    TOKEN FOR TAB( ?
3639   B91F  27 5D                    BEQ   LB97E                   YES
3640   B921  81 2C                    CMPA  #',                     COMMA?
3641   B923  27 41                    BEQ   LB966                   YES - ADVANCE TO NEXT TAB FIELD
3642   B925  81 3B                    CMPA  #';                     SEMICOLON?
3643   B927  27 6E                    BEQ   LB997                   YES - DO NOT ADVANCE CURSOR
3644   B929  BD B1 56                 JSR   LB156                   EVALUATE EXPRESSION
3645   B92C  96 Ø6                    LDA   VALTYP                  * GET VARIABLE TYPE AND
3646   B92E  34 Ø2                    PSHS  A                       * SAVE IT ON THE STACK
3647   B93Ø  26 Ø6                    BNE   LB938                   BRANCH IF STRING VARIABLE
3648   B932  BD BD D9                 JSR   LBDD9                   CONVERT FP NUMBER TO AN ASCII STRING
3649   B935  BD B5 16                 JSR   LB516                   PARSE A STRING FROM (X-1) AND PUT
```

```
3650                            *                                       DESCRIPTOR ON STRING STACK
3651   B938  8D 65              LB938     BSR   LB99F                   PRINT STRING POINTED TO BY X
3652   B93A  35 04                       PULS  B                       GET VARIABLE TYPE BACK
3653   B93C  BD A3 5F                    JSR   LA35F                   SET UP TAB WIDTH ZONE, ETC
3654   B93F  0D 6E                       TST   PRTDEV                  * CHECK THE PRINT DEVICE
3655   B941  27 06                       BEQ   LB949                   * AND BRANCH IF NOT CASSETTE
3656   B943  8D 13                       BSR   LB958                   SEND A CARRIAGE RETURN TO CONSOLE OUT
3657   B945  9D A5                       JSR   GETCCH                  GET CURRENT INPUT CHARACTER
3658   B947  20 D2                       BRA   LB91B                   CHECK FOR MORE PRINT DATA
3659   B949  5D                 LB949     TSTB                          CHECK CURRENT PRINT POSITION
3660   B94A  26 08                       BNE   LB954                   BRANCH IF NOT AT START OF LINE
3661   B94C  9D A5                       JSR   GETCCH                  GET CURRENT INPUT CHARACTER
3662   B94E  81 2C                       CMPA  #',                     COMMA?
3663   B950  27 14                       BEQ   LB966                   SKIP TO NEXT TAB FIELD
3664   B952  8D 58                       BSR   LB9AC                   SEND A SPACE TO CONSOLE OUT
3665   B954  9D A5               LB954     JSR   GETCCH                GET CURRENT INPUT CHARACTER
3666   B956  26 C5                       BNE   LB91D                   BRANCH IF NOT END OF LINE
3667   B958  86 0D               LB958     LDA   #CR                     * SEND A CR TO
3668   B95A  20 55                       BRA   LB9B1                   * CONSOLE OUT
3669   B95C  BD A3 5F            LB95C     JSR   LA35F                   SET UP TAB WIDTH, ZONE ETC
3670   B95F  27 F7                       BEQ   LB958                   BRANCH IF WIDTH = ZERO
3671   B961  96 6C                       LDA   DEVPOS                  GET PRINT POSITION
3672   B963  26 F3                       BNE   LB958                   BRANCH IF NOT AT START OF LINE
3673   B965  39                  LB965     RTS
3674                            * SKIP TO NEXT TAB FIELD
3675   B966  BD A3 5F            LB966     JSR   LA35F                   SET UP TAB WIDTH, ZONE ETC
3676   B969  27 0A                       BEQ   LB975                   BRANCH IF LINE WIDTH = 0 (CASSETTE)
3677   B96B  D6 6C                       LDB   DEVPOS                  GET CURRENT POSITION
3678   B96D  D1 6B                       CMPB  DEVLCF                  COMPARE TO LAST TAB ZONE
3679   B96F  25 06                       BCS   LB977                   BRANCH IF < LAST TAB ZONE
3680   B971  8D E5                       BSR   LB958                   SEND A CARRIAGE RETURN TO CONSOLE OUT
3681   B973  20 22                       BRA   LB997                   GET MORE DATA
3682   B975  D6 6C               LB975     LDB   DEVPOS                 *
3683   B977  D0 6A               LB977     SUBB  DEVCFW                  * SUBTRACT TAB FIELD WIDTH FROM CURRENT
3684   B979  24 FC                       BCC   LB977                   * POSITION UNTIL CARRY SET - NEGATING THE
3685   B97B  50                          NEGB                          * REMAINDER LEAVES THE NUMBER OF SPACES TO NEXT
3686                            *                                       * TAB ZONE IN ACCB
3687   B97C  20 10                       BRA   LB98E                   GO ADVANCE TO NEXT TAB ZONE
3688
3689                            * PRINT TAB(
3690   B97E  BD B7 09            LB97E     JSR   LB709                   EVALUATE EXPRESSION - RETURN VALUE IN B
3691   B981  81 29                       CMPA  #')                     * 'SYNTAX' ERROR IF NOT ')'
3692   B983  10 26 F8 F0                 LBNE  LB277                   *
3693   B987  BD A3 5F                    JSR   LA35F                   SET UP TAB WIDTH, ZONE ETC
3694   B98A  D0 6C                       SUBB  DEVPOS                  GET DIFFERENCE OF PRINT POSITION & TAB POSITION
3695   B98C  23 09                       BLS   LB997                   BRANCH IF TAB POSITION < CURRENT POSITION
3696   B98E  0D 6E               LB98E     TST   PRTDEV                * GET PRINT DEVICE NUMBER AND
3697   B990  26 05                       BNE   LB997                   * BRANCH IF CASSETTE
3698   B992  8D 18               LB992     BSR   LB9AC                   SEND A SPACE TO CONSOLE OUT
3699   B994  5A                         DECB                          DECREMENT DIFFERENCE COUNT
3700   B995  26 FB                       BNE   LB992                   BRANCH UNTIL CURRENT POSITION = TAB POSITION
3701   B997  9D 9F               LB997     JSR   GETNCH                GET NEXT CHARACTER FROM BASIC
3702   B999  7E B9 1B                    JMP   LB91B                   LOOK FOR MORE PRINT DATA
3703                            * COPY A STRING FROM (X) TO CONSOLE OUT
3704   B99C  BD B5 18            LB99C     JSR   LB518                   PARSE A STRING FROM X AND PUT
3705                            *                                       DESCRIPTOR ON STRING STACK
3706   B99F  BD B6 57            LB99F     JSR   LB657                   GET LENGTH OF STRING AND REMOVE
3707                            *                                       DESCRIPTOR FROM STRING STACK
3708   B9A2  5C                          INCB                          COMPENSATE FOR DECB BELOW
3709   B9A3  5A                  LB9A3     DECB                          DECREMENT COUNTER
3710   B9A4  27 BF                       BEQ   LB965                   EXIT ROUTINE
3711   B9A6  A6 80                       LDA   ,X+                     GET A CHARACTER FROM X
3712   B9A8  8D 07                       BSR   LB9B1                   SEND TO CONSOLE OUT
3713   B9AA  20 F7                       BRA   LB9A3                   KEEP LOOPING
3714   B9AC  86 20               LB9AC     LDA   #SPACE                SPACE TO CONSOLE OUT
3715   B9AE  8C                          FCB   SKP2                    SKIP NEXT TWO BYTES
3716   B9AF  86 3F               LB9AF     LDA   #'?                   QUESTION MARK TO CONSOLE OUT
3717   B9B1  7E A2 82            LB9B1     JMP   PUTCHR                  JUMP TO CONSOLE OUT
3718
3719                            * FLOATING POINT MATH PACKAGE
3720
3721                            * ADD .5 TO FPA0
3722   B9B4  8E BE C0            LB9B4     LDX   #LBEC0                  FLOATING POINT CONSTANT (.5)
3723   B9B7  20 09                       BRA   LB9C2                   ADD .5 TO FPA0
3724                            * SUBTRACT FPA0 FROM FP NUMBER POINTED
3725                            * TO BY (X), LEAVE RESULT IN FPA0
3726   B9B9  BD BB 2F            LB9B9     JSR   LBB2F                   COPY PACKED FP DATA FROM (X) TO FPA1
3727
3728                            * ARITHMETIC OPERATION (-) JUMPS HERE - SUBTRACT FPA0 FROM FPA1 (ENTER
3729                            * WITH EXPONENT OF FPA0 IN ACCB AND EXPONENT OF FPA1 IN ACCA)
3730   B9BC  03 54               LB9BC     COM   FP0SGN                  CHANGE MANTISSA SIGN OF FPA0
3731   B9BE  03 62                       COM   RESSGN                  REVERSE RESULT SIGN FLAG
3732   B9C0  20 03                       BRA   LB9C5                   GO ADD FPA1 AND FPA0
3733                            * ADD FP NUMBER POINTED TO BY
3734                            * (X) TO FPA0 - LEAVE RESULT IN FPA0
3735   B9C2  BD BB 2F            LB9C2     JSR   LBB2F                   UNPACK PACKED FP DATA FROM (X) TO
3736                            *                                       FPA1; RETURN EXPONENT OF FPA1 IN ACCA
3737
3738                            * ARITHMETIC OPERATION (+) JUMPS HERE - ADD FPA0 TO
```

```
3739                            * FPA1 (ENTER WITH EXPONENT OF FPA0 IN ACCB AND EXPONENT OF FPA1 IN ACCA
3740   B9C5  5D                 LB9C5     TSTB                          CHECK EXPONENT OF FPA0
3741   B9C6  10 27 02 80                  LBEQ    LBC4A                 COPY FPA1 TO FPA0 IF FPA0 =  0
3742   B9CA  8E 00 5C                     LDX     #FP1EXP               POINT X TO FPA1
3743   B9CD  1F 89              LB9CD     TFR     A,B                   PUT EXPONENT OF FPA1 INTO ACCB
3744   B9CF  5D                           TSTB                          CHECK EXPONENT
3745   B9D0  27 6C                        BEQ     LBA3E                 RETURN IF EXPONENT = 0 (ADDING 0 TO FPA0)
3746   B9D2  D0 4F                        SUBB    FP0EXP                SUBTRACT EXPONENT OF FPA0 FROM EXPONENT OF FPA1
3747   B9D4  27 69                        BEQ     LBA3F                 BRANCH IF EXPONENTS ARE EQUAL
3748   B9D6  25 0A                        BCS     LB9E2                 BRANCH IF EXPONENT FPA0 > FPA1
3749   B9D8  97 4F                        STA     FP0EXP                REPLACE FPA0 EXPONENT WITH FPA1 EXPONENT
3750   B9DA  96 61                        LDA     FP1SGN                * REPLACE FPA0 MANTISSA SIGN
3751   B9DC  97 54                        STA     FP0SGN                * WITH FPA1 MANTISSA SIGN
3752   B9DE  8E 00 4F                     LDX     #FP0EXP               POINT X TO FPA0
3753   B9E1  50                           NEGB                          NEGATE DIFFERENCE OF EXPONENTS
3754   B9E2  C1 F8              LB9E2     CMPB    #-8                   TEST DIFFERENCE OF EXPONENTS
3755   B9E4  2F 59                        BLE     LBA3F                 BRANCH IF DIFFERENCE OF EXPONENTS <= 8
3756   B9E6  4F                           CLRA                          CLEAR OVERFLOW BYTE
3757   B9E7  64 01                        LSR     1,X                   SHIFT MS BYTE OF MANTISSA; BIT 7 = 0
3758   B9E9  BD BA BA                     JSR     LBABA                 GO SHIFT MANTISSA OF (X) TO THE RIGHT (B) TIMES
3759   B9EC  D6 62              LB9EC     LDB     RESSGN                GET SIGN FLAG
3760   B9EE  2A 0B                        BPL     LB9FB                 BRANCH IF FPA0 AND FPA1 SIGNS ARE THE SAME
3761   B9F0  63 01                        COM     1,X                   * COMPLEMENT MANTISSA POINTED
3762   B9F2  63 02                        COM     2,X                   * TO BY (X) THE
3763   B9F4  63 03                        COM     3,X                   * ADCA BELOW WILL
3764   B9F6  63 04                        COM     4,X                   * CONVERT THIS OPERATION
3765   B9F8  43                           COMA                          * INTO A NEG (MANTISSA)
3766   B9F9  89 00                        ADCA    #0                    ADD ONE TO ACCA - COMA ALWAYS SETS THE CARRY FLAG
3767                           * THE PREVIOUS TWO BYTES MAY BE REPLACED BY A NEGA
3768                           *
3769                           * ADD MANTISSAS OF FPA0 AND FPA1, PUT RESULT IN FPA0
3770   B9FB  97 63              LB9FB     STA     FPSBYT                SAVE FPA SUB BYTE
3771   B9FD  96 53                        LDA     FPA0+3                * ADD LS BYTE
3772   B9FF  99 60                        ADCA    FPA1+3                * OF MANTISSA
3773   BA01  97 53                        STA     FPA0+3                SAVE IN FPA0 LSB
3774   BA03  96 52                        LDA     FPA0+2                * ADD NEXT BYTE
3775   BA05  99 5F                        ADCA    FPA1+2                * OF MANTISSA
3776   BA07  97 52                        STA     FPA0+2                SAVE IN FPA0
3777   BA09  96 51                        LDA     FPA0+1                * ADD NEXT BYTE
3778   BA0B  99 5E                        ADCA    FPA1+1                * OF MANTISSA
3779   BA0D  97 51                        STA     FPA0+1                SAVE IN FPA0
3780   BA0F  96 50                        LDA     FPA0                  * ADD MS BYTE
3781   BA11  99 5D                        ADCA    FPA1                  * OF MANTISSA
3782   BA13  97 50                        STA     FPA0                  SAVE IN FPA0
3783   BA15  5D                           TSTB                          TEST SIGN FLAG
3784   BA16  2A 44                        BPL     LBA5C                 BRANCH IF FPA0 & FPA1 SIGNS WERE ALIKE
3785   BA18  25 02              LBA18     BCS     LBA1C                 BRANCH IF POSITIVE MANTISSA
3786   BA1A  8D 5D                        BSR     LBA79                 NEGATE FPA0 MANTISSA
3787
3788                           * NORMALIZE FPA0
3789   BA1C  5F                 LBA1C     CLRB                          CLEAR TEMPORARY EXPONENT ACCUMULATOR
3790   BA1D  96 50              LBA1D     LDA     FPA0                  TEST MSB OF MANTISSA
3791   BA1F  26 2E                        BNE     LBA4F                 BRANCH IF <> 0
3792   BA21  96 51                        LDA     FPA0+1                * IF THE MSB IS
3793   BA23  97 50                        STA     FPA0                  * 0, THEN SHIFT THE
3794   BA25  96 52                        LDA     FPA0+2                * MANTISSA A WHOLE BYTE
3795   BA27  97 51                        STA     FPA0+1                * AT A TIME. THIS
3796   BA29  96 53                        LDA     FPA0+3                * IS FASTER THAN ONE
3797   BA2B  97 52                        STA     FPA0+2                * BIT AT A TIME
3798   BA2D  96 63                        LDA     FPSBYT                * BUT USES MORE MEMORY.
3799   BA2F  97 53                        STA     FPA0+3                * FPSBYT, THE CARRY IN
3800   BA31  0F 63                        CLR     FPSBYT                * BYTE, REPLACES THE MATISSA LSB.
3801   BA33  CB 08                        ADDB    #8                    SHIFTING ONE BYTE = 8 BIT SHIFTS; ADD 8 TO EXPONENT
3802   BA35  C1 28                        CMPB    #5*8                  CHECK FOR 5 SHIFTS
3803   BA37  2D E4                        BLT     LBA1D                 BRANCH IF < 5 SHIFTS, IF > 5, THEN MANTISSA = 0
3804   BA39  4F                 LBA39     CLRA                          A ZERO EXPONENT = 0 FLOATING POINT
3805   BA3A  97 4F              LBA3A     STA     FP0EXP                ZERO OUT THE EXPONENT
3806   BA3C  97 54                        STA     FP0SGN                ZERO OUT THE MANTISSA SIGN
3807   BA3E  39                 LBA3E     RTS
3808   BA3F  8D 6D              LBA3F     BSR     LBAAE                 SHIFT FPA0 MANTISSA TO RIGHT
3809   BA41  5F                           CLRB                          CLEAR CARRY FLAG
3810   BA42  20 A8                        BRA     LB9EC
3811                           * SHIFT FPA0 LEFT ONE BIT UNTIL BIT 7
3812                           * OF MATISSA MS BYTE = 1
3813   BA44  5C                 LBA44     INCB                          ADD ONE TO EXPONENT ACCUMULATOR
3814   BA45  08 63                        ASL     FPSBYT                SHIFT SUB BYTE ONE LEFT
3815   BA47  09 53                        ROL     FPA0+3                SHIFT LS BYTE
3816   BA49  09 52                        ROL     FPA0+2                SHIFT NS BYTE
3817   BA4B  09 51                        ROL     FPA0+1                SHIFT NS BYTE
3818   BA4D  09 50                        ROL     FPA0                  SHIFT MS BYTE
3819   BA4F  2A F3              LBA4F     BPL     LBA44                 BRANCH IF NOT YET NORMALIZED
3820   BA51  96 4F                        LDA     FP0EXP                GET CURRENT EXPONENT
3821   BA53  34 04                        PSHS    B                     SAVE EXPONENT MODIFIER CAUSED BY NORMALIZATION
3822   BA55  A0 E0                        SUBA    ,S+                   SUBTRACT ACCUMULATED EXPONENT MODIFIER
3823   BA57  97 4F                        STA     FP0EXP                SAVE AS NEW EXPONENT
3824   BA59  23 DE                        BLS     LBA39                 SET FPA0 = 0 IF THE NORMALIZATION CAUSED
3825                           *                                       MORE OR EQUAL NUMBER OF LEFT SHIFTS THAN THE
3826                           *                                       SIZE OF THE EXPONENT
3827   BA5B  8C                           FCB     SKP2                  SKIP 2 BYTES
```

```
3828   BA5C  25 08           BA5C     BCS   LBA66                  BRANCH IF MANTISSA OVERFLOW
3829   BA5E  08 63                    ASL   FPSBYT                 SUB BYTE BIT 7 TO CARRY - USE AS ROUND-OFF
3830                         *                                      FLAG (TRUNCATE THE REST OF SUB BYTE)
3831   BA60  86 00                    LDA   #0                     CLRA, BUT DO NOT CHANGE CARRY FLAG
3832   BA62  97 63                    STA   FPSBYT                 CLEAR THE SUB BYTE
3833   BA64  20 0C                    BRA   LBA72                  GO ROUND-OFF RESULT
3834   BA66  0C 4F           LBA66    INC   FP0EXP                 INCREMENT EXPONENT - MULTIPLY BY 2
3835   BA68  27 28                    BEQ   LBA92                  OVERFLOW ERROR IF CARRY PAST $FF
3836   BA6A  06 50                    ROR   FPA0                   * SHIFT MANTISSA
3837   BA6C  06 51                    ROR   FPA0+1                 * ONE TO
3838   BA6E  06 52                    ROR   FPA0+2                 * THE RIGHT -
3839   BA70  06 53                    ROR   FPA0+3                 * DIVIDE BY TWO
3840   BA72  24 04           LBA72    BCC   LBA78                  BRANCH IF NO ROUND-OFF NEEDED
3841   BA74  8D 0D                    BSR   LBA83                  ADD ONE TO MANTISSA - ROUND OFF
3842   BA76  27 EE                    BEQ   LBA66                  BRANCH iF OVERFLOW - MANTISSA = 0
3843   BA78  39              LBA78    RTS
3844                         * NEGATE FPA0 MANTISSA
3845   BA79  03 54           LBA79    COM   FP0SGN                 TOGGLE SIGN OF MANTISSA
3846   BA7B  03 50           LBA7B    COM   FPA0                   * COMPLEMENT ALL 4 MANTISSA BYTES
3847   BA7D  03 51                    COM   FPA0+1                 *
3848   BA7F  03 52                    COM   FPA0+2                 *
3849   BA81  03 53                    COM   FPA0+3                 *
3850                         * ADD ONE TO FPA0 MANTISSA
3851   BA83  9E 52           LBA83    LDX   FPA0+2                 * GET BOTTOM 2 MANTISSA
3852   BA85  30 01                    LEAX  1,X                    * BYTES, ADD ONE TO
3853   BA87  9F 52                    STX   FPA0+2                 * THEM AND SAVE THEM
3854   BA89  26 06                    BNE   LBA91                  BRANCH IF NO OVERFLOW
3855   BA8B  9E 50                    LDX   FPA0                   * IF OVERFLOW ADD ONE
3856   BA8D  30 01                    LEAX  1,X                    * TO TOP 2 MANTISSA
3857   BA8F  9F 50                    STX   FPA0                   * BYTES AND SAVE THEM
3858   BA91  39              LBA91    RTS
3859   BA92  C6 0A           LBA92    LDB   #2*5                   'OV' OVERFLOW ERROR
3860   BA94  7E AC 46                 JMP   LAC46                  PROCESS AN ERROR
3861   BA97  8E 00 12        LBA97    LDX   #FPA2-1                POINT X TO FPA2
3862                         * SHIFT FPA POINTED TO BY (X) TO
3863                         * THE RIGHT -(B) TIMES. EXIT WITH
3864                         * ACCA CONTAINING DATA SHIFTED OUT
3865                         * TO THE RIGHT (SUB BYTE) AND THE DATA
3866                         * SHIFTED IN FROM THE LEFT WILL COME FROM FPCARY
3867   BA9A  A6 04           LBA9A    LDA   4,X                    GET LS BYTE OF MANTISSA (X)
3868   BA9C  97 63                    STA   FPSBYT                 SAVE IN FPA SUB BYTE
3869   BA9E  A6 03                    LDA   3,X                    * SHIFT THE NEXT THREE BYTES OF THE
3870   BAA0  A7 04                    STA   4,X                    * MANTISSA RIGHT ONE COMPLETE BYTE.
3871   BAA2  A6 02                    LDA   2,X                    *
3872   BAA4  A7 03                    STA   3,X                    *
3873   BAA6  A6 01                    LDA   1,X                    *
3874   BAA8  A7 02                    STA   2,X                    *
3875   BAAA  96 5B                    LDA   FPCARY                 GET THE CARRY IN BYTE
3876   BAAC  A7 01                    STA   1,X                    STORE AS THE MS MANTISSA BYTE OF (X)
3877   BAAE  CB 08           LBAAE    ADDB  #8                     ADD 8 TO DIFFERENCE OF EXPONENTS
3878   BAB0  2F E8                    BLE   LBA9A                  BRANCH IF EXPONENT DIFFERENCE < -8
3879   BAB2  96 63                    LDA   FPSBYT                 GET FPA SUB BYTE
3880   BAB4  C0 08                    SUBB  #8                     CAST OUT THE 8 ADDED IN ABOVE
3881   BAB6  27 0C                    BEQ   LBAC4                  BRANCH IF EXPONENT DIFFERENCE = 0
3882                         * SHIFT MANTISSA POINTED TO BY (X) TO
3883                         $ THE RIGHT (B) TIMES. OVERFLOW RETAINED IN ACCA.
3884   BAB8  67 01           LBAB8    ASR   1,X                    * SHIFT MANTISSA AND SUB BYTE ONE BIT TO THE RIGHT
3885   BABA  66 02           LBABA    ROR   2,X                    *
3886   BABC  66 03                    ROR   3,X                    *
3887   BABE  66 04                    ROR   4,X                    *
3888   BAC0  46                       RORA                         *
3889   BAC1  5C                       INCB                         ADD ONE TO EXPONENT DIFFERENCE
3890   BAC2  26 F4                    BNE   LBAB8                  BRANCH IF EXPONENTS NOT =
3891   BAC4  39              LBAC4    RTS
3892   BAC5  81 00 00 00 00  LBAC5    FCB   $81,$00,$00,$00,$00    FLOATING POINT CONSTANT 1.0
3893
3894                         * ARITHMETIC OPERATION (*) JUMPS HERE - MULTIPLY
3895                         * FPA0 BY (X) - RETURN PRODUCT IN FPA0
3896   BACA  8D 63           LBACA    BSR   LBB2F                  MOVE PACKED FPA FROM (X) TO FPA1
3897   BACC  27 60                    BEQ   LBB2E                  BRANCH IF EXPONENT OF FPA0 = 0
3898   BACE  8D 78                    BSR   LBB48                  CALCULATE EXPONENT OF PRODUCT
3899                         * MULTIPLY FPA0 MANTISSA BY FPA1. NORMALIZE
3900                         * HIGH ORDER BYTES OF PRODUCT IN FPA0. THE
3901                         * LOW ORDER FOUR BYTES OF THE PRODUCT WILL
3902                         * BE STORED IN VAB-VAE.
3903   BAD0  86 00           LBAD0    LDA   #0                     * ZERO OUT MANTISSA OF FPA2
3904   BAD2  97 13                    STA   FPA2                   *
3905   BAD4  97 14                    STA   FPA2+1                 *
3906   BAD6  97 15                    STA   FPA2+2                 *
3907   BAD8  97 16                    STA   FPA2+3                 *
3908   BADA  D6 53                    LDB   FPA0+3                 GET LS BYTE OF FPA0
3909   BADC  8D 22                    BSR   LBB00                  MULTIPLY BY FPA1
3910   BADE  D6 63                    LDB   FPSBYT                 * TEMPORARILY SAVE SUB BYTE 4
3911   BAE0  D7 AE                    STB   VAE                    *
3912   BAE2  D6 52                    LDB   FPA0+2                 GET NUMBER 3 MANTISSA BYTE OF FPA0
3913   BAE4  8D 1A                    BSR   LBB00                  MULTIPLY BY FPA1
3914   BAE6  D6 63                    LDB   FPSBYT                 * TEMPORARILY SAVE SUB BYTE 3
3915   BAE8  D7 AD                    STB   VAD                    *
3916   BAEA  D6 51                    LDB   FPA0+1                 GET NUMBER 2 MANTISSA BYTE OF FPA0
```

```
3917   BAEC  8D 12              BSR   LBB00                 MULTIPLY BY FPA1
3918   BAEE  D6 63              LDB   FPSBYT                * TEMPORARILY SAVE SUB BYTE 2
3919   BAF0  D7 AC              STB   VAC                   *
3920   BAF2  D6 50              LDB   FPA0                  GET MS BYTE OF FPA0 MANTISSA
3921   BAF4  8D 0C              BSR   LBB02                 MULTIPLY BY FPA1
3922   BAF6  D6 63              LDB   FPSBYT                * TEMPORARILY SAVE SUB BYTE 1
3923   BAF8  D7 AB              STB   VAB                   *
3924   BAFA  BD BC 0B           JSR   LBC0B                 COPY MANTISSA FROM FPA2 TO FPA0
3925   BAFD  7E BA 1C           JMP   LBA1C                 NORMALIZE FPA0
3926   BB00  27 95        LBB00 BEQ   LBA97                 SHIFT FPA2 ONE BYTE TO RIGHT
3927   BB02  43           LBB02 COMA                        SET CARRY FLAG
3928                     * MULTIPLY FPA1 MANTISSA BY ACCB AND
3929                     * ADD PRODUCT TO FPA2 MANTISSA
3930   BB03  96 13        LBB03 LDA   FPA2                  GET FPA2 MS BYTE
3931   BB05  56                 RORB                        ROTATE CARRY FLAG INTO SHIFT COUNTER;
3932                     *                                   DATA BIT INTO CARRY
3933   BB06  27 26              BEQ   LBB2E                 BRANCH WHEN 8 SHIFTS DONE
3934   BB08  24 16              BCC   LBB20                 DO NOT ADD FPA1 IF DATA BIT = 0
3935   BB0A  96 16              LDA   FPA2+3                * ADD MANTISSA LS BYTE
3936   BB0C  9B 60              ADDA  FPA1+3                *
3937   BB0E  97 16              STA   FPA2+3                *
3938   BB10  96 15              LDA   FPA2+2                = ADD MANTISSA NUMBER 3 BYTE
3939   BB12  99 5F              ADCA  FPA1+2                =
3940   BB14  97 15              STA   FPA2+2                =
3941   BB16  96 14              LDA   FPA2+1                * ADD MANTISSA NUMBER 2 BYTE
3942   BB18  99 5E              ADCA  FPA1+1                *
3943   BB1A  97 14              STA   FPA2+1                *
3944   BB1C  96 13              LDA   FPA2                  = ADD MANTISSA MS BYTE
3945   BB1E  99 5D              ADCA  FPA1                  =
3946   BB20  46           LBB20 RORA                        * ROTATE CARRY INTO MS BYTE
3947   BB21  97 13              STA   FPA2                  *
3948   BB23  06 14              ROR   FPA2+1                = ROTATE FPA2 ONE BIT TO THE RIGHT
3949   BB25  06 15              ROR   FPA2+2                =
3950   BB27  06 16              ROR   FPA2+3                =
3951   BB29  06 63              ROR   FPSBYT                =
3952   BB2B  4F                 CLRA                        CLEAR CARRY FLAG
3953   BB2C  20 D5              BRA   LBB03                 KEEP LOOPING
3954   BB2E  39           LBB2E RTS
3955                     * UNPACK A FP NUMBER FROM (X) TO FPA1
3956   BB2F  EC 01        LBB2F LDD   1,X                   GET TWO MSB BYTES OF MANTISSA FROM
3957                                                        FPA POINTED TO BY X
3958   BB31  97 61              STA   FP1SGN                SAVE PACKED MANTISSA SIGN BYTE
3959   BB33  8A 80              ORA   #$80                  FORCE BIT 7 OF MSB MANTISSA = 1
3960   BB35  DD 5D              STD   FPA1                  SAVE 2 MSB BYTES IN FPA1
3961   BB37  D6 61              LDB   FP1SGN                * GET PACKED MANTISSA SIGN BYTE. EOR W/FPA0
3962   BB39  D8 54              EORB  FP0SGN                * SIGN - NEW SIGN POSITION IF BOTH OLD SIGNS ALIKE,
3963   BB3B  D7 62              STB   RESSGN                * NEG IF BOTH OLD SIGNS DIFF. SAVE ADJUSTED
3964                     *                                   * MANTISSA SIGN BYTE
3965   BB3D  EC 03              LDD   3,X                   = GET 2 LSB BYTES OF MANTISSA
3966   BB3F  DD 5F              STD   FPA1+2                = AND PUT IN FPA1
3967   BB41  A6 84              LDA   ,X                    * GET EXPONENT FROM (X) AND
3968   BB43  97 5C              STA   FP1EXP                * PUT IN EXPONENT OF FPA1
3969   BB45  D6 4F              LDB   FP0EXP                GET EXPONENT OF FPA0
3970   BB47  39                 RTS
3971                     * CALCULATE EXPONENT FOR PRODUCT OF FPA0 & FPA1
3972                     * ENTER WITH EXPONENT OF FPA1 IN ACCA
3973   BB48  4D           LBB48 TSTA                        TEST EXPONENT OF FPA1
3974   BB49  27 16              BEQ   LBB61                 PURGE RETURN ADDRESS & SET FPA0 = 0
3975   BB4B  9B 4F              ADDA  FP0EXP                ADD FPA1 EXPONENT TO FPA0 EXPONENT
3976   BB4D  46                 RORA                        ROTATE CARRY INTO BIT 7; BIT 0 INTO CARRY
3977   BB4E  49                 ROLA                        SET OVERFLOW FLAG
3978   BB4F  28 10              BVC   LBB61                 BRANCH IF EXPONENT TOO LARGE OR SMALL
3979   BB51  8B 80              ADDA  #$80                  ADD $80 BIAS TO EXPONENT
3980   BB53  97 4F              STA   FP0EXP                SAVE NEW EXPONENT
3981   BB55  27 0C              BEQ   LBB63                 SET FPA0
3982   BB57  96 62              LDA   RESSGN                GET MANTISSA SIGN
3983   BB59  97 54              STA   FP0SGN                SAVE AS MANTISSA SIGN OF FPA0
3984   BB5B  39                 RTS
3985                     * IF FPA0 = POSITIVE THEN 'OV' ERROR IF FPA0
3986                     * = IS NEGATIVE THEN FPA0 = 0
3987   BB5C  96 54        LBB5C LDA   FP0SGN                GET MANTISSA SIGN OF FPA0
3988   BB5E  43                 COMA                        CHANGE SIGN OF FPA0 MANTISSA
3989   BB5F  20 02              BRA   LBB63
3990   BB61  32 62        LBB61 LEAS  2,S                   PURGE RETURN ADDRESS FROM STACK
3991   BB63  10 2A FE D2  LBB63 LBPL  LBA39                 ZERO FPA0 MANTISSA SIGN & EXPONENT
3992   BB67  7E BA 92     LBB67 JMP   LBA92                 'OV' OVERFLOW ERROR
3993                     * FAST MULTIPLY BY 10 AND LEAVE RESULT IN FPA0
3994   BB6A  BD BC 5F     LBB6A JSR   LBC5F                 TRANSFER FPA0 TO FPA1
3995   BB6D  27 0D              BEQ   LBB7C                 BRANCH IF EXPONENT = 0
3996   BB6F  8B 02              ADDA  #2                    ADD 2 TO EXPONENT (TIMES 4)
3997   BB71  25 F4              BCS   LBB67                 'OV' ERROR IF EXPONENT > $FF
3998   BB73  0F 62              CLR   RESSGN                CLEAR RESULT SIGN BYTE
3999   BB75  BD B9 CD           JSR   LB9CD                 ADD FPA1 TO FPA0 (TIMES 5)
4000   BB78  0C 4F              INC   FP0EXP                ADD ONE TO EXPONENT (TIMES 10)
4001   BB7A  27 EB              BEQ   LBB67                 'OV' ERROR IF EXPONENT > $FF
4002   BB7C  39           LBB7C RTS
4003   BB7D  84 20 00 00 00 LBB7D FCB   $84,$20,$00,$00,$00  FLOATING POINT CONSTANT 10
4004                     * DIVIDE FPA0 BY 10
4005   BB82  BD BC 5F     LBB82 JSR   LBC5F                 MOVE FPA0 TO FPA1
```

```
4006   BB85  8E BB 7D                LDX   #LBB7D                  POINT TO FLOATING POINT CONSTANT 1Ø
4007   BB88  5F                      CLRB                          ZERO MANTISSA SIGN BYTE
4008   BB89  D7 62            LBB89  STB   RESSGN                   STORE THE QUOTIENT MANTISSA SIGN BYTE
4009   BB8B  BD BC 14                JSR   LBC14                   UNPACK AN FP NUMBER FROM (X) INTO FPAØ
4010   BB8E  8C                      FCB   SKP2                    SKIP TWO BYTES
4011                        * DIVIDE (X) BY FPAØ-LEAVE NORMALIZED QUOTIENT IN FPAØ
4012   BB8F  8D 9E            LBB8F  BSR   LBB2F                   GET FP NUMBER FROM (X) TO FPA1
4013
4014                        * ARITHMETIC OPERATION (/) JUMPS HERE. DIVIDE FPA1 BY FPAØ (ENTER WITH
4015                        * EXPONENT OF FPA1 IN ACCA AND FLAGS SET BY TSTA)
4016
4017                        * DIVIDE FPA1 BY FPAØ
4018   BB91  27 73            LBB91  BEQ   LBC06                   '/Ø' DIVIDE BY ZERO ERROR
4019   BB93  ØØ 4F                   NEG   FPØEXP                  GET EXPONENT OF RECIPROCAL OF DIVISOR
4020   BB95  8D B1                   BSR   LBB48                   CALCULATE EXPONENT OF QUOTIENT
4021   BB97  ØC 4F                   INC   FPØEXP                  INCREMENT EXPONENT
4022   BB99  27 CC                   BEQ   LBB67                   'OV' OVERFLOW ERROR
4023   BB9B  8E ØØ 13                LDX   #FPA2                   POINT X TO MANTISSA OF FPA2 - HOLD
4024                        *                                     TEMPORARY QUOTIENT IN FPA2
4025   BB9E  C6 Ø4                   LDB   #4                      5 BYTE DIVIDE
4026   BBAØ  D7 Ø3                   STB   TMPLOC                  SAVE BYTE COUNTER
4027   BBA2  C6 Ø1                   LDB   #1                      SHIFT COUNTER-AND TEMPORARY QUOTIENT BYTE
4028                        * COMPARE FPAØ MANTISSA TO FPA1 MANTISSA -
4029                        * SET CARRY FLAG IF FPA1 >= FPAØ
4030   BBA4  96 5Ø            LBBA4  LDA   FPAØ                    * COMPARE THE TWO MS BYTES
4031   BBA6  91 5D                   CMPA  FPA1                    * OF FPAØ AND FPA1 AND
4032   BBA8  26 13                   BNE   LBBBD                   * BRANCH IF <>
4033   BBAA  96 51                   LDA   FPAØ+1                  = COMPARE THE NUMBER 2
4034   BBAC  91 5E                   CMPA  FPA1+1                  = BYTES AND
4035   BBAE  26 ØD                   BNE   LBBBD                   = BRANCH IF <>
4036   BBBØ  96 52                   LDA   FPAØ+2                  * COMPARE THE NUMBER 3
4037   BBB2  91 5F                   CMPA  FPA1+2                  * BYTES AND
4038   BBB4  26 Ø7                   BNE   LBBBD                   * BRANCH IF <>
4039   BBB6  96 53                   LDA   FPAØ+3                  = COMPARE THE LS BYTES
4040   BBB8  91 6Ø                   CMPA  FPA1+3                  = AND BRANCH
4041   BBBA  26 Ø1                   BNE   LBBBD                   = IF <>
4042   BBBC  43                      COMA                          SET CARRY FLAG IF FPAØ = FPA1
4043   BBBD  1F A8            LBBBD  TFR   CC,A                    SAVE CARRY FLAG STATUS IN ACCA; CARRY
4044                        *                                     CLEAR IF FPAØ > FPA1
4045   BBBF  59                      ROLB                          ROTATE CARRY INTO TEMPORARY QUOTIENT BYTE
4046   BBCØ  24 ØA                   BCC   LBBCC                   CARRY WILL BE SET AFTER 8 SHIFTS
4047   BBC2  E7 8Ø                   STB   ,X+                     SAVE TEMPORARY QUOTIENT
4048   BBC4  ØA Ø3                   DEC   TMPLOC                  DECREMENT BYTE COUNTER
4049   BBC6  2B 34                   BMI   LBBFC                   BRANCH IF DONE
4050   BBC8  27 2E                   BEQ   LBBF8                   BRANCH IF LAST BYTE
4051   BBCA  C6 Ø1                   LDB   #1                      RESET SHIFT COUNTER AND TEMPORARY QUOTIENT BYTE
4052   BBCC  1F 8A            LBBCC  TFR   A,CC                    RESTORE CARRY FLAG AND
4053   BBCE  25 ØE                   BCS   LBBDE                   BRANCH IF FPAØ =< FPA1
4054   BBDØ  Ø8 6Ø            LBBDØ  ASL   FPA1+3                  * SHIFT FPA1 MANTISSA 1 BIT TO LEFT
4055   BBD2  Ø9 5F                   ROL   FPA1+2                  *
4056   BBD4  Ø9 5E                   ROL   FPA1+1                  *
4057   BBD6  Ø9 5D                   ROL   FPA1                    *
4058   BBD8  25 E3                   BCS   LBBBD                   BRANCH IF CARRY - ADD ONE TO PARTIAL QUOTIENT
4059   BBDA  2B C8                   BMI   LBBA4                   IF MSB OF HIGH ORDER MANTISSA BYTE IS
4060                        *                                     SET, CHECK THE MAGNITUDES OF FPAØ, FPA1
4061   BBDC  2Ø DF                   BRA   LBBBD                   CARRY IS CLEAR, CHECK ANOTHER BIT
4062                        * SUBTRACT FPAØ FROM FPA1 - LEAVE RESULT IN FPA1
4063   BBDE  96 6Ø            LBBDE  LDA   FPA1+3                  * SUBTRACT THE LS BYTES OF MANTISSA
4064   BBEØ  9Ø 53                   SUBA  FPAØ+3                  *
4065   BBE2  97 6Ø                   STA   FPA1+3                  *
4066   BBE4  96 5F                   LDA   FPA1+2                  = THEN THE NEXT BYTE
4067   BBE6  92 52                   SBCA  FPAØ+2                  =
4068   BBE8  97 5F                   STA   FPA1+2                  =
4069   BBEA  96 5E                   LDA   FPA1+1                  * AND THE NEXT
4070   BBEC  92 51                   SBCA  FPAØ+1                  *
4071   BBEE  97 5E                   STA   FPA1+1                  *
4072   BBFØ  96 5D                   LDA   FPA1                    = AND FINALLY, THE MS BYTE OF MANTISSA
4073   BBF2  92 5Ø                   SBCA  FPAØ                    =
4074   BBF4  97 5D                   STA   FPA1                    =
4075   BBF6  2Ø D8                   BRA   LBBDØ                   GO SHIFT FPA1
4076   BBF8  C6 4Ø            LBBF8  LDB   #$4Ø                    USE ONLY TWO BITS OF THE LAST BYTE (FIFTH)
4077   BBFA  2Ø DØ                   BRA   LBBCC                   GO SHIFT THE LAST BYTE
4078   BBFC  56               LBBFC  RORB                          * SHIFT CARRY (ALWAYS SET HERE) INTO
4079   BBFD  56                      RORB                          * BIT 5 AND MOVE
4080   BBFE  56                      RORB                          * BITS 1,Ø TO BITS 7,6
4081   BBFF  D7 63                   STB   FPSBYT                  SAVE SUB BYTE
4082   BCØ1  8D Ø8                   BSR   LBCØB                   MOVE MANTISSA OF FPA2 TO FPAØ
4083   BCØ3  7E BA 1C                JMP   LBA1C                   NORMALIZE FPAØ
4084   BCØ6  C6 14            LBC06  LDB   #2*1Ø                   '/Ø' ERROR
4085   BCØ8  7E AC 46                JMP   LAC46                   PROCESS THE ERROR
4086                        * COPY MANTISSA FROM FPA2 TO FPAØ
4087   BCØB  9E 13            LBCØB  LDX   FPA2                    * MOVE TOP 2 BYTES
4088   BCØD  9F 5Ø                   STX   FPAØ                    *
4089   BCØF  9E 15                   LDX   FPA2+2                  = MOVE BOTTOM 2 BYTES
4090   BC11  9F 52                   STX   FPAØ+2                  =
4091   BC13  39                      RTS
4092                        * COPY A PACKED FP NUMBER FROM (X) TO FPAØ
4093   BC14  34 Ø2            LBC14  PSHS  A                       SAVE ACCA
4094   BC16  EC Ø1                   LDD   1,X                     GET TOP TWO MANTISSA BYTES
```

```
4095   BC18  97 54                    STA   FPØSGN                    SAVE MS BYTE OF MANTISSA AS MANTISSA SIGN
4096   BC1A  8A 80                    ORA   #$80                      UNPACK MS BYTE
4097   BC1C  DD 50                    STD   FPA0                      SAVE UNPACKED TOP 2 MANTISSA BYTES
4098   BC1E  0F 63                    CLR   FPSBYT                    CLEAR MANTISSA SUB BYTE
4099   BC20  E6 84                    LDB   ,X                        GET EXPONENT TO ACCB
4100   BC22  AE 03                    LDX   3,X                       * MOVE LAST 2
4101   BC24  9F 52                    STX   FPA0+2                    * MANTISSA BYTES
4102   BC26  D7 4F                    STB   FPØEXP                    SAVE EXPONENT
4103   BC28  35 82                    PULS  A,PC                      RESTORE ACCA AND RETURN
4104
4105   BC2A  8E 00 45          LBC2A  LDX   #V45                      POINT X TO MANTISSA OF FPA4
4106   BC2D  20 06                    BRA   LBC35                     MOVE FPA0 TO FPA4
4107   BC2F  8E 00 40          LBC2F  LDX   #V40                      POINT X TO MANTISSA OF FPA3
4108   BC32  8C                       FCB   SKP2                      SKIP TWO BYTES
4109   BC33  9E 3B             LBC33  LDX   VARDES                    POINT X TO VARIABLE DESCRIPTOR IN VARDES
4110                         * PACK FPA0 AND MOVE IT TO ADDRESS IN X
4111   BC35  96 4F             LBC35  LDA   FPØEXP                    * COPY EXPONENT
4112   BC37  A7 84                    STA   ,X                        *
4113   BC39  96 54                    LDA   FPØSGN                    GET MANTISSA SIGN BIT
4114   BC3B  8A 7F                    ORA   #$7F                      MASK THE BOTTOM 7 BITS
4115   BC3D  94 50                    ANDA  FPA0                      AND BIT 7 OF MANTISSA SIGN INTO BIT 7 OF MS BYTE
4116   BC3F  A7 01                    STA   1,X                       SAVE MS BYTE
4117   BC41  96 51                    LDA   FPA0+1                    * MOVE 2ND MANTISSA BYTE
4118   BC43  A7 02                    STA   2,X                       *
4119   BC45  DE 52                    LDU   FPA0+2                    = MOVE BOTTOM 2 MANTISSA BYTES
4120   BC47  EF 03                    STU   3,X                       =
4121   BC49  39                       RTS
4122                         * MOVE FPA1 TO FPA0 RETURN W/MANTISSA SIGN IN ACCA
4123   BC4A  96 61             LBC4A  LDA   FP1SGN                    * COPY MANTISSA SIGN FROM
4124   BC4C  97 54             LBC4C  STA   FPØSGN                    * FPA1 TO FPA0
4125   BC4E  9E 5C                    LDX   FP1EXP                    = COPY EXPONENT + MS BYTE FROM
4126   BC50  9F 4F                    STX   FPØEXP                    = FPA1 TO FPA0
4127   BC52  0F 63                    CLR   FPSBYT                    CLEAR MANTISSA SUB BYTE
4128   BC54  96 5E                    LDA   FPA1+1                    * COPY 2ND MANTISSA BYTE
4129   BC56  97 51                    STA   FPA0+1                    * FROM FPA1 TO FPA0
4130   BC58  96 54                    LDA   FPØSGN                    GET MANTISSA SIGN
4131   BC5A  9E 5F                    LDX   FPA1+2                    * COPY 3RD AND 4TH MANTISSA BYTE
4132   BC5C  9F 52                    STX   FPA0+2                    * FROM FPA1 TO FPA0
4133   BC5E  39                       RTS
4134                         * TRANSFER FPA0 TO FPA1
4135   BC5F  DC 4F             LBC5F  LDD   FPØEXP                    * TRANSFER EXPONENT & MS BYTE
4136   BC61  DD 5C                    STD   FP1EXP                    *
4137   BC63  9E 51                    LDX   FPA0+1                    = TRANSFER MIDDLE TWO BYTES
4138   BC65  9F 5E                    STX   FPA1+1                    =
4139   BC67  9E 53                    LDX   FPA0+3                    * TRANSFER BOTTOM TWO BYTES
4140   BC69  9F 60                    STX   FPA1+3                    *
4141   BC6B  4D                       TSTA                            SET FLAGS ACCORDING TO EXPONENT
4142   BC6C  39                       RTS
4143                         * CHECK FPA0; RETURN ACCB = 0 IF FPA0 = 0,
4144                         * ACCB = $FF IF FPA0 = NEGATIVE, ACCB = 1 IF FPA0 = POSITIVE
4145   BC6D  D6 4F             LBC6D  LDB   FPØEXP                    GET EXPONENT
4146   BC6F  27 08                    BEQ   LBC79                     BRANCH IF FPA0 = 0
4147   BC71  D6 54             LBC71  LDB   FPØSGN                    GET SIGN OF MANTISSA
4148   BC73  59                LBC73  ROLB                            BIT 7 TO CARRY
4149   BC74  C6 FF                    LDB   #$FF                      NEGATIVE FLAG
4150   BC76  25 01                    BCS   LBC79                     BRANCH IF NEGATIVE MANTISSA
4151   BC78  50                       NEGB                            ACCB = 1 IF POSITIVE MANTISSA
4152   BC79  39                LBC79  RTS
4153
4154                         * SGN
4155   BC7A  8D F1             SGN    BSR   LBC6D                     SET ACCB ACCORDING TO SIGN OF FPA0
4156                         * CONVERT A SIGNED NUMBER IN ACCB INTO A FLOATING POINT NUMBER
4157   BC7C  D7 50             LBC7C  STB   FPA0                      SAVE ACCB IN FPA0
4158   BC7E  0F 51                    CLR   FPA0+1                    CLEAR NUMBER 2 MANTISSA BYTE OF FPA0
4159   BC80  C6 88                    LDB   #$88                      EXPONENT REQUIRED IF FPA0 IS TO BE AN INTEGER
4160   BC82  96 50             LBC82  LDA   FPA0                      GET MS BYTE OF MANTISSA
4161   BC84  80 80                    SUBA  #$80                      SET CARRY IF POSITIVE MANTISSA
4162   BC86  D7 4F             LBC86  STB   FPØEXP                    SAVE EXPONENT
4163   BC88  DC 8A                    LDD   ZERO                      * ZERO OUT ACCD AND
4164   BC8A  DD 52                    STD   FPA0+2                    * BOTTOM HALF OF FPA0
4165   BC8C  97 63                    STA   FPSBYT                    CLEAR SUB BYTE
4166   BC8E  97 54                    STA   FPØSGN                    CLEAR SIGN OF FPA0 MANTISSA
4167   BC90  7E BA 18                 JMP   LBA18                     GO NORMALIZE FPA0
4168
4169                         * ABS
4170   BC93  0F 54             ABS    CLR   FPØSGN                    FORCE MANTISSA SIGN OF FPA0 POSITIVE
4171   BC95  39                       RTS
4172                         * COMPARE A PACKED FLOATING POINT NUMBER POINTED TO
4173                         * BY (X) TO AN UNPACKED FP NUMBER IN FPA0. RETURN
4174                         * ZERO FLAG SET AND ACCB = 0, IF EQUAL; ACCB = 1 IF
4175                         * FPA0 > (X); ACCB = $FF IF FPA0 < (X)
4176   BC96  E6 84             LBC96  LDB   ,X                        CHECK EXPONENT OF (X)
4177   BC98  27 D3                    BEQ   LBC6D                     BRANCH IF FPA = 0
4178   BC9A  E6 01                    LDB   1,X                       GET MS BYTE OF MANTISSA OF (X)
4179   BC9C  D8 54                    EORB  FPØSGN                    EOR WITH SIGN OF FPA0
4180   BC9E  2B D1                    BMI   LBC71                     BRANCH IF SIGNS NOT =
4181                         * COMPARE FPA0 WITH FP NUMBER POINTED TO BY (X).
4182                         * FPA0 IS NORMALIZED, (X) IS PACKED.
4183   BCA0  D6 4F             LBCA0  LDB   FPØEXP                    * GET EXPONENT OF
```

```
4184   BCA2  E1 84                    CMPB  ,X                    * FPAØ, COMPARE TO EXPONENT OF
4185   BCA4  26 1D                    BNE   LBCC3                 * (X) AND BRANCH IF <>.
4186   BCA6  E6 Ø1                    LDB   1,X                   * GET MS BYTE OF (X), KEEP ONLY
4187   BCA8  CA 7F                    ORB   #$7F                  * THE SIGN BIT - 'AND' THE BOTTOM 7
4188   BCAA  D4 5Ø                    ANDB  FPAØ                  * BITS OF FPAØ INTO ACCB
4189   BCAC  E1 Ø1                    CMPB  1,X                   = COMPARE THE BOTTOM 7 BITS OF THE MANTISSA
4190   BCAE  26 13                    BNE   LBCC3                 = MS BYTE AND BRANCH IF <>
4191   BCBØ  D6 51                    LDB   FPAØ+1                * COMPARE 2ND BYTE
4192   BCB2  E1 Ø2                    CMPB  2,X                   * OF MANTISSA,
4193   BCB4  26 ØD                    BNE   LBCC3                 * BRANCH IF <>
4194   BCB6  D6 52                    LDB   FPAØ+2                = COMPARE 3RD BYTE
4195   BCB8  E1 Ø3                    CMPB  3,X                   = OF MANTISSA,
4196   BCBA  26 Ø7                    BNE   LBCC3                 = BRANCH IF <>
4197   BCBC  D6 53                    LDB   FPAØ+3                * SUBTRACT LS BYTE
4198   BCBE  EØ Ø4                    SUBB  4,X                   * OF (X) FROM LS BYTE OF
4199   BCCØ  26 Ø1                    BNE   LBCC3                 * FPAØ, BRANCH IF <>
42ØØ   BCC2  39                       RTS                         RETURN IF FP (X) = FPAØ
42Ø1   BCC3  56              LBCC3     RORB                        SHIFT CARRY TO BIT 7; CARRY SET IF FPAØ < (X)
42Ø2   BCC4  D8 54                     EORB  FPØSGN                TOGGLE SIZE COMPARISON BIT IF FPAØ IS NEGATIVE
42Ø3   BCC6  2Ø AB                     BRA   LBC73                 GO SET ACCB ACCORDING TO COMPARISON
4204                        * DE-NORMALIZE FPAØ : SHIFT THE MANTISSA UNTIL THE BINARY POINT IS TO THE RIGHT
4205                        * OF THE LEAST SIGNIFICANT BYTE OF THE MANTISSA
4206   BCC8  D6 4F           LBCC8     LDB   FPØEXP                GET EXPONENT OF FPAØ
4207   BCCA  27 3D                     BEQ   LBD09                 ZERO MANTISSA IF FPAØ = Ø
4208   BCCC  CØ AØ                     SUBB  #$AØ                  SUBTRACT $AØ FROM FPAØ EXPONENT T THIS WILL YIELD
4209                        *                                       THE NUMBER OF SHIFTS REQUIRED TO DENORMALIZE FPAØ. WHEN
4210                        *                                       THE EXPONENT OF FPAØ IS = ZERO, THEN THE BINARY POINT
4211                        *                                       WILL BE TO THE RIGHT OF THE MANTISSA
4212   BCCE  96 54                     LDA   FPØSGN                TEST SIGN OF FPAØ MANTISSA
4213   BCDØ  2A Ø5                     BPL   LBCD7                 BRANCH IF POSITIVE
4214   BCD2  Ø3 5B                     COM   FPCARY                COMPLEMENT CARRY IN BYTE
4215   BCD4  BD BA 7B                  JSR   LBA7B                 NEGATE MANTISSA OF FPAØ
4216   BCD7  8E ØØ 4F        LBCD7     LDX   #FPØEXP               POINT X TO FPAØ
4217   BCDA  C1 F8                     CMPB  #-8                   EXPONENT DIFFERENCE < -8?
4218   BCDC  2E Ø6                     BGT   LBCE4                 YES
4219   BCDE  BD BA AE                  JSR   LBAAE                 SHIFT FPAØ RIGHT UNTIL FPAØ EXPONENT = $AØ
422Ø   BCE1  ØF 5B                     CLR   FPCARY                CLEAR CARRY IN BYTE
4221   BCE3  39                       RTS
4222   BCE4  ØF 5B           LBCE4     CLR   FPCARY                CLEAR CARRY IN BYTE
4223   BCE6  96 54                     LDA   FPØSGN                * GET SIGN OF FPAØ MANTISSA
4224   BCE8  49                        ROLA                        * ROTATE IT INTO THE CARRY FLAG
4225   BCE9  Ø6 5Ø                     ROR   FPAØ                  ROTATE CARRY (MANTISSA SIGN) INTO BIT 7
4226                        *                                       OF LS BYTE OF MANTISSA
4227   BCEB  7E BA BA                  JMP   LBABA                 DE-NORMALIZE FPAØ
4228
4229                        * INT
4230                        * THE INT STATEMENT WILL "DENORMALIZE" FPAØ - THAT IS IT WILL SHIFT THE BINARY POINT
4231                        * TO THE EXTREME RIGHT OF THE MANTISSA TO FORCE ITS EXPONENT TO BE $AO. ONCE
4232                        * THIS IS DONE THE MANTISSA OF FPAØ WILL CONTAIN THE FOUR LEAST SIGNIFICANT
4233                        * BYTES OF THE INTEGER PORTION OF FPAØ. AT THE CONCLUSION OF THE DE-NORMALIZATION
4234                        * ONLY THE INTEGER PORTION OF FPAØ WILL REMAIN.
4235                        *
4236   BCEE  D6 4F           INT       LDB   FPØEXP                GET EXPONENT OF FPAØ
4237   BCFØ  C1 AØ                     CMPB  #$AØ                  LARGEST POSSIBLE INTEGER EXPONENT
4238   BCF2  24 1D                     BCC   LBD11                 RETURN IF FPAØ >= 32768
4239   BCF4  8D D2                     BSR   LBCC8                 SHIFT THE BINARY POINT ONE TO THE RIGHT OF THE
4240                        *                                       LS BYTE OF THE FPAØ MANTISSA
4241   BCF6  D7 63                     STB   FPSBYT                ACCB = Ø: ZERO OUT THE SUB BYTE
4242   BCF8  96 54                     LDA   FPØSGN                GET MANTISSA SIGN
4243   BCFA  D7 54                     STB   FPØSGN                FORCE MANTISSA SIGN TO BE POSITIVE
4244   BCFC  8Ø 8Ø                     SUBA  #$8Ø                  SET CARRY IF MANTISSA
4245   BCFE  86 AØ                     LDA   #$AØ                  * GET DENORMALIZED EXPONENT AND
4246   BDØØ  97 4F                     STA   FPØEXP                * SAVE IT IN FPAØ EXPONENT
4247   BDØ2  96 53                     LDA   FPAØ+3                = GET LS BYTE OF FPAØ AND
4248   BDØ4  97 Ø1                     STA   CHARAC                = SAVE IT IN CHARAC
4249   BDØ6  7E BA 18                  JMP   LBA18                 NORMALIZE FPAØ
4250
4251   BDØ9  D7 5Ø           LBDØ9     STB   FPAØ                  * LOAD MANTISSA OF FPAØ WITH CONTENTS OF ACCB
4252   BDØB  D7 51                     STB   FPAØ+1                *
4253   BDØD  D7 52                     STB   FPAØ+2                *
4254   BDØF  D7 53                     STB   FPAØ+3                *
4255   BD11  39              LBD11     RTS                         *
4256
4257                        * CONVERT ASCII STRING TO FLOATING POINT
4258   BD12  9E 8A           LBD12     LDX   ZERO                  (X) = Ø
4259   BD14  9F 54                     STX   FPØSGN                * ZERO OUT FPAØ & THE SIGN FLAG (COEFCT)
4260   BD16  9F 4F                     STX   FPØEXP                *
4261   BD18  9F 51                     STX   FPAØ+1                *
4262   BD1A  9F 52                     STX   FPAØ+2                *
4263   BD1C  9F 47                     STX   V47                   INITIALIZE EXPONENT & EXPONENT SIGN FLAG TO ZERO
4264   BD1E  9F 45                     STX   V45                   INITIALIZE RIGHT DECIMAL CTR & DECIMAL PT FLAG TO Ø
4265   BD2Ø  25 64                     BCS   LBD86                 IF CARRY SET (NUMERIC CHARACTER), ASSUME ACCA CONTAINS FIRST
4266                        *                                       NUMERIC CHAR, SIGN IS POSITIVE AND SKIP THE RAM HOOK
4267   BD22  BD Ø1 97                  JSR   RVEC19                HOOK INTO RAM
4268   BD25  81 2D                     CMPA  #'-                   * CHECK FOR A LEADING MINUS SIGN AND BRANCH
4269   BD27  26 Ø4                     BNE   LBD2D                 * IF NO MINUS SIGN
4270   BD29  Ø3 55                     COM   COEFCT                TOGGLE SIGN; Ø = +; FF = -
4271   BD2B  2Ø Ø4                     BRA   LBD31                 INTERPRET THE REST OF THE STRING
4272   BD2D  81 2B           LBD2D     CMPA  #'+                   * CHECK FOR LEADING PLUS SIGN AND BRANCH
```

```
4273  BD2F  26 04                    BNE   LBD35                    * IF NOT A PLUS SIGN
4274  BD31  9D 9F             LBD31  JSR   GETNCH                    GET NEXT INPUT CHARACTER FROM BASIC
4275  BD33  25 51                    BCS   LBD86                    BRANCH IF NUMERIC CHARACTER
4276  BD35  81 2E             LBD35  CMPA  #'.                       DECIMAL POINT?
4277  BD37  27 28                    BEQ   LBD61                    YES
4278  BD39  81 45                    CMPA  #'E                      "E" SHORTHAND FORM (SCIENTIFIC NOTATION)?
4279  BD3B  26 28                    BNE   LBD65                    NO
4280                           * EVALUATE EXPONENT OF EXPONENTIAL FORMAT
4281  BD3D  9D 9F                    JSR   GETNCH                    GET NEXT INPUT CHARACTER FROM BASIC
4282  BD3F  25 64                    BCS   LBDA5                     BRANCH IF NUMERIC
4283  BD41  81 AC                    CMPA  #$AC                     MINUS TOKEN?
4284  BD43  27 0E                    BEQ   LBD53                    YES
4285  BD45  81 2D                    CMPA  #'-                      ASCII MINUS?
4286  BD47  27 0A                    BEQ   LBD53                    YES
4287  BD49  81 AB                    CMPA  #$AB                     PLUS TOKEN?
4288  BD4B  27 08                    BEQ   LBD55                    YES
4289  BD4D  81 2B                    CMPA  #'+                      ASCII PLUS?
4290  BD4F  27 04                    BEQ   LBD55                    YES
4291  BD51  20 06                    BRA   LBD59                    BRANCH IF NO SIGN FOUND
4292  BD53  03 48              LBD53  COM   V48                      SET EXPONENT SIGN FLAG TO NEGATIVE
4293                           * STRIP A DECIMAL NUMBER FROM BASIC LINE, CONVERT IT TO BINARY IN V47
4294  BD55  9D 9F             LBD55  JSR   GETNCH                    GET NEXT INPUT CHARACTER FROM BASIC
4295  BD57  25 4C                    BCS   LBDA5                     IF NUMERIC CHARACTER, CONVERT TO BINARY
4296  BD59  0D 48              LBD59  TST   V48                      * CHECK EXPONENT SIGN FLAG
4297  BD5B  27 08                    BEQ   LBD65                    * AND BRANCH IF POSITIVE
4298  BD5D  00 47                    NEG   V47                      NEGATE VALUE OF EXPONENT
4299  BD5F  20 04                    BRA   LBD65
4300  BD61  03 46              LBD61  COM   V46                      *TOGGLE DECIMAL PT FLAG AND INTERPRET ANOTHER
4301  BD63  26 CC                    BNE   LBD31                    *CHARACTER IF <> 0 - TERMINATE INTERPRETATION
4302                           *                                     IF SECOND DECIMAL POINT
4303                           * ADJUST FPA0 FOR THE DECIMAL EXPONENT IN V47
4304  BD65  96 47              LBD65  LDA   V47                      * GET EXPONENT, SUBTRACT THE NUMBER OF
4305  BD67  90 45                    SUBA  V45                      * PLACES TO THE RIGHT OF DECIMAL POINT
4306  BD69  97 47                    STA   V47                      * AND RESAVE IT.
4307  BD6B  27 12                    BEQ   LBD7F                    EXIT ROUTINE IF ADJUSTED EXPONENT = ZERO
4308  BD6D  2A 09                    BPL   LBD78                    BRANCH IF POSITIVE EXPONENT
4309  BD6F  BD BB 82           LBD6F  JSR   LBB82                    DIVIDE FPA0 BY 10
4310  BD72  0C 47                    INC   V47                      INCREMENT EXPONENT COUNTER (MULTIPLY BY 10)
4311  BD74  26 F9                    BNE   LBD6F                    KEEP MULTIPLYING
4312  BD76  20 07                    BRA   LBD7F                    EXIT ROUTINE
4313  BD78  BD BB 6A           LBD78  JSR   LBB6A                    MULTIPLY FPA0 BY 10
4314  BD7B  0A 47                    DEC   V47                      DECREMENT EXPONENT COUNTER (DIVIDE BY 10)
4315  BD7D  26 F9                    BNE   LBD78                    KEEP MULTIPLYING
4316  BD7F  96 55              LBD7F  LDA   COEFCT                  GET THE SIGN FLAG
4317  BD81  2A 8E                    BPL   LBD11                    RETURN IF POSITIVE
4318  BD83  7E BE E9                 JMP   LBEE9                    TOGGLE MANTISSA SIGN OF FPA0, IF NEGATIVE
4319                           *MULTIPLY FPA0 BY TEN AND ADD ACCA TO THE RESULT
4320  BD86  D6 45              LBD86  LDB   V45                      *GET THE RIGHT DECIMAL COUNTER AND SUBTRACT
4321  BD88  D0 46                    SUBB  V46                      *THE DECIMAL POINT FLAG FROM IT. IF DECIMAL POINT
4322  BD8A  D7 45                    STB   V45                      *FLAG=0, NOTHING HAPPENS. IF DECIMAL POINT FLAG IS
4323                           *                                     -1, THEN RIGHT DECIMAL COUNTER IS INCREMENTED BY ONE
4324  BD8C  34 02                    PSHS  A                        SAVE NEW DIGIT ON STACK
4325  BD8E  BD BB 6A                 JSR   LBB6A                    MULTIPLY FPA0 BY 10
4326  BD91  35 04                    PULS  B                        GET NEW DIGIT BACK
4327  BD93  C0 30                    SUBB  #'0                      MASK OFF ASCII
4328  BD95  8D 02                    BSR   LBD99                    ADD ACCB TO FPA0
4329  BD97  20 98                    BRA   LBD31                    GET ANOTHER CHARACTER FROM BASIC
4330  BD99  BD BC 2F           LBD99  JSR   LBC2F                    PACK FPA0 AND SAVE IT IN FPA3
4331  BD9C  BD BC 7C                 JSR   LBC7C                    CONVERT ACCB TO FP NUMBER IN FPA0
4332  BD9F  8E 00 40                 LDX   #V40                     * ADD FPA0 TO
4333  BDA2  7E B9 C2                 JMP   LB9C2                    * FPA3
4334                           * MULTIPLY V47 BY 10 AND ADD TO ASCII NUMBER IN
4335                           * ACCA - SAVE BINARY RESULT IN V47
4336  BDA5  D6 47              LBDA5  LDB   V47
4337  BDA7  58                       ASLB                           TIMES 2
4338  BDA8  58                       ASLB                           TIMES 4
4339  BDA9  DB 47                    ADDB  V47                      ADD 1 = TIMES 5
4340  BDAB  58                       ASLB                           TIMES 10
4341  BDAC  80 30                    SUBA  #'0                      *MASK OFF ASCII FROM ACCA, PUSH
4342  BDAE  34 04                    PSHS  B                        *RESULT ONTO THE STACK AND
4343  BDB0  AB E0                    ADDA  ,S+                      ADD IT TO ACCB
4344  BDB2  97 47                    STA   V47                      SAVE IN V47
4345  BDB4  20 9F                    BRA   LBD55                    INTERPRET ANOTHER CHARACTER
4346                           *
4347  BDB6  9B 3E BC 1F FD     LBDB6  FCB   $9B,$3E,$BC,$1F,$FD      * 99999999.9
4348  BDBB  9E 6E 6B 27 FD     LBDBB  FCB   $9E,$6E,$6B,$27,$FD      * 999999999
4349  BDC0  9E 6E 6B 28 00     LBDC0  FCB   $9E,$6E,$6B,$28,$00      * 1E + 09
4350                           *
4351  BDC5  8E AB E7           LBDC5  LDX   #LABE8-1                 POINT X TO " IN " MESSAGE
4352  BDC8  8D 0C                    BSR   LBDD6                    COPY A STRING FROM (X) TO CONSOLE OUT
4353  BDCA  DC 68                    LDD   CURLIN                   GET CURRENT BASIC LINE NUMBER TO ACCD
4354                           * CONVERT VALUE IN ACCD INTO A DECIMAL NUMBER
4355                           * AND PRINT IT TO CONSOLE OUT
4356  BDCC  DD 50              LBDCC  STD   FPA0                     SAVE ACCD IN TOP HALF OF FPA0
4357  BDCE  C6 90                    LDB   #$90                     REQ D EXPONENT IF TOP HALF OF ACCD = INTEGER
4358  BDD0  43                       COMA                           SET CARRY FLAG - FORCE POSITIVE MANTISSA
4359  BDD1  BD BC 86                 JSR   LBC86                    ZERO BOTTOM HALF AND SIGN OF FPA0, THEN
4360                           *                                     SAVE EXPONENT AND NORMALIZE IT
4361  BDD4  8D 03                    BSR   LBDD9                    CONVERT FP NUMBER TO ASCII STRING
```

```
4362   BDD6  7E B9 9C          LBDD6     JMP   LB99C                    COPY A STRING FROM (X) TO CONSOLE OUT
4363
4364                           * CONVERT FP NUMBER TO ASCII STRING
4365   BDD9  CE Ø3 DA          LBDD9     LDU   #STRBUF+3                POINT U TO BUFFER WHICH WILL NOT CAUSE
4366                           *                                        THE STRING TO BE STORED IN STRING SPACE
4367   BDDC  86 2Ø             LBDDC     LDA   #SPACE                   SPACE = DEFAULT SIGN FOR POSITIVE #
4368   BDDE  D6 54                       LDB   FPØSGN                   GET SIGN OF FPAØ
4369   BDEØ  2A Ø2                       BPL   LBDE4                    BRANCH IF POSITIVE
4370   BDE2  86 2D                       LDA   #'-                      ASCII MINUS SIGN
4371   BDE4  A7 CØ             LBDE4     STA   ,U+                      STORE SIGN OF NUMBER
4372   BDE6  DF 64                       STU   COEFPT                   SAVE BUFFER POINTER
4373   BDE8  97 54                       STA   FPØSGN                   SAVE SIGN (IN ASCII)
4374   BDEA  86 3Ø                       LDA   #'Ø                      ASCII ZERO IF EXPONENT = Ø
4375   BDEC  D6 4F                       LDB   FPØEXP                   GET FPAØ EXPONENT
4376   BDEE  1Ø 27 ØØ C6                 LBEQ  LBEB8                    BRANCH IF FPAØ = Ø
4377   BDF2  4F                          CLRA                           BASE 1Ø EXPONENT=Ø FOR FP NUMBER > 1
4378   BDF3  C1 8Ø                       CMPB  #$8Ø                     CHECK EXPONENT
4379   BDF5  22 Ø8                       BHI   LBDFF                    BRANCH IF FP NUMBER > 1
4380                           * IF FPAØ < 1.Ø, MULTIPLY IT BY 1E+Ø9 TO SPEED UP THE CONVERSION PROCESS
4381   BDF7  8E BD CØ                    LDX   #LBDCØ                   POINT X TO FP 1E+Ø9
4382   BDFA  BD BA CA                    JSR   LBACA                    MULTIPLY FPAØ BY (X)
4383   BDFD  86 F7                       LDA   #-9                      BASE 1Ø EXPONENT = -9
4384   BDFF  97 45              LBDFF     STA   V45                      BASE 1Ø EXPONENT
4385                           * PSEUDO - NORMALIZE THE FP NUMBER TO A VALUE IN THE RANGE
4386                           * OF 999,999,999 RO 99,999,999.9 - THIS IS THE LARGEST
4387                           * NUMBER RANGE IN WHICH ALL OF THE DIGITS ARE
4388                           * SIGNIFICANT WHICH CAN BE DISPLAYED WITHOUT USING
4389                           * SCIENTIFIC NOTATION
4390   BEØ1  8E BD BB           LBEØ1     LDX   #LBDBB                   POINT X TO FP 999,999,999
4391   BEØ4  BD BC AØ                    JSR   LBCAØ                    COMPARE FPAØ TO 999,999,999
4392   BEØ7  2E ØF                       BGT   LBE18                    BRANCH IF > 999,999,999
4393   BEØ9  8E BD B6           LBEØ9     LDX   #LBDB6                   POINT X TO FP 99,999,999.9
4394   BEØC  BD BC AØ                    JSR   LBCAØ                    COMPARE FPAØ TO 99,999,999.9
4395   BEØF  2E ØE                       BGT   LBE1F                    BRANCH IF > 99,999,999.9 (IN RANGE)
4396   BE11  BD BB 6A                    JSR   LBB6A                    MULTIPLY FPAØ BY 1Ø
4397   BE14  ØA 45                       DEC   V45                      SUBTRACT ONE FROM DECIMAL OFFSET
4398   BE16  2Ø F1                       BRA   LBEØ9                    PSEUDO - NORMALIZE SOME MORE
4399   BE18  BD BB 82           LBE18     JSR   LBB82                    DIVIDE FPAØ BY 1Ø
44ØØ   BE1B  ØC 45                       INC   V45                      ADD ONE TO BASE 1Ø EXPONENT
44Ø1   BE1D  2Ø E2                       BRA   LBEØ1                    PSEUDO - NORMALIZE SOME MORE
44Ø2   BE1F  BD B9 B4           LBE1F     JSR   LB9B4                    ADD .5 TO FPAØ (ROUND OFF)
44Ø3   BE22  BD BC C8                    JSR   LBCC8                    CONVERT FPAØ TO AN INTEGER
44Ø4   BE25  C6 Ø1                       LDB   #1                       DEFAULT DECIMAL POINT FLAG (FORCE IMMED DECIMAL PT)
44Ø5   BE27  96 45                       LDA   V45                      * GET BASE 1Ø EXPONENT AND ADD TEN TO IT
44Ø6   BE29  8B ØA                       ADDA  #9+1                     * (NUMBER  NORMALIZED  TO 9 PLACES & DECIMAL PT)
44Ø7   BE2B  2B Ø9                       BMI   LBE36                    BRANCH IF NUMBER < 1.Ø
44Ø8   BE2D  81 ØB                       CMPA  #9+2                     NINE PLACES MAY BE DISPLAYED WITHOUT
44Ø9                           *                                        USING SCIENTIFIC NOTATION
4410   BE2F  24 Ø5                       BCC   LBE36                    BRANCH IF SCIENTIFIC NOTATION REQUIRED
4411   BE31  4A                          DECA                           * SUBTRACT 1 FROM MODIFIED BASE 1Ø EXPONENT CTR
4412   BE32  1F 89                       TFR   A,B                      * AND SAVE IT IN ACCB (DECiMAL POINT FLAG)
4413   BE34  86 Ø2                       LDA   #2                       FORCE EXPONENT = Ø - DON'T USE SCIENTIFIC NOTATION
4414   BE36  4A                 LBE36     DECA                           * SUBTRACT TWO (WITHOUT AFFECTING CARRY)
4415   BE37  4A                          DECA                           * FROM BASE 1Ø EXPONENT
4416   BE38  97 47                       STA   V47                      SAVE EXPONENT - ZERO EXPONENT = DO NOT DISPLAY
4417                           *                                        IN SCIENTIFIC NOTATION
4418   BE3A  D7 45                       STB   V45                      DECIMAL POINT FLAG - NUMBER OF PLACES TO
4419                           *                                        LEFT OF DECIMAL POINT
4420   BE3C  2E ØD                       BGT   LBE4B                    BRANCH IF >= 1
4421   BE3E  DE 64                       LDU   COEFPT                   POINT U TO THE STRING BUFFER
4422   BE4Ø  86 2E                       LDA   #'.                      * STORE A PERIOD
4423   BE42  A7 CØ                       STA   ,U+                      * IN THE BUFFER
4424   BE44  5D                          TSTB                           CHECK DECIMAL POINT FLAG
4425   BE45  27 Ø4                       BEQ   LBE4B                    BRANCH IF NOTHING TO LEFT OF DECIMAL POINT
4426   BE47  86 3Ø                       LDA   #'Ø                      * STORE A ZERO
4427   BE49  A7 CØ                       STA   ,U+                      * IN THE BUFFER
4428
4429                           * CONVERT FPAØ INTO A STRING OF ASCII DIGITS
4430   BE4B  8E BE C5           LBE4B     LDX   #LBEC5                   POINT X TO FP POWER OF 1Ø MANTISSA
4431   BE4E  C6 8Ø                       LDB   #Ø+$8Ø                   INITIALIZE DIGIT COUNTER TO Ø+$8Ø
4432                           * BIT 7 SET IS USED TO INDICATE THAT THE POWER OF 1Ø MANTISSA
4433                           * IS NEGATIVE. WHEN YOU 'ADD' A NEGATIVE MANTISSA, IT IS
4434                           * THE SAME AS SUBTRACTING A POSITIVE ONE AND BIT 7 OF ACCB IS HOW
4435                           * THE ROUTINE KNOWS THAT A 'SUBTRACTION' IS OCCURING.
4436   BE5Ø  96 53              LBE5Ø     LDA   FPAØ+3                   * ADD MANTISSA LS
4437   BE52  AB Ø3                       ADDA  3,X                      * BYTE OF FPAØ
4438   BE54  97 53                       STA   FPAØ+3                   * AND (X)
4439   BE56  96 52                       LDA   FPAØ+2                   = ADD MANTISSA
4440   BE58  A9 Ø2                       ADCA  2,X                      = NUMBER 3 BYTE OF
4441   BE5A  97 52                       STA   FPAØ+2                   = FPAØ AND (X)
4442   BE5C  96 51                       LDA   FPAØ+1                   * ADD MANTISSA
4443   BE5E  A9 Ø1                       ADCA  1,X                      * NUMBER 2 BYTE OF
4444   BE6Ø  97 51                       STA   FPAØ+1                   * FPAØ AND (X)
4445   BE62  96 5Ø                       LDA   FPAØ                     = ADD MANTISSA
4446   BE64  A9 84                       ADCA  ,X                       = MS BYTE OF
4447   BE66  97 5Ø                       STA   FPAØ                     = FPAØ AND (X)
4448   BE68  5C                          INCB                           ADD ONE TO DIGIT COUNTER
4449   BE69  56                          RORB                           ROTATE CARRY INTO BIT 7
4450   BE6A  59                          ROLB                           *SET OVERFLOW FLAG AND BRANCH IF CARRY = 1 AND
```

```
4451   BE6B  28 E3                  BVC   LBE5Ø                  *POSITIVE MANTISSA OR CARRY = Ø AND NEG MANTISSA
4452   BE6D  24 Ø3                  BCC   LBE72                  BRANCH IF NEGATIVE MANTISSA
4453   BE6F  CØ ØB                  SUBB  #1Ø+1                  * TAKE THE 9 S COMPLEMENT IF
4454   BE71  5Ø                     NEGB                         * ADDING MANTISSA
4455   BE72  CB 2F           LBE72  ADDB  #'Ø-1                  ADD ASCII OFFSET TO DIGIT
4456   BE74  3Ø Ø4                  LEAX  4,X                    MOVE TO NEXT POWER OF 1Ø MANTISSA
4457   BE76  1F 98                  TFR   B,A                    SAVE DIGIT IN ACCA
4458   BE78  84 7F                  ANDA  #$7F                   MASK OFF BIT 7 (ADD/SUBTRACT FLAG)
4459   BE7A  A7 CØ                  STA   ,U+                    STORE DIGIT IN STRING BUFFER
4460   BE7C  ØA 45                  DEC   V45                    DECREMENT DECIMAL POINT FLAG
4461   BE7E  26 Ø4                  BNE   LBE84                  BRANCH IF NOT TIME FOR DECIMAL POINT
4462   BEØØ  86 2E                  LDA   #'.                    * STORE DECIMAL POINT IN
4463   BE82  A7 CØ                  STA   ,U+                    * STRING BUFFER
4464   BE84  53              LBE84  COMB                         TOGGLE BIT 7 (ADD/SUBTRACT FLAG)
4465   BE85  C4 8Ø                  ANDB  #$8Ø                   MASK OFF ALL BUT ADD/SUBTRACT FLAG
4466   BE87  8C BE E9               CMPX  #LBEC5+9*4             COMPARE X TO END OF MANTISSA TABLE
4467   BE8A  26 C4                  BNE   LBE5Ø                  BRANCH IF NOT AT END OF TABLE
4468                         * BLANK TRAILING ZEROS AND STORE EXPONENT IF ANY
4469   BE8C  A6 C2           LBE8C  LDA   ,-U                    GET THE LAST CHARACTER; MOVE POINTER BACK
4470   BE8E  81 3Ø                  CMPA  #'Ø                    WAS IT A ZERO?
4471   BE9Ø  27 FA                  BEQ   LBE8C                  IGNORE TRAILING ZEROS IF SO
4472   BE92  81 2E                  CMPA  #'.                    CHECK FOR DECIMAL POINT
4473   BE94  26 Ø2                  BNE   LBE98                  BRANCH IF NOT DECIMAL POINT
4474   BE96  33 5F                  LEAU  -1,U                   STEP OVER THE DECIMAL POINT
4475   BE98  86 2B           LBE98  LDA   #'+                    ASCII PLUS SIGN
4476   BE9A  D6 47                  LDB   V47                    GET SCIENTIFIC NOTATION EXPONENT
4477   BE9C  27 1C                  BEQ   LBEBA                  BRANCH IF NOT SCIENTIFIC NOTATION
4478   BE9E  2A Ø3                  BPL   LBEA3                  BRANCH IF POSITIVE EXPONENT
4479   BEAØ  86 2D                  LDA   #'-                    ASCII MINUS SIGN
4480   BEA2  5Ø                     NEGB                         NEGATE EXPONENT IF NEGATIVE
4481   BEA3  A7 42           LBEA3  STA   2,U                    STORE EXPONENT SIGN IN STRING
4482   BEA5  86 45                  LDA   #'E                    * GET ASCII  E  (SCIENTIFIC NOTATION
4483   BEA7  A7 41                  STA   1,U                    * FLAG) AND SAVE IT IN THE STRING
4484   BEA9  86 2F                  LDA   #'Ø-1                  INITIALIZE ACCA TO ASCII ZERO
4485                         * CONVERT BINARY VALUE IN ACCB TO DECIMAL
4486                         * ASCII NUMBER (< 1ØØ) IN ACCD
4487   BEAB  4C              LBEAB  INCA                         ADD ONE TO 1Ø S DIGIT OF EXPONENT
4488   BEAC  CØ ØA                  SUBB  #1Ø                    SUBTRACT 1Ø FROM ACCB
4489   BEAE  24 FB                  BCC   LBEAB                  ADD 1 TO 1Ø S DIGIT IF NO CARRY
4490   BEBØ  CB 3A                  ADDB  #'9+1                  CONVERT UNITS DIGIT TO ASCII
4491   BEB2  ED 43                  STD   3,U                    SAVE EXPONENT IN STRING
4492   BEB4  6F 45                  CLR   5,U                    CLEAR LAST BYTE (TERMINATOR)
4493   BEB6  2Ø Ø4                  BRA   LBEBC                  GO RESET POINTER
4494   BEB8  A7 C4           LBEB8  STA   ,U                     STORE LAST CHARACTER
4495   BEBA  6F 41           LBEBA  CLR   1,U                    CLEAR LAST BYTE (TERMINATOR - REQUIRED BY
4496                         *                                   PRINT SUBROUTINES)
4497   BEBC  8E Ø3 DA        LBEBC  LDX   #STRBUF+3              RESET POINTER TO START OF BUFFER
4498   BEBF  39                     RTS
4499                         *
4500   BEC Ø  8Ø ØØ ØØ ØØ ØØ LBEC Ø  FCB   $8Ø,$ØØ,$ØØ,$ØØ,$ØØ  FLOATING POINT .5
4501                         *
4502                         *** TABLE OF UNNORMALIZED POWERS OF 1Ø
4503   BEC5  FA ØA 1F ØØ      LBEC5  FCB   $FA,$ØA,$1F,$ØØ       -1ØØØØØØØ
4504   BEC9  ØØ 98 96 8Ø      LBEC9  FCB   $ØØ,$98,$96,$8Ø        1ØØØØØØ
4505   BECD  FF FØ BD CØ      LBECD  FCB   $FF,$FØ,$BD,$CØ       -1ØØØØØ
4506   BED1  ØØ Ø1 86 AØ      LBED1  FCB   $ØØ,$Ø1,$86,$AØ        1ØØØØ
4507   BED5  FF FF D8 FØ      LBED5  FCB   $FF,$FF,$D8,$FØ       -1ØØØ
4508   BED9  ØØ ØØ Ø3 E8      LBED9  FCB   $ØØ,$ØØ,$Ø3,$E8        1ØØ
4509   BEDD  FF FF FF 9C      LBEDD  FCB   $FF,$FF,$FF,$9C       -1Ø
4510   BEE1  ØØ ØØ ØØ ØA      LBEE1  FCB   $ØØ,$ØØ,$ØØ,$ØA        1
4511   BEE5  FF FF FF FF      LBEE5  FCB   $FF,$FF,$FF,$FF       -1
4512                         *
4513                         *
4514   BEE9  96 4F            LBEE9  LDA   FPØEXP                 GET EXPONENT OF FPAØ
4515   BEEB  27 Ø2                   BEQ   LBEEF                 BRANCH IF FPAØ = Ø
4516   BEED  Ø3 54                   COM   FPØSGN                TOGGLE MANTISSA SIGN OF FPAØ
4517   BEEF  39              LBEEF  RTS
4518                         * EXPAND A POLYNOMIAL OF THE FORM
4519                         * AQ+BQ**3+CQ**5+DQ**7.... WHERE Q = FPAØ
4520                         * AND THE X REGISTER POINTS TO A TABLE OF
4521                         * COEFFICIENTS A,B,C,D....
4522   BEFØ  9F 64            LBEFØ  STX   COEFPT                 SAVE COEFFICIENT TABLE POINTER
4523   BEF2  BD BC 2F                JSR   LBC2F                 MOVE FPAØ TO FPA3
4524   BEF5  8D Ø5                   BSR   LBEFC                 MULTIPLY FPA3 BY FPAØ
4525   BEF7  8D Ø8                   BSR   LBFØ1                 EXPAND POLYNOMIAL
4526   BEF9  8E ØØ 4Ø                LDX   #V4Ø                  POINT X TO FPA3
4527   BEFC  7E BA CA         LBEFC  JMP   LBACA                 MULTIPLY (X) BY FPAØ
4528
4529                         * CALCULATE THE VALUE OF AN EXPANDED POLYNOMIAL
4530                         * EXPRESSION. ENTER WITH (X) POINTING TO A TABLE
4531                         * OF COEFFICIENTS, THE FIRST BYTE OF WHICH IS THE
4532                         * NUMBER OF (COEFFICIENTS-1) FOLLOWED BY THAT NUMBER
4533                         * OF PACKED FLOATING POINT NUMBERS. THE
4534                         * POLYNOMIAL IS EVALUATED AS FOLLOWS: VALUE =
4535                         * (((FPAØ*Y Ø+Y1)*FPAØ+Y2)*FPAØ YN)
4536   BEFF  9F 64            LBEFF  STX   COEFPT                 SAVE COEFFICIENT TABLE POINTER
4537   BFØ1  BD BC 2A         LBFØ1  JSR   LBC2A                 MOVE FPAØ TO FPA4
4538   BFØ4  9E 64                   LDX   COEFPT                GET THE COEFFICIENT POINTER
4539   BFØ6  E6 8Ø                   LDB   ,X+                   GET THE TOP OF COEFFICIENT TABLE TO
```

```
4540   BF08  D7 55                    STB   COEFCT                  * USE AND STORE IT IN TEMPORARY COUNTER
4541   BF0A  9F 64                    STX   COEFPT                  SAVE NEW COEFFICIENT POINTER
4542   BF0C  8D EE            LBF0C   BSR   LBEFC                   MULTIPLY (X) BY FPA0
4543   BF0E  9E 64                    LDX   COEFPT                  *GET COEFFICIENT POINTER
4544   BF10  30 05                    LEAX  5,X                     *MOVE TO NEXT FP NUMBER
4545   BF12  9F 64                    STX   COEFPT                  *SAVE NEW COEFFICIENT POINTER
4546   BF14  BD B9 C2                 JSR   LB9C2                   ADD (X) AND FPA0
4547   BF17  8E 00 45                 LDX   #V45                    POINT (X) TO FPA4
4548   BF1A  0A 55                    DEC   COEFCT                  DECREMENT TEMP COUNTER
4549   BF1C  26 EE                    BNE   LBF0C                   BRANCH IF MORE COEFFICIENTS LEFT
4550   BF1E  39                       RTS
4551
4552                          * RND
4553   BF1F  BD BC 6D          RND    JSR   LBC6D                   TEST FPA0
4554   BF22  2B 21                    BMI   LBF45                   BRANCH IF FPA0 = NEGATIVE
4555   BF24  27 15                    BEQ   LBF3B                   BRANCH IF FPA0 = 0
4556   BF26  8D 10                    BSR   LBF38                   CONVERT FPA0 TO AN INTEGER
4557   BF28  BD BC 2F                 JSR   LBC2F                   PACK FPA0 TO FPA3
4558   BF2B  8D 0E                    BSR   LBF3B                   GET A RANDOM NUMBER: FPA0 < 1.0
4559   BF2D  8E 00 40                 LDX   #V40                    POINT (X) TO FPA3
4560   BF30  8D CA                    BSR   LBEFC                   MULTIPLY (X) BY FPA0
4561   BF32  8E BA C5                 LDX   #LBAC5                  POINT (X) TO FP VALUE OF 1.0
4562   BF35  BD B9 C2                 JSR   LB9C2                   ADD 1.0 TO FPA0
4563   BF38  7E BC EE          LBF38  JMP   INT                     CONVERT FPA0 TO AN INTEGER
4564                          * CALCULATE A RANDOM NUMBER IN THE RANGE 0.0 < X <= 1.0
4565   BF3B  BE 01 16          LBF3B  LDX   RVSEED+1                * MOVE VARIABLE
4566   BF3E  9F 50                    STX   FPA0                    * RANDOM NUMBER
4567   BF40  BE 01 18                 LDX   RVSEED+3                * SEED TO
4568   BF43  9F 52                    STX   FPA0+2                  * FPA0
4569   BF45  BE BF 74          LBF45  LDX   RSEED                   = MOVE FIXED
4570   BF48  9F 5D                    STX   FPA1                    = RANDOM NUMBER
4571   BF4A  BE BF 76                 LDX   RSEED+2                 = SEED TO
4572   BF4D  9F 5F                    STX   FPA1+2                  = MANTISSA OF FPA0
4573   BF4F  BD BA D0                 JSR   LBAD0                   MULTIPLY FPA0 X FPA1
4574   BF52  DC AD                    LDD   VAD                     GET THE TWO LOWEST ORDER PRODUCT BYTES
4575   BF54  C3 65 8B                 ADDD  #$658B                  ADD A CONSTANT
4576   BF57  FD 01 18                 STD   RVSEED+3                SAVE NEW LOW ORDER VARIABLE RANDOM # SEED
4577   BF5A  DD 52                    STD   FPA0+2                  SAVE NEW LOW ORDER BYTES OF FPA0 MANTISSA
4578   BF5C  DC AB                    LDD   VAB                     GET 2 MORE LOW ORDER PRODUCT BYTES
4579   BF5E  C9 B0                    ADCB  #$B0                    ADD A CONSTANT
4580   BF60  89 05                    ADCA  #5                      ADD A CONSTANT
4581   BF62  FD 01 16                 STD   RVSEED+1                SAVE NEW HIGH ORDER VARIABLE RANDOM # SEED
4582   BF65  DD 50                    STD   FPA0                    SAVE NEW HIGH ORDER FPA0 MANTISSA
4583   BF67  0F 54                    CLR   FP0SGN                  FORCE FPA0 MANTISSA = POSITIVE
4584   BF69  86 80                    LDA   #$80                    * SET FPA0 BIASED EXPONENT
4585   BF6B  97 4F                    STA   FP0EXP                  * TO 0 1 < FPA0 < 0
4586   BF6D  96 15                    LDA   FPA2+2                  GET A BYTE FROM FPA2 (MORE RANDOMNESS)
4587   BF6F  97 63                    STA   FPSBYT                  SAVE AS SUB BYTE
4588   BF71  7E BA 1C                 JMP   LBA1C                   NORMALIZE FPA0
4589                          *
4590   BF74  40 E6             RSEED  FDB   $40E6                   *CONSTANT RANDOM NUMBER GENERATOR SEED
4591   BF76  4D AB                    FDB   $4DAB                   *
4592
4593                          * SIN
4594                          * THE SIN FUNCTION REQUIRES AN ARGUMENT IN RADIANS AND WILL REPEAT ITSELF EVERY
4595                          * 2*PI RADIANS. THE ARGUMENT IS DIVIDED BY 2*PI AND ONLY THE FRACTIONAL PART IS
4596                          * RETAINED. SINCE THE ARGUMENT WAS DIVIDED BY 2*P1, THE COEFFICIENTS MUST BE
4597                          * MULTIPLIED BY THE APPROPRIATE POWER OF 2*PI.
4598
4599                          * SIN IS EVALUATED USING THE TRIGONOMETRIC IDENTITIES BELOW:
4600                          * SIN(X)=SIN(PI-X) & -SIN(PI/2-X)=SIN((3*PI)/2+X)
4601   BF78  BD BC 5F          SIN    JSR   LBC5F                   COPY FPA0 TO FPA1
4602   BF7B  8E BF BD                 LDX   #LBFBD                  POINT (X) TO 2*PI
4603   BF7E  D6 61                    LDB   FP1SGN                  *GET MANTISSA SIGN OF FPA1
4604   BF80  BD BB 89                 JSR   LBB89                   *AND DIVIDE FPA0 BY 2*PI
4605   BF83  BD BC 5F                 JSR   LBC5F                   COPY FPA0 TO FPA1
4606   BF86  8D B0                    BSR   LBF38                   CONVERT FPA0 TO AN INTEGER
4607   BF88  0F 62                    CLR   RESSGN                  SET RESULT SIGN = POSITIVE
4608   BF8A  96 5C                    LDA   FP1EXP                  *GET EXPONENT OF FPA1
4609   BF8C  D6 4F                    LDB   FP0EXP                  *GET EXPONENT OF FPA0
4610   BF8E  BD B9 BC                 JSR   LB9BC                   *SUBTRACT FPA0 FROM FPA1
4611                          *NOW FPA0 CONTAINS ONLY THE FRACTIONAL PART OF ARGUMENT/2*PI
4612   BF91  8E BF C2                 LDX   #LBFC2                  POINT X TO FP (.25)
4613   BF94  BD B9 B9                 JSR   LB9B9                   SUBTRACT FPA0 FROM .25 (PI/2)
4614   BF97  96 54                    LDA   FP0SGN                  GET MANTISSA SIGN OF FPA0
4615   BF99  34 02                    PSHS  A                       SAVE IT ON STACK
4616   BF9B  2A 09                    BPL   LBFA6                   BRANCH IF MANTISSA POSITIVE
4617   BF9D  BD B9 B4                 JSR   LB9B4                   ADD .5 (PI) TO FPA0
4618   BFA0  96 54                    LDA   FP0SGN                  GET SIGN OF FPA0
4619   BFA2  2B 05                    BMI   LBFA9                   BRANCH IF NEGATIVE
4620   BFA4  03 0A                    COM   RELFLG                  COM IF +(3*PI)/2 >= ARGUMENT >+ PI/2 (QUADRANT FLAG)
4621   BFA6  BD BE E9          LBFA6  JSR   LBEE9                   TOGGLE MANTISSA SIGN OF FPA0
4622   BFA9  8E BF C2          LBFA9  LDX   #LBFC2                  POINT X TO FP (.25)
4623   BFAC  BD B9 C2                 JSR   LB9C2                   ADD .25 (PI/2) TO FPA0
4624   BFAF  35 02                    PULS  A                       GET OLD MANTISSA SIGN
4625   BFB1  4D                       TSTA                          * BRANCH IF OLD
4626   BFB2  2A 03                    BPL   LBFB7                   * SIGN WAS POSITIVE
4627   BFB4  BD BE E9                 JSR   LBEE9                   TOGGLE MANTISSA SIGN
4628   BFB7  8E BF C7          LBFB7  LDX   #LBFC7                  POINT X TO TABLE OF COEFFICIENTS
```

```
4629    BFBA   7E BE F0                    JMP    LBEF0                    GO CALCULATE POLYNOMIAL VALUE
4630
4631    BFBD   83 49 0F DA A2     LBFBD     FCB    $83,$49,$0F,$DA,$A2      6.28318531 (2*PI)
4632    BFC2   7F 00 00 00 00     LBFC2     FCB    $7F,$00,$00,$00,$00      .25
4633
4634                              * MODIFIED TAYLOR SERIES SIN COEFFICIENTS
4635    BFC7   05                 LBFC7     FCB    6-1                      SIX COEFFICIENTS
4636    BFC8   84 E6 1A 2D 1B     LBFC8     FCB    $84,$E6,$1A,$2D,$1B      * -((2*PI)**11)/11!
4637    BFCD   86 28 07 FB F8     LBFC8     FCB    $86,$28,$07,$FB,$F8      *  ((2*PI)**9)/9!
4638    BFD2   87 99 68 89 01     LBFD2     FCB    $87,$99,$68,$89,$01      * -((2*PI)**7)/7!
4639    BFD7   87 23 35 DF E1     LBFD7     FCB    $87,$23,$35,$DF,$E1      *  ((2*PI)**5)/5!
4640    BFDC   86 A5 5D E7 28     LBFDC     FCB    $86,$A5,$5D,$E7,$28      * -((2*PI)**3)/3!
4641    BFE1   83 49 0F DA A2     LBFE1     FCB    $83,$49,$0F,$DA,$A2      *    2*PI
4642
4643    BFE6   A1 54 46 8F 13 8F  LBFE6     FCB    $A1,$54,$46,$8F,$13      UNUSED GARBAGE BYTES
4644    BFEC   52 43 89 CD                 FCB    $8F,$52,$43,$89,$CD      UNUSED GARBAGE BYTES
4645
4646                              * INTERRUPT VECTORS
4647    BFF0   A6 81              LBFF0     FDB    LA681                    RESERVED
4648    BFF2   01 00              LBFF2     FDB    SW3VEC                   SWI3
4649    BFF4   01 03              LBFF4     FDB    SW2VEC                   SWI2
4650    BFF6   01 0F              LBFF6     FDB    FRQVEC                   FIRQ
4651    BFF8   01 0C              LBFF8     FDB    IRQVEC                   IRQ
4652    BFFA   01 06              LBFFA     FDB    SWIVEC                   SWI
4653    BFFC   01 09              LBFFC     FDB    NMIVEC                   NMI
4654    BFFE   A0 27              LBFFE     FDB    RESVEC                   RESET
```

MODIFIED REGISTERS	ADDRESS	DESCRIPTION
ALL	RESVEC (AØ27)	RESET ENTRY POINT - set SAM, PIAs ram check - if RSTFLG = $55, then jump to RSTVEC; otherwise do a cold start.
ALL	BACDST (AØ74)	COLD START ENTRY POINT - clear 1st 1K of RAM, reset BASIC's pointers.
ALL	BAWMST (AØE8)	WARM START ENTRY POINT - reset some of BASIC's pointers.
A	A171	ASCII CONSOLE IN - read a character from CONSOLE IN. Mask off bit 7 and return character in ACCA.
A	A176	CONSOLE IN - read a character from active input device (DEVNUM). Return character in ACCA.
A,B,X	A199	CURSOR DRIVER - put a cursor block on the screen at the address in CURPOS.
A	A1B1	WAIT FOR A KEYSTROKE - blink the cursor while waiting for a keystroke. Return the ASCII value of the key in ACCA when a key is depressed.
A	KEYIN (A1CB)	SCAN THE KEYBOARD FOR A KEY DEPRESSION - Return zero flag = 1 if no new key down. Return the ASCII value of the key in ACCA if a new key is depressed.
NONE	PUTCHR (A282)	CONSOLE OUT - sends character in ACCA to output device. The output device is specified in DEVNUM.
NONE	A2BF	RS232 OUTPUT DRIVER - software UART specifically formatted to drive a line printer. The routine may be used to drive other devices, and has been modified by all revisions to BASIC. Version 1.2 will not begin transmitting data until the destination device is ready.
NONE	A30A	PUT A CHARACTER ON THE SCREEN - place a character on the screen (the screen starts at $4ØØ) at the location stored in CURPOS.
NONE	A35F	INITIALIZE PRINT PARAMETERS - set up tab field width, tab zone, current position and line width according to the device selected (DEVNUM). This routine will vector into RAM at RVEC2.
A,B,X	A38D	GET A BASIC INPUT LINE - this routine will allow the

		inputting of a BASIC input line from CONSOLE IN.
A	A3ED	INPUT DEVICE NUMBER CHECK - check for a valid input device number and file mode. This routine will vector into RAM at RVEC5.
A	A4Ø6	OUTPUT DEVICE NUMBER CHECK - check for a valid output device number and file mode. This routine will vector into RAM at RVEC6.
A	A42D	CLOSE A FILE - closes the file specified by DEVNUM. Vectors into RAM at RVEC8.
A	A549	BREAK CHECK - check to see if the break key or the pause (shift @) key is down. Vectors into RAM at RVEC11.
A,B,X	A7Ø1	START TAPE, READ A BLOCK - load a block of data from the cassette tape into RAM and return error status in ACCB.
A,B,X	GETBLK (A7ØB)	READ A BLOCK - load a block of data from the cassette tape into RAM.
A,B,X	CASON (A77C)	SYNC THE TAPE DECK - turn on the tape and wait for sync bytes.
X	A7D1	LONG DELAY - approximately 1/2 second delay. Loads X register with zero and counts it down to zero.
A,B,X,Y	WRLDR (A7D8)	WRITE LEADER - write a leader to the cassette tape.
A,B,X,Y	SNDBLK (A7F4)	WRITE BLOCK TO CASSETTE - take a block of RAM and write it to cassette.
A,B,X,Y	A82A	WRITE A BYTE TO TAPE - write ACCA to tape. This routine does the "dirty work" of actually writing a byte to tape.
B,X	A928	CLEAR SCREEN - store blanks ($6Ø) to the video display screen.
A	A974	DISABLE ANALOG MULTIPLEXER - disable the sound analog multiplexer. This will not allow any sound input to pass through to the analog multiplexer.
A	A976	ENABLE ANALOG MULTIPLEXER - enable the sound analog multiplexer. This will allow sound inputs to pass through the analog multiplexer.

A,U	A9A2	SET ANALOG MULTIPLEXER - set the control inputs to the analog multiplexer to allow one of the four inputs to pass through.
A,B,X,U	GETJOY (A9DE)	READ JOYSTICKS - software 6-bit analog to digital conversion routine used to read the joystick potentiometers.
A,B	AC33	FREE RAM CHECK - check to see if there is room to store 2*ACCB bytes in free RAM, OM error if not.
A,B,X,U	AD01	LINE NUMBER SEARCH - search the BASIC program for the line number stored in BINVAL. Set the carry flag if no match.
X	AD26	ERASE VARIABLES - erase BASIC's variables and reset pointers.
A,B,X	AF67	CONVERT LINE NUMBER TO BINARY - convert an ASCII line number in a BASIC program to binary and return the value in BINVAL.
A,B,X,Y	AFA4	PUT STRING IN STRING SPACE - move a string whose descriptor is located at FPA0+2 into the string space.
NONE	B143	NUMERIC TYPE MODE CHECK - test the contents of VALTYP and return if positive. TM error if negative.
NONE	B146	STRING TYPE MODE CHECK - test the contents of VALTYP and return if negative. TM error if positive.
ALL	B156	EVALUATE EXPRESSION - evaluate an expression in a BASIC statement. BASIC's input pointer must be pointed to the expression.
ALL	B357	EVALUATE VARIABLE - evaluate the variable to which BASIC's input pointer is pointing. Return with X and VARPTR pointing to the variable descriptor. If the variable is not stored in the variable table, that variable name with a value of zero is inserted into the variable table.
A	B3A2	SET CARRY IF NOT ALPHA - set the carry flag if ACCA is not and ASCII alpha character.
A,B,X	INTCNV (B3ED)	CONVERT FPA0 TO INTEGER. Convert FPA0 to a signed 2-byte integer; return the value in ACCD.
A,B,X	GIVABF (B4F4)	CONVERT INTEGER TO FLOATING POINT - convert the value in ACCD into a floating point number in FPA0.

A,B,X	B54C	PUT DESCRIPTOR ON STRING STACK - put the direct page descriptor buffer data (STRDES) onto the string stack. Set the variable type (VALTYP) to string type.
A,B,X	B56D	RESERVE ACCB BYTES IN STRING SPACE - reserve ACCB bytes in the string storage space. Return with the starting address of the reserved string space in X and FRESPC.
A,B,X	B74Ø	INTEGER SIZE CHECK - check FPAØ to make sure it is in the range -32768 <= FPAØ <= 32767. If it is, return the value of that integer in X.
ALL	B7C2	UNCRUNCH - uncrunch a basic line into BASIC's line input buffer. Vectors into RAM at RVEC24.
ALL	B821	CRUNCH - crunch the line that the input pointer is pointing to into the line input buffer and return the length of the crunched line in ACCD.
A,B,X,U	B99C	SEND STRING TO CONSOLE OUT - parse a string which is pointed to by X and send it to CONSOLE OUT.
A,B,X	BD12	CONVERT STRING TO FLOATING POINT - convert an ASCII string pointed to by BASIC's input pointer into a floating point value in FPAØ. Vectors into RAM at RVEC19.
A,B,X,U	BDCC	PRINT DECIMAL NUMBER TO CONSOLE OUT - convert the value in ACCD into a decimal number and send it to CONSOLE OUT.
A,B,X,U	BDD9	FLOATING POINT TO ASCII STRING - convert the floating point number in FPAØ into an ASCII string in the string buffer.
A,B,X	BEFF	EXPAND THE POLYNOMIAL - calculate the value of an expanded polynomial expression. Enter with X pointing to a table of coefficients the first byte of which is the number of (coefficients -1) followed by that number of packed floating point numbers. The polynomial is evaluated as follows: value = (((FPAØ*YØ+Y1)*FPAØ+Y2)* FPAØ +YN)

ADDRESS	DESCRIPTION
B9B4	ADD .5 TO FPAØ
B9B9	SUBTRACT FPAØ FROM FLOATING POINT NUMBER POINTED TO BY X, LEAVE RESULT IN FPAØ
B9BC	ARITHMETIC OPERATION (-) JUMPS HERE - SUBTRACT FPAØ FROM FPA1 (ENTER WITH EXPONENT OF FPAØ IN ACCB)
B9C2	ADD FLOATING POINT NUMBER POINTED TO BY X TO FPAØ - LEAVE RESULTS IN FPAØ
B9C5	ARITHMETIC OPERATION (+) JUMPS HERE - ADD FPAØ TO FPA1 (ENTER WITH EXPONENT OF FPAØ IN ACCB AND EXPONENT OF FPA1 IN ACCA)
BA1C	NORMALIZE FPAØ
BA79	NEGATE FPAØ MANTISSA
BA83	ADD ONE TO FPAØ MANTISSA
BACA	ARITHMETIC OPERATION (*) JUMPS HERE - MULTIPLY FPAØ BY X - RETURN PRODUCT IN FPAØ
BADØ	MULTIPLY FPAØ MANTISSA BY FPA1, NORMALIZE HIGH ORDER BYTES OF PRODUCT IN FPAØ. THE LOW ORDER FOUR BYTES OF THE PRODUCT WILL BE STORED IN VAB-VAE
BB2F	UNPACK A FLOATING POINT NUMBER FROM X INTO FPA1
BB6A	FAST MULTIPLY FPAØ BY 1Ø AND LEAVE RESULT IN FPAØ
BB82	DIVIDE FPAØ BY 1Ø
BB8F	DIVIDE X BY FPAØ - LEAVE NORMALIZED QUOTIENT IN FPAØ
BB91	DIVIDE FPA1 BY FPAØ. ENTER WITH EXPONENT OF FPA1 IN ACCA AND FLAGS SET BY TSTA
BBA4	COMPARE FPAØ MANTISSA TO FPA1 MANTISSA - SET CARRY FLAG IF FPA1 >= FPAØ
BCØB	COPY MANTISSA FROM FPA2 TO FPAØ
BC14	COPY A PACKED FLOATING POINT NUMBER FROM X TO FPAØ
BC2A	PACK FPAØ AND SAVE IT IN FPA4

BC2F	PACK FPAØ AND SAVE IT IN FPA3
BC33	PACK FPAØ AND SAVE IT IN ADDRESS STORED IN VARDES
BC35	PACK FPAØ AND SAVE IT IN ADDRESS POINTED TO BY X
BC4A	MOVE FPA1 TO FPAØ RETURN WITH MANTISSA SIGN IN ACCA
BC5F	TRANSFER FPAØ TO FPA1
BC6D	CHECK FPAØ; RETURN ACCB = Ø IF FPAØ = Ø, ACCB = $FF IF FPAØ = NEGATIVE, ACCB = 1 IF FPAØ = POSITIVE
BC7C	CONVERT A SIGNED NUMBER IN ACCB INTO A FLOTING POINT NUMBER
BC96	COMPARE A PACKED FLOATING POINT NUMBER POINTED TO BY X TO AN UNPACKED FLOATING POINT NUMBER IN FPAØ. RETURN ZERO FLAG SET AND ACCB=Ø IF EQUAL; ACCB = 1 IF FPAØ > (X); ACCB = $FF IF FPAØ < (X)
BDØ9	FILL MANTISSA OF FPAØ WITH CONTENTS OF ACCB

START	END	DESCRIPTION
A000	A00D	INDIRECT JUMP TABLE
A10D	A128	DIRECT PAGE ROM IMAGE
A129	A146	PAGE ONE ROM IMAGE
A147	A170	COPYRIGHT MESSAGES
A26E	A281	SPECIAL KEY LOOKUP TABLE
A85C	A87F	SINE WAVE LOOKUP TABLE
AA29	AA50	SECONDARY DISPATCH TABLE
AA51	AA65	OPERATOR PRECEDENCE TABLE
AA66	AB19	PRIMARY RESERVED WORD TABLE
AB1A	AB66	SECONDARY RESERVED WORD TABLE
AB67	ABAE	PRIMARY DISPATCH TABLE
ABAF	ABE0	ERROR MESSAGES
ABE1	ABEC	"ERROR IN" MESSAGE
ABED	ABF1	"OK" MESSAGE
ABF2	ABF8	"BREAK" MESSAGE
AFCF	AFD5	"?REDO" MESSAGE
B0E8	B0F7	"?EXTRA IGNORED" MESSAGE
B3DF	B3E3	FLOATING POINT NUMBER -32768
BAC5	BAC9	FLOATING POINT NUMBER 1.0
BB7D	BB81	FLOATING POINT NUMBER 10
BDB6	BDBA	FLOATING POINT NUMBER 99999999.9
BDBB	BDBF	FLOATING POINT NUMBER 999999999
BDC0	BDC4	FLOATING POINT NUMBER 1E+9
BEC0	BEC4	FLOATING POINT NUMBER .5

BEC5	BEE8	TABLE OF MANTISSAS OF UNNORMALIZED POWERS OF TEN
BF74	BF77	CONSTANT RANDOM NUMBER SEED
BFBD	BFC1	FLOATING POINT NUMBER 2*PI
BFC2	BFC6	FLOATING POINT NUMBER .25
BFC7	BFE5	MODIFIED TAYLOR SERIES SINE COEFFICIENTS
BFE6	BFEF	TWO GARBAGE FLOATING POINT NUMBERS
BFFØ	BFFF	INTERRUPT VECTORS

MEMORY MAP

One of the most important tools to have at your fingertips, if you are going to attempt to use any machine's built-in operating system, is a complete and accurate map of that system's memory structure. At the beginning of the BASIC disassembly listing you will find the most complete memory map available for the Color Computer outside of Microsoft’s domain. It explains all of the variables in the direct page and what their functions are, defines all of the variables and buffers between the direct page and the video display RAM, and all of the variables that are used by Disk RAM. It identifies the areas in memory used by the variable tables, the array tables, the string space, cleared memory, and other important areas.

The direct page provides the most useful source of rapidly accessed variable space available in the 6809. When you become familiar with 6809 Machine Language programming you will notice that it is quicker byte wise and time wise to access variables which are located in the direct page. The Color Computer, of course, keeps the direct page in page 0, which makes it relatively compatible with 6502 programming. The direct page is also at the very bottom of RAM, where it is conveniently out of the way of any programs written by the user for whatever purpose he has in mind. If you look at the memory map at the beginning of Color BASIC Unravelled you will find that some of the variables will have asterisks designation in front of them. If that designation is PV, it defines a Permanent Variable. Permanent Variable has been chosen for lack of a better word; exactly what this means is that the variable has a defined function that is used by every command that BASIC has. Such a variable would be the beginning of BASIC, the top of free RAM, the beginning of array variables, the beginning of normal variables, the top of the string stock, the present pointer to the current value in the string stock and so forth. These variables will cause permanent harm to a BASIC program if they are modified during the course of your machine language program; therefore, you will have to be very careful how you use these variables. If you change the value, for instance, of the start of BASIC, you will cause BASIC to feel that there are fewer or more lines in the program than there actually are, which could easily result in an error or crash. Therefore, when you are making a program, which is designed to run and mesh with BASIC, you should not make any changes to a PV type variable, unless you are absolutely sure that you know what you are doing. Obviously, there can be some instances when you will want to change the start of BASIC. If there is some value you might want to change, you will have to make your decision based on what you're doing as to whether you want to change the variable or not, just be aware that PV variables are very tricky to change and changing them may blow up BASIC.

Other variables are designated TV, which is a temporary variable, and is a variable whose function should be uniform for all BASIC commands. A temporary variable has one specific use and one specific BASIC command. A perfect example would be the variable labeled DIMFLG This variable is used when defining a dimensioned array in order to specifically tell a certain routine in BASIC that the variable currently being defined is an array variable as opposed to a string variable or single precision variable. Once the variable is defined, obviously the value being stored in DIMFLG has no appropriate use for any other BASIC command. Therefore it may be modified if required for your own use, but you should be aware that the DIMFLG variable can be changed during the course of a BASIC program and that if you use it for some specific value in your program you may not harm BASIC, but don't expect the value to be unmodified by BASIC in the course of the normal operation of BASIC.

The variables not labeled either TV nor PV will be used by many different routines in BASIC and are neither Temporary nor Permanent, because they are used by so many routines that it doesn't matter what happens to these variables. These variables have a particular function, such as a pointer, an address counter or normal counter. They have been given a specific label because they do have a particular function that remains common from BASIC command to BASIC command. There are also variables, which are referred to as Scratch Pad variables. These variables have the designation VXX, where the XX is the actual Hex address of that variable in RAM. These are different from the temporary variables, in that they may be used for any particular function by any particular BASIC command; therefore they could be pointers, counters or addresses or any other kind of temporary storage. These are the most useful to use as temporary storage for your own routines, since if you modify these routines between various BASIC commands (not in a BASIC command but between BASIC commands) it will not cause any harm whatsoever to the operation of a BASIC program.

Other variables may be identified with a DV designation, which may be in conjunction with a TV or PV or may be by itself. This particular designation is used to define a double variable; that is, there are places in BASIC where a variable is loaded into a 16-byte register even though the variable is an 8-byte quantity. As such, the variable and the variable immediately following the DV designation may not be separated from each other; they must be immediately adjacent to one another in the memory space of the computer. If for any reason these variables are separated from one another in the memory space of the computer, the instructions in BASIC, which grab data from the double variable, will not function properly. There are some variables in Extended BASIC, which are two 16-byte quantities, which must be kept next to each other, such as HORBEG or VERBEG. This is necessary because an index register is pointed at the first of these variables, then it is incremented to the appropriate variable.

AREA BETWEEN THE DIRECT PAGE AND VIDEO DISPLAY

The area between the direct page and the video display at $400 is used by several different routines to store pieces of useful information. There are some large buffers and some small 2 or 3 byte value storage blocks, which are used by some specific routines and some that are in general used by many different routines. The interrupt jump vectors are stored from address $100 to $111. These contain the addresses where the interrupt factors jump to IRQ, FIRQ, NMI, RESET, and Software Interrupt routines. Immediately following the interrupt vectors are several small variables, which are used by different routines. The first of these is the USRJMP variable. These are three bytes, which Color BASIC uses to store the jump address for the USR function. When Extended BASIC is in the machine, these locations are not used by the USR function any more. They are instead used to store the timer value (TIMVAL). Timer value is only two bytes and Extended or Disk BASIC does not use the third byte of this three-byte block of data. The next variable stored is RVSEED, a five-byte value for the variable Random Number Seed. Following that is CASFLG, the Case Flag, which determines whether the characters being put on the screen are in upper or lower case. If the value stored here is zero you are in lower case. If it is $FF they are in upper case. Then comes DEBVAL, which is the keyboard debounce delay value, a two-byte quantity. Following that is EXPJMP, which is the address that the EXEC command uses to jump to. Next are the command interpretation tables. There are normally only as many interpretation tables as there are ROMs plugged into the computer. If extended BASIC and not disk BASIC is plugged in, in the area following COMVEC are the USR function jump vectors for Extended BASIC. If Disk BASIC is plugged in, the USR jump vectors are transferred to the disk RAM. After the command interpretation tables is the keyboard buffer. This is the memory that is used so that there can be rollover in the keyboard

routine. Eight bytes are used to store the information on which keys have and have not been pressed. Following that are four bytes (POTVAL) to store the values of the joystick potentiometers. After POTVAL come BASIC's RAM vectors. An explanation of these vectors is provided in the memory map at the beginning of this book and the user should refer to that explanation in order to get a detailed description of how the RAM vectors function. Following the RAM hooks is a 40-byte block of data used to store string descriptors. This is the string stack, which is used in string manipulation functions. After the string stack comes the cassette file name buffer where the cassette file name is stored prior to searching for a cassette file. After that is a 256 byte block of data which is the cassette I/O buffer. Following that is a two-byte sub-block to the line input buffer, which is called the line input header. This is used to store the jump address of the next BASIC line. After line header comes LINBUF which is a 251-byte buffer to store BASIC input line as it is being typed in. This 251-byte area is also used for several different functions but primarily it is used as a line input buffer. The last block of data is a 41-byte block of data following LINBUF up to the video display RAM which is called STRBUF. This is a string buffer, which is used to hold temporary string information and temporary strings before they are moved into the string space. It is most commonly used in floating point to ASCII string and ASCII string to floating point data conversions. Then from $400 to $5FF is the 512-byte block of video display RAM. If you have a disk hooked up to your system, the area from $600 to $989 is used by the disk for its own special I/O buffers and disk variables.

INTERRUPTS

BASIC uses the 6809 interrupt structure to control those commands, which require precise timing intervals. The manner in which the interrupt signals are handled by the 6809 and 6821 (or 6822) in the Color Computer is described in the FACTS book and will not be covered in this book. Only the software aspects will be covered.

IRQ — The 6821 Peripheral Interface, Adapter (PIA) may be programmed to pass either a 16.67 ms (60HZ) or a 63.5 microsecond input to the 6809's IRQ interrupt pin. Color BASIC uses only the 60HZ interrupt; the 63.5 microsecond input is never enabled. The IRQ routine is used to increment or decrement the following parameters:

COMMAND	VARIABLE	IRQ FUNCTION
SOUND	SNDDUR	Decrement SNDDUR
PLAY	PLYTMR	Decrement VD5 from PLYTMR
TIMER	TIMVAL	Increment TIMVAL
(DOS BASIC)	RDYTMR	Decrement RDYTMR

The SOUND and PLAY commands will fall into an endless timing loop which will only be terminated by the IRQ routine's decrementing SNDDUR and/or PLYTMR to zero.

FIRQ — The PIAs may be programmed to pass either the RS 232 status input or the cartridge interrupt signal to the 6809's FIRQ interrupt pin. Color BASIC uses FIRQ to vector control to a ROM-PAK if pins 7 & 8 of the cartridge port are connected together by a cartridge.

NMI — The 6809's NMI pin is connected only to the cartridge port. The Disk Operating System (DOS) uses the NMI to vector out of data transfers between the 6809 and the 1793 Floppy Disk Controller.

SWI — Not used by Color BASIC

SWI2 — Not used by Color BASIC

SWI3 — The DOS command of Disk BASIC calls SWI3. The user must provide a SWI3 servicing routine – Disk BASIC does not provide one.

EXPRESSIONS AND OPERATORS

Relational Operators		Arithmetic Operators	
=	Equal	+	Add
<	Less Than	-	Subtract
>	Greater Than	*	Multiply
<=	Less Than or Equal	/	Divide
>=	Greater Than or Equal	^	Exponentiation
<>	Not Equal	-	Negation

Boolean Operators	String Operators	
AND	+	Concatenation
OR		
NOT		

Rules for Evaluating Expressions

1. Operations of higher precedence are performed before operations of lower precedence. This means the multiplications and divisions are performed before additions and subtractions. As an example, 2+10/5 equals 4, not 2.4. When operations of equal precedence are found in a formula, the left-hand one is executed first: 6-3+5 = 8, not -2.
2. The order in which operations are performed can always be specified explicitly through the use of parentheses. For instance, to add 5 to 3 and then divide that by 4, we would use (5+3)/4, which equals 2. If, instead, we had used 5+3/4, we would get 5.75 as a result (5 plus 3/4).

The precedence of operators used in evaluating expressions is as follows, in order beginning with the highest precedence: (Note: Operators listed on the same line have the same precedence).

1)	FORMULAS ENCLOSED IN PARENTHESIS ARE ALWAYS EVALUATED FIRST.	
2)	^	EXPONENTATION
3)	NEGATION	-X WHERE X MAY BE A FORMULA
4)	*/	MULTIPLICATION AND DIVISION
5)	+-	ADDITION AND SUBTRACTION
6)	RELATIONAL OPERATORS	= EQUAL
		<> NOT EQUAL
	(EQUAL PRECEDENCE FOR	< LESS THAN
	ALL SIX).	> GREATER THAN
		<= LESS THAN OR EQUAL
		>= GREATER THAN OR EQUAL
7)	NOT	LOGICAL AND BITWISE “NOT” LIKE NEGATION, NOT TAKES ONLY THE FORMULA TO ITS RIGHT AS AN ARGUMENT
8)	AND	LOGICAL AND BITWISE “AND”
9)	OR	LOGICAL AND BITWISE “OR”

The ASCII table is defined in Appendix J. It contains the order in which characters within the Color Computer are represented when two strings are compared. Characters within a set of strings are compared starting at the leftmost character to the end of the field specified.

Using the ASCII table, we can compare a string containing an "A" to one containing a "B" in the same position. The result is that the second string is greater than the first.

A string containing a blank is less than a "1", which is less than an "A", which is less than a "B". The string "A" is less than the string "ABC" or any string containing "A" as the first character. All characters are compared in sequence with the first unequal character defining the relationship between the strings. Thus, the same relational functions may be used for both strings and numbers.

Listed below are the differences between Color BASIC Version 1.0 and Version 1.2

CHANGE	ADDRESS	Version 1.0	Version 1.2
A	$A001	$C1	$CB
B	$A01B-$A0C7	SEE LISTING 1	
C	$A102-$A104	LDU #$A108	LEAY $A108,PCR
D	$A114	$57	$58
E	$A155	$30	$32
F	$A15E	$30	$32
G	$A1B5-$A26D	SEE LISTING 2	
H	$A2C3-$A2FA	SEE LISTING 3	
I	$A440	$08	$03
J	$A56A	$C1	$CB
K	$A6EB	$07	$14
L	$ADFD	$C1	$CB
M	$B23F	$E8	$E3
N	$B38E	$72	$75
P	$B3ED-$B427	SEE LISTING 4	
Q	$B9D6	$2B BMI	$25 BCS

Change A is a branch length change caused by the keyboard driver mod.

Change B (Listing 1) is a major rework of the warm and cold start initializations required to allow the computer to accept 64K dynamic RAMs.

Change C was required because change B changed the storage location of the RESET jump vector from the U register to the Y register.

Change D modified the line printer baud rate.

Change E is the version number.

Change F is the copyright year.

Change G is a major change (Listing 2) to the keyboard driver allowing a quick scan of the keyboard if no keys are down and filtering of joystick button depressions out of the keyboard scan.

Change H is also a major change (Listing 3) which allows the line printer driver to output an eight bit character and will not allow any transmission until the receiving device is ready.

Change I causes an end of program block to be written to a cassette file if the buffer is empty when the file is closed.

Change J speeds up the INKEY$ command by not entering the keyboard scan routine if no keys are depressed.

Change K is a minor change which causes the upper left hand corner of the screen to blink during cassette loading operations.

Change L speeds up the BREAK key check routine by not entering the

keyboard scan routine if no more keys are depressed.

Change M is a minor change which slightly speeds up the expression evaluation routine.

Change N is merely a different length branch caused by change P.

Change P (Listing 4) causes a numeric variable type check to be done before the INTCNV (FPAØ to integer) routine.

Change Q fixed a minor bug in the floating point addition routine.

*** LISTING 1

```
A01B                BNE   BACDST       NO - DO A COLD START
A01D                LDX   RSTVEC       WARM START VECTOR
A01F                LDA   ,X           GET FIRST BYTE OF WARM START ADDR
A021                CMPA  #$12         IS IT A NOP?
A023                BNE   BACDST       NO - DO A COLD START
A025                JMP   ,X           YES, GO THERE
                *
A027            RESVEC  LDU   #LA00E       BASIC WARM START ENTRY (RESET)
A02A            LA02A   CLRB               *
A02B                TFR   B,DP             * USE PAGE Ø AS DIRECT PAGE
                ********************
A02D                LDX   #PIA0        POINT X TO PIAØ
A030                CLR   1,X          CLEAR CONTROL REGISTER A ON PIAØ(U8)
A032                CLR   3,X          CLEAR CONTROL REGISTER B
A034                CLR   ,X           A SIDE IS INPUT
A036                LDD   #$FF34
A039                STA   2,X          B SIDE IS OUTPUT
A03B                STB   1,X          ENABLE PERIPHERAL REGISTERS
A03D                STB   3,X          AND CA2, CB2 AS OUTPUTS
                ********************
A03F                LDX   #PIA1        POINT X TO PIA1
A042                CLR   1,X          * CLEAR CONTROL REGISTER A ON PIA1(U4)
A044                CLR   3,X          * CLEAR CONTROL REGISTER B
A046                DECA               A - REG NOW HAS $FE
A047                STA   ,X           = BITS 1-7 ARE OUTPUTS, BIT Ø IS INPUT
                *                      = ON SIDE A
A049                LDA   #$F8         *
A04B                STA   2,X          * BITS Ø-2 ARE INPUTS, BITS 3-7 ARE
                *                      * OUTPUTS ON B SIDE
A04D                STB   1,X          ENABLE PERIPHERAL REGISTERS
A04F                STB   3,X          AND CA2, CB2 AS OUTPUTS
A051                CLR   2,X          ZEROS TO 6847
A053                LDA   #2           *
A055                STA   ,X           * MAKE SERIAL OUTPUT MARKING
A057                LDA   2,X          READ PORT B OF U4 (TO GET RAM SIZE)
A059                LDX   #SAMREG      SAM CONTROL REGISTER ADDR
A05C                LDB   #16          16 SAM CONTROL REGISTER BITS
A05E            LA05E   STA   ,X++         ZERO OUT SAM CONTROL REGISTER (CLEAR BITS)
A060                DECB               DECREMENT REGISTER COUNTER
A061                BNE   LA05E        BRANCH IF NOT DONE
A063                STA   SAMREG+9     SET DISPLAY PAGE AT $400
A066                ANDA  #4           MASK OFF ALL BUT RAM SIZE BIT
A068                BEQ   LA06C        BRANCH IF 4K RAM
A06A                STA   -5,X         SET FOR 16K DYNAMIC
```

```
A06C            LA06C   JMP   ,U            GO DO A WARM START
A06E            BACDST  LDX   #0            POINT X TO TOP OF DIRECT PAGE
A071            LA071   CLR   ,X+           CLEAR FIRST 1K OF RAM
A073                    CMPX  #VIDRAM       COMPARE TO TOP OF DISPLAY (1K)
A076                    BNE   LA071         BRANCH IF NOT DONE
A078                    JSR   LA928         CLEAR SCREEN
A07B                    LDX   #LA10D        POINT X TO ROM IMAGE OF DIRECT PAGE VARS
A07E                    LDU   #CMPMID       POINT U TO RAM DESTINATION
A081                    LDB   #28           28 BYTES
A083                    JSR   LA59A         MOVE (B) BYTES FROM (X) TO (U)
A086                    LDU   #IRQVEC       POINT U TO NON-DIRECT PAGE VARIABLES
A089                    LDB   #30           30 BYTES
A08B                    JSR   LA59A         MOVE (B) BYTES FROM (X) TO (U)
A08E                    LDX   #LB277        ADDR OF SYNTAX ERROR ROUTINE
A091                    STX   3,U           * SET EXBAS PRIMARY AND SECONDARY
                *                           * COMMAND INTERPRETATION TABLES TO
A093                    STX   8,U           * SYNTAX ERROR (U POINTS TO $12A AT
                *                           * THIS POINT)
A095                    LDX   #RVEC0        POINT X TO RAM VECTORS
A098                    LDA   #$39          OP CODE OF RTS
A09A            L409A   STA   ,X+           PUT RTS'S IN THE RAM VECTORS
A09C                    CMPX  #RVEC+25*3    END OF RAM VECTORS?
A09F                    BNE   LA09A         NO KEEP INSERTING RTS
A0A1                    STA   LINHDR-1      PUT RTS IN $2D9
A0A4                    LDX   VIDRAM+$200   POINT TO COLOR BASIC'S START OF PROGRAM
A0A7                    CLR   ,X+           PUT A ZERO AT THE START OF BASIC
A0A9                    STX   TXTTAB        BEGINNING OF BASIC PROGRAM
A0AB            LA0AB   LDA   2,X           LOOK FOR END OF PROGRAM
A0AD                    COMA
A0AE                    STA   2,X           STORE IN RAM
A0B0                    CMPA  2,X           IS VALUE IN MEMORY THE SAME AS WHAT WAS
                *                           JUST PUT THERE?
A0B2                    BNE   LA0BA         IF NOT, THEN IT IS NOT RAM OR THE RAM IS
                *                           BAD
A0B4                    LEAX  1,X           MOVE TO NEXT RAM LOCATION
A0B6            LA0B6   COM   1,X           RESTORE VALUE OF MEMORY JUST CHANGED
A0B8                    BRA   LA0AB         KEEP CHECKING RAM
A0BA            LA0BA   STX   TOPRAM        SET TOP OF RAM POINTER
A0BC                    STX   MEMSIZ        TOP OF STRING SPACE
A0BE                    STX   STRTAB        START OF STRING VARIABLES
A0C0                    LEAX  -200,X        * CLEAR 200 BYTES ON A COLD START -
A0C4                    STX   FRETOP        * SAVE NEW TOP OF FREE RAM
A0C6                    TFR   X,S           PUT STACK THERE (AT MEMEND-200)

*** LISTING 2
A1B5                    BSR   KEYIN         GO CHECK KEYBOARD
A1B7                    BEQ   LA1B3         LOOP IF NO KEY DOWN
A1B9                    LDB   #$60          BLANK
A1BB                    STB   [CURPOS]      BLANK CURRENT CURSOR CHAR ON SCREEN
A1BF            LA1BF   PULS  B,X,PC
                *
                * THIS ROUTINE GETS A KEYSTROKE FROM THE KEYBOARD IF A KEY
                * IS DOWN. IT RETURNS ZERO TRUE IF THERE WAS NO KEY DOWN.
                *
A1C1            KEYIN   PSHS  B,X           SAVE REGISTERS
A1C3                    BSR   LA1C8         GET KEYSTROKE
A1C5                    TSTA                SET FLAGS
```

```
A1C6                    PULS  B,X,PC       RESTORE REGISTERS
A1C8           LA1C8    LEAS  -3,S         ALLOCATE 3 STORAGE BYTES ON STACK
A1CA                    LDX   #KEYBUF      SET X TO KEYBOARD MEMORY BUFFER
A1CD                    CLR   ,S           RESET COLUMN COUNTER
A1CF                    LDB   #$FE         COLUMN STROBE DATA, CHECK BIT Ø FIRST
                ** A COLUMN IS BEING CHECKED IF THE
                ** CORRESPONDING BIT IN THE COLUMN STROBE
                ** REGISTER ($FFØ2) HAS A ZERO IN IT.
A1D1                    STB   PIAØ+2       STORE IN COLUMN STROBE REGISTER
A1D4           LA1D4    BSR   LA238        GET KEY DATA
A1D6                    STA   1,S          TEMP STORE KEY DATA
A1D8                    EORA  ,X           COMPARE WITH KEY MEMORY DATA
A1DA                    ANDA  ,X           ACCA=Ø IF THIS KEY WAS DOWN LAST TIME, TOO
A1DC                    LDB   1,S          GET NEW KEY DATA
A1DE                    STB   ,X+          STORE IT IN KEY MEMORY
A1EØ                    TSTA               WAS A NEW KEY DOWN?
A1E1                    BNE   LA1ED        YES
A1E3                    INC   ,S           NO, INCREMENT COLUMN COUNTER
A1E5                    COMB               SET CARRY FLAG
A1E6                    ROL   PIAØ+2       ROTATE COLUMN STROBE DATA LEFT ONE BIT
A1E9                    BCS   LA1D4        ALL COLUMNS CHECKED WHEN ZERO IN THE
                * COLUMN STROBE DATA IS ROTATED INTO THE
                * CARRY FLAG
A1EB                    PULS  B,X,PC       RESTORE REGISTERS

A1ED           LA1ED    LDB   PIAØ+2       GET COLUMN STROBE DATA

                ****************************************************************
                ** THIS ROUTINE CONVERTS THE KEY DEPRESSION INTO A NUMBER
                ** FROM Ø-5Ø IN ACCB CORRESPONDING TO THE KEY THAT WAS DOWN
A1FØ                    STB   2,S          TEMP STORE IT
A1F2                    LDB   #$F8         TO MAKE SURE ACCB=Ø AFTER FIRST ADDB #8
A1F4           LA1F4    ADDB  #8           ADD 8 FOR EACH ROW OF KEYBOARD
A1F6                    LSRA               ACCA CONTAINS THE ROW NUMBER OF THIS KEY
                * ADD 8 FOR EACH ROW
A1F7                    BCC   LA1F4        GO ON UNTIL A ZERO APPEARS IN THE CARRY
A1F9                    ADDB  ,S           ADD IN THE COLUMN NUMBER
                ****************************************************************
                ** NOW CONVERT THE VALUE IN ACCB INTO ASCII
A1FB                    BEQ   LA245        THE 'AT SIGN' KEY WAS DOWN
A1FD                    CMPB  #26          WAS IT A LETTER?
A1FF                    BHI   LA247        NO
A2Ø1                    ORB   #$4Ø         YES, CONVERT TO UPPER CASE ASCII
A2Ø3                    BSR   LA22D        CHECK FOR THE SHIFT KEY
A2Ø5                    BEQ   LA2ØE        IT WAS DOWN
A2Ø7                    LDA   CASFLG       NOT DOWN, CHECK THE UPPER/LOWER CASE FLAG
A2ØA                    BNE   LA2ØE        UPPER CASE
A2ØC                    ORB   #$2Ø         CONVERT TO LOWER CASE
A2ØE           LA2ØE    STB   ,S           TEMP STORE ASCII VALUE
A21Ø                    LDX   DEBVAL       GET KEYBOARD DEBOUNCE
A213                    JSR   LA7D3        GO WAIT A WHILE
A216                    LDB   2,S          GET COLUMN STROBE DATA
A218                    STB   PIAØ+2       STORE IT
A21B                    BSR   LA238        READ A KEY
A21D                    CMPA  1,S          IS IT THE SAME KEY AS BEFORE DEBOUNCE?
A21F                    PULS  A            PUT THE ASCII VALUE OF KEY BACK IN ACCA
A221                    BNE   LA22A        NOT THE SAME KEY
```

```
A223                CMPA  #$12          IS SHIFT ZERO DOWN?
A225                BNE   LA22B         NO
A227                COM   CASFLG        YES, TOGGLE UPPER/LOWER CASE FLAG
A22A          LA22A CLRA                SET ZERO FLAG TO INDICATE NO NEW KEY DOWN
A22B          LA22B PULS  X,PC          RESTORE REGISTERS

              *** TEST FOR THE SHIFT KEY
A22D          LA22D LDA   #$7F          COLUMN STROBE
A22F                STA   PIAØ+2        STORE TO PIA
A232                LDA   PIAØ          READ KEY DATA
A235                ANDA  #$4Ø          CHECK FOR SHIFT KEY, SET ZERO FLAG IF DOWN
A237                RTS                 RETURN

              *** READ THE KEYBOARD
A238          LA238 LDA   PIAØ          READ PIAØ, PORT A TO SEE IF KEY IS DOWN
              **                        A BIT WILL BE ZERO IF ONE IS
A23B                ORA   #$8Ø          MASK OFF THE JOYSTICK COMPARATOR INPUT
A23D                TST   PIAØ+2        ARE WE STROBING COLUMN 7?
A24Ø                BMI   LA244         NO
A242                ORA   #$CØ          YES, FORCE ROW 6 TO BE HIGH -THIS WILL
              **                        CAUSE THE SHIFT KEY TO BE IGNORED
A244          LA244 RTS                 RETURN

A245          LA245 LDB   #51           CODE FOR 'AT SIGN'
A247          LA247 LDX   #CONTAB-$36   POINT X TO CONTROL CODE TABLE
A24A                CMPB  #33           KEY NUMBER <33?
A24C                BLO   LA264         YES (ARROW KEYS, SPACE BAR, ZERO)
A24E                LDX   #CONTAB-$54   POINT X TO MIDDLE OF CONTROL TABLE
A251                CMPB  #48           KEY NUMBER > 48?
A253                BHS   LA264         YES (ENTER, CLEAR, BREAK, AT SIGN)
A255                BSR   LA22D         CHECK SHIFT KEY (ACCA WILL CONTAIN STATUS)
A257                CMPB  #43           IS KEY A NUMBER, COLON OR SEMICOLON?
A259                BLS   LA25D         YES
A25B                EORA  #$4Ø          TOGGLE BIT 6 OF ACCA WHICH CONTAINS THE
              **                        SHIFT DATA ONLY FOR SLASH, HYPHEN, PERIOD,
              **                        COMMA
A25D          LA25D TSTA                SHIFT KEY DOWN?
A25E                BEQ   LA2ØE         YES
A26Ø                ADDB  #$1Ø          NO, ADD IN ASCII OFFSET CORRECTION
A262                BRA   LA2ØE         GO CHECK FOR DEBOUNCE

A264          LA264 ASLB                MULT ACCB BY 2 - THERE ARE 2 ENTRIES IN
              **                        CONTROL TABLE FOR EACH KEY - ONE SHIFTED,
              **                        ONE NOT
A265                BSR   LA22D         CHECK SHIFT KEY
A267                BNE   LA26A         NOT DOWN
A269                INCB                ADD ONE TO GET THE SHIFTED VALUE
A26A          LA26A LDB   B,X           GET ASCII CODE FROM CONTROL TABLE
A26C                BRA   LA2ØE         GO CHECK DEBOUNCE

*** LISTING 3
A2C3                BSR   LA2FB         SET OUTPUT TO MARKING
A2C5                ASLA                SEND 7 BITS AND ONE STOP BIT (BIT 7=Ø)
A2C6                LDB   #8            SEND 8 BITS
A2C8          LA2C8 PSHS  B             SAVE BIT COUNTER
A2CA                CLRB                CLEAR DA IMAGE 1 ZEROS TO DA WHEN SENDING
              *                         RS-232 DATA
```

```
A2CB                LSRA            ROTATE NEXT BIT OF OUTPUT CHARACTER TO
                *                   CARRY FLAG
A2CC                ROLB            * ROTATE CARRY FLAG INTO BIT ONE
A2CD                ROLB            * AND ALL OTHER BITS SET TO ZERO
A2CE                STB   DA        STORE IT TO DA CONVERTER
A2D1                BSR   LA3Ø2     GO WAIT A WHILE
A2D3                NOP
A2D4                NOP
A2D5                NOP
A2D6                BSR   LA3Ø2     GO WAIT SOME MORE
A2D8                PULS  B         GET BIT COUNTER
A2DA                DECB            SENT ALL 8 BITS?
A2DB                BNE   LA2C8     NO
A2DD                BSR   LA2FB     SEND STOP BIT (ACCB=Ø)
A2DF                PULS  CC,A      RESTORE OUTPUT CHARACTER & INTERRUPT STATS
A2E1                CMPA  #CR       IS IT A CARRIAGE RETURN?
A2E3                BEQ   LA2ED     YES
A2E5                INC   LPTPOS    INCREMENT CHARACTER COUNTER
A2E7                LDB   LPTPOS    CHECK FOR END OF LINE PRINTER LINE
A2E9                CMPB  LPTWID    AT END OF LINE PRINTER LINE?
A2EB                BLO   LA2F3     NO
A2ED          LA2ED CLRB  LPTPOS    RESET CHARACTER COUNTER
A2EF                BSR   LA3Ø5     *
A2F1                BSR   LA3Ø5     * DELAY FOR CARRIAGE RETURN
A2F3          LA2F3 LDB   PIA1+2    WAIT FOR HANDSHAKE
A2F6                LSRB            CHECK FOR RS232 STATUS
A2F7                BCS   LA2F3     NOT YET READY
A2F9                PULS  B,X,PC    RESTORE REGISTERS

*** LISTING 4
B3ED          INTCNV LDA  FPØEXP    GET FPAØ EXPONENT
B3EF                CMPA  #$9Ø      * COMPARE TO 32768 - LARGEST INTEGER
B3F1                BCS   LB3FB     * EXPONENT AND BRANCH IF FPAØ < 32768
B3F3                LDX   #LB3DF    POINT X TO FP VALUE OF -32768
B3F6                JSR   LBC96     COMPARE -32768 TO FPAØ
B3F9                BNE   LB44A     'FC' ERROR IF NOT =
B3FB          LB3FB JSR   LBCC8     CONVERT FPAØ TO A TWO BYTE INTEGER
F3FE                LDD   FPAØ+3    GET THE INTEGER
B4ØØ                RTS
                * EVALUATE AN ARRAY VARIABLE
B4Ø1          LB4Ø1 LDB   DIMFLG    GET ARRAY FLAG
B4Ø3                LDA   VALTYP    GET VARIABLE TYPE
B4Ø5                PSHS  A,B       SAVE THEM ON THE STACK
B4Ø7                CLRB            RESET DIMENSION COUNTER
B4Ø8          LB4Ø8 LDX   VARNAM    GET VARIABLE NAME
B4ØA                PSHS  B,X       SAVE VARIABLE NAME AND DIMENSION COUNTER
B4ØC                BSR   LB3E4     EVALUATE EXPRESSION (DIMENSION LENGTH)
B4ØE                PULS  B,X,Y     PULL OFF VARIABLE NAME, DIMENSION COUNTER
                *                   ARRAY FLAG
B41Ø                STX   VARNAM    SAVE VARIABLE NAME AND VARIABLE TYPE
B412                LDU   FPAØ+2    GET DIMENSION LENGTH
B414                PSHS  Y,U       SAVE DIMENSION LENGTH, ARRAY FLAG,
                                    VARIABLE TYPE
B416                INCB            INCREASE DIMENSION COUNTER
B417                JSR   GETCCH    GET CURRENT INPUT CHARACTER
B419                CMPA  #',       CHECK FOR ANOTHER DIMENSION
B41B                BEQ   LB4Ø8     BRANCH IF MORE
```

```
B41D                    STB   TMPLOC      SAVE DIMENSION COUNTER
B41F                    JSR   LB267       SYNTAX CHECK FOR A ")"
B422                    PULS  A,B         * RESTORE VARIABLE TYPE AND ARRAY
B424                    STA   VALTYP      * FLAG - LEAVE DIMENSION LENGTH ON STACK
B426                    STB   DIMFLG      *
```

Listed below are the differences between Color BASIC Version 1.1 and Version 1.2

CHANGE	ADDRESS	Version 1.Ø	Version 1.2
A	$AØØ1	$C1	$CB
B	$A114	$57	$58
C	$A155	$31	$32
D	$A15E	$3Ø	$32
E	$A1B5-$A26D	SEE LISTING 5	
F	$A2C3-$A2FA	SEE LISTING 6	
G	$A56A	$C1	$CB
H	$ADFD	$C1	$CB
J	$B23F	$E8	$E3
K	$B38E	$72	$75
L	$B3ED-$B427	SEE LISTING 7	
M	$B9D6	$2B BMI	$25 BCS

Change A is a branch length change caused by the keyboard driver mod.

Change B modified the line printer baud rate.

Change C is the version number.

Change D is the copyright year.

Change E is a major change (Listing 5) to the keyboard driver allowing a quick scan of the keyboard if no keys are down.

Change F is also a major change to the line printer driver which will not allow any transmission until the receiving device is ready (Listing 6).

Change G speeds up the INKEY$ command by not entering the keyboard scan routine if no keys are depressed.

Change H speeds up the BREAK key check routine by not entering the keyboard scan routine if no keys are depressed.

Change J is a minor change which slightly speeds up the expression evaluation routine.

Change K is merely a different length branch caused by change L.

Change L (Listing 7) causes a numeric variable type check to be done before the INTCNV (FPAØ to integer) routine.

Change M fixed a minor bug in the floating point addition routine.

*** LISTING 5

```
A1B5                BSR   KEYIN        GO CHECK KEYBOARD
A1B7                BEQ   LA1B3        LOOP IF NO KEY DOWN
A1B9                LDB   #$6Ø         BLANK
A1BB                STB   [CURPOS]     BLANK CURRENT CURSOR CHAR ON SCREEN
A1BF          LA1BF PULS  B,X,PC
```

```
                    *
                    * THIS ROUTINE GETS A KEYSTROKE FROM THE KEYBOARD IF A KEY
                    * IS DOWN. IT RETURNS ZERO TRUE IF THERE WAS NO KEY DOWN.
                    *
A1C1                KEYIN   PSHS  B,X,U       SAVE REGISTERS
A1C3                        BSR   LA1C8       GET KEYSTROKE
A1C5                        TSTA              SET FLAGS
A1C6                        PULS  B,X,U,PC    RESTORE REGISTERS
A1C8                LA1C8   LDU   #PIAØ       POINT TO PIAØ
A1CA                        LDX   #KEYBUF     KEYBOARD MEMORY BUFFER
A1CE                        CLRA              * CLEAR CARRY FLAG, SET COLUMN COUNTER
A1CF                        DECA              * (ACCA) TO $FF
A1DØ                        PSHS  X,A         SAVE COLUMN CTR & 2 BLANK (X REG) ON STACK
A1D2                        STA   2,U         INITIALIZE COLUMN STROBE TO $FF
A1D4                        FCB   SKP1        SKIP ONE BYTE
A1D5                LA1D5   COMB              SET CARRY FLAG
A1D6                        ROL   2,U         * ROTATE COLUMN STROBE DATA LEFT 1 BIT,
A1D8                        BCC   LA1BF       * CARRY INTO BIT Ø-RETURN IF 8 BITS DONE
A1DA                        INC   ,S          INCREMENT COLUMN POINTER
A1DC                        BSR   LA239       READ KEYBOARD DATA ROW
A1DE                        STA   1,S         TEMP STORE KEY DATA
A1EØ                        EORA  ,X          SET ANY BIT WHERE A KEY HAS MOVED
A1E2                        ANDA  ,X          ACCA=Ø IF NO NEW KEY DOWN, <7Ø IF KEY WAS
                    *                         RELEASED
A1E4                        LDB   1,S         GET NEW KEY DATA
A1E6                        STB   ,X+         STORE IT IN KEY MEMORY
A1E8                        TSTA              WAS A NEW KEY DOWN?
A1E9                        BEQ   LA1D5       NO-CHECK ANOTHER COLUMN
A1EB                        LDB   2,U         * GET COLUMN STROBE DATA AND
A1ED                        STB   2,S         * TEMP STORE IT ON THE STACK
                    ************************************************************
                    ** THIS ROUTINE CONVERTS THE KEY DEPRESSION INTO A NUMBER
                    ** FROM Ø-5Ø IN ACCB CORRESPONDING TO THE KEY THAT WAS DOWN
A1EF                        LDB   #$F8        TO MAKE SURE ACCB=Ø AFTER FIRST ADDB #8
A1F1                LA1F1   ADDB  #8          ADD 8 FOR EACH ROW OF KEYBOARD
A1F3                        LSRA              ACCA CONTAINS THE ROW NUMBER OF THIS KEY
                    *                         ADD 8 FOR EACH ROW
A1D4                        BCC   LA1F1       GO ON UNTIL A ZERO APPEARS IN THE CARRY
A1F6                        ADDB  ,S          ADD IN THE COLUMN NUMBER
                    ************************************************************
                    ** NOW CONVERT THE VALUE IN ACCB INTO ASCII
A1F8                        BEQ   LA244       THE 'AT SIGN' KEY WAS DOWN
A1FA                        CMPB  #26         WAS IT A LETTER?
A1FC                        BHI   LA246       NO
A1FE                        ORB   #$4Ø        YES, CONVERT TO UPPER CASE ASCII
A2ØØ                        BSR   LA22E       CHECK FOR THE SHIFT KEY
A2Ø2                        BEQ   LA2ØB       IT WAS DOWN
A2Ø4                        LDA   CASFLG      NOT DOWN, CHECK THE UPPER/LOWER CASE FLAG
A2Ø7                        BNE   LA2ØB       UPPER CASE
A2Ø9                        ORB   #$2Ø        CONVERT TO LOWER CASE
A2ØB                LA2ØB   STB   ,S          TEMP STORE ASCII VALUE
A2ØD                        LDX   DEBVAL      GET KEYBOARD DEBOUNCE
A21Ø                        JSR   LA7D3       GO WAIT A WHILE
A213                        LDB   #$FF        * SET COLUMN STROBE TO ALL ONES (NO
A215                        BSR   LA237       * STROBE) AND READ KEYBOARD
A217                        INCA              = INCR ROW DATA, ACCA NOW Ø IF NO JOYSTK
A218                        BNE   LA22Ø       = BUTTON DOWN. BRANCH IF JOYSTK BUTTON DN
```

```
A21A            LA21A   LDB   2,S          GET COLUMN STROBE DATA
A21C                    BSR   LA237        READ A KEY
A21E                    CMPA  1,S          IS IT THE SAME KEY AS BEFORE DEBOUNCE?
A220            LA220   PULS  A            PUT THE ASCII VALUE OF KEY BACK IN ACCA
A222                    BNE   LA22B        NOT THE SAME KEY
A224                    CMPA  #$12         IS SHIFT ZERO DOWN?
A226                    BNE   LA22C        NO
A228                    COM   CASFLG       YES, TOGGLE UPPER/LOWER CASE FLAG
A22B            LA22B   CLRA               SET ZERO FLAG TO INDICATE NO NEW KEY DOWN
A22C            LA22C   PULS  X,PC         REMOVE TEMP STORAGE SLOTS FROM STACK

                *** TEST FOR THE SHIFT KEY
A22E            LA22E   LDA   #$7F         COLUMN STROBE
A230                    STA   2,U          STORE TO PIA
A232                    LDA   ,U           READ KEY DATA
A234                    ANDA  #$40         CHECK FOR SHIFT KEY, SET ZERO FLAG IF DOWN
A236                    RTS                RETURN

                *** READ THE KEYBOARD
A237            LA237   STB   2,U          SAVE NEW COLUMN STROBE VALUE
A239            LA239   LDA   ,U           READ PIA0, PORT A TO SEE IF KEY IS DOWN
                **                         A BIT WILL BE ZERO IF ONE IS
A23B                    ORA   #$80         MASK OFF THE JOYSTICK COMPARATOR INPUT
A23D                    TST   2,U          ARE WE STROBING COLUMN 7?
A23F                    BMI   LA243        NO
A241                    ORA   #$C0         YES, FORCE ROW 6 TO BE HIGH -THIS WILL
                **                         CAUSE THE SHIFT KEY TO BE IGNORED
A243            LA243   RTS                RETURN

A244            LA244   LDB   #51          CODE FOR 'AT SIGN'
A246            LA246   LDX   #CONTAB-$36  POINT X TO CONTROL CODE TABLE
A249                    CMPB  #33          KEY NUMBER <33?
A24B                    BLO   LA263        YES (ARROW KEYS, SPACE BAR, ZERO)
A24D                    LDX   #CONTAB-$54  POINT X TO MIDDLE OF CONTROL TABLE
A250                    CMPB  #48          KEY NUMBER > 48?
A252                    BHS   LA263        YES (ENTER, CLEAR, BREAK, AT SIGN)
A254                    BSR   LA22E        CHECK SHIFT KEY (ACCA WILL CONTAIN STATUS)
A256                    CMPB  #43          IS KEY A NUMBER, COLON OR SEMICOLON?
A258                    BLS   LA25C        YES
A25A                    EORA  #$40         TOGGLE BIT 6 OF ACCA WHICH CONTAINS THE
                **                         SHIFT DATA ONLY FOR SLASH, HYPHEN, PERIOD,
                **                         COMMA
A25C            LA25C   TSTA               SHIFT KEY DOWN?
A25D                    BEQ   LA20B        YES
A25F                    ADDB  #$10         NO, ADD IN ASCII OFFSET CORRECTION
A261                    BRA   LA20B        GO CHECK FOR DEBOUNCE

A263            LA263   ASLB               MULT ACCB BY 2 - THERE ARE 2 ENTRIES IN
                **                         CONTROL TABLE FOR EACH KEY - ONE SHIFTED,
                **                         ONE NOT
A264                    BSR   LA22E        CHECK SHIFT KEY
A266                    BNE   LA269        NOT DOWN
A268                    INCB               ADD ONE TO GET THE SHIFTED VALUE
A269            LA269   LDB   B,X          GET ASCII CODE FROM CONTROL TABLE
A26B                    BRA   LA20B        GO CHECK DEBOUNCE
A26D                    FCB   0            WASTED SPACE IN VERSION 1.1
```

```
*** LISTING 6
A2C3                BSR   LA2FB       SET OUTPUT TO MARKING
A2C5                CLRB              *
A2C6                BSR   LA2FD       * TRANSMIT ONE START BIT
A2C8                LDB   #8          SEND 8 BITS
A2CA          LA2CA PSHS  B           SAVE BIT COUNTER
A2CC                CLRB              CLEAR DA IMAGE 1 ZEROS TO DA WHEN SENDING
              *                       RS-232 DATA
A2CD                LSRA              ROTATE NEXT BIT OF OUTPUT CHARACTER TO
              *                       CARRY FLAG
A2CE                ROLB              * ROTATE CARRY FLAG INTO BIT ONE
A2CF                ASLB              * AND ALL OTHER BITS SET TO ZERO
A2D0                BSR   LA2FD       TRANSMIT DATA BYTE
A2D2                PULS  B           GET BIT COUNTER
A2D4                DECB              SENT ALL 8 BITS?
A2D5                BNE   LA2CA       NO
A2D7                BSR   LA2FB       SEND STOP BIT (ACCB=0)
A2D9                PULS  CC,A        RESTORE OUTPUT CHARACTER & INTERRUPT STATS
A2DB                CMPA  #CR         IS IT A CARRIAGE RETURN?
A2DD                BEQ   LA2E7       YES
A2DF                INC   LPTPOS      INCREMENT CHARACTER COUNTER
A2E1                LDB   LPTPOS      CHECK FOR END OF LINE PRINTER LINE
A2E3                CMPB  LPTWID      AT END OF LINE PRINTER LINE?
A2E5                BLO   LA2ED       NO
A2E7          LA2E7 CLR   LPTPOS      RESET CHARACTER COUNTER
A2E9                BSR   LA305       *
A2EB                BSR   LA305       * DELAY FOR CARRIAGE RETURN
A2ED          LA2ED LDB   PIA1+2      WAIT FOR HANDSHAKE
A2F0                LSRB              CHECK FOR RS232 STATUS
A2F1                BCS   LA2ED       NOT YET READY
A2F3                PULS  B,X,PC      RESTORE REGISTERS
A2F5                FDB   0,0,0       WASTED SPACE IN VERSION 1.1

*** LISTING 7
B3ED          INTCNV LDA  FP0EXP      GET FPA0 EXPONENT
B3EF                CMPA  #$90        * COMPARE TO 32768 - LARGEST INTEGER
B3F1                BCS   LB3FB       * EXPONENT AND BRANCH IF FPA0 < 32768
B3F3                LDX   #LB3DF      POINT X TO FP VALUE OF -32768
B3F6                JSR   LBC96       COMPARE -32768 TO FPA0
B3F9                BNE   LB44A       'FC' ERROR IF NOT =
B3FB          LB3FB JSR   LBCC8       CONVERT FPA0 TO A TWO BYTE INTEGER
F3FE                LDD   FPA0+2      GET THE INTEGER
B400                RTS
              * EVALUATE AN ARRAY VARIABLE
B401          LB401 LDB   DIMFLG      GET ARRAY FLAG
B403                LDA   VALTYP      GET VARIABLE TYPE
B405                PSHS  A,B         SAVE THEM ON THE STACK
B407                CLRB              RESET DIMENSION COUNTER
B408          LB408 LDX   VARNAM      GET VARIABLE NAME
B40A                PSHS  B,X         SAVE VARIABLE NAME AND DIMENSION COUNTER
B40C                BSR   LB3E4       EVALUATE EXPRESSION (DIMENSION LENGTH)
B40E                PULS  B,X,Y       PULL OFF VARIABLE NAME, DIMENSION COUNTER
              *                       ARRAY FLAG
B410                STX   VARNAM      SAVE VARIABLE NAME AND VARIABLE TYPE
B412                LDU   FPA0+2      GET DIMENSION LENGTH
B414                PSHS  Y,U         SAVE DIMENSION LENGTH, ARRAY FLAG,
                                      VARIABLE TYPE
```

```
B416                    INCB              INCREASE DIMENSION COUNTER
B417                    JSR   GETCCH      GET CURRENT INPUT CHARACTER
B419                    CMPA  #',         CHECK FOR ANOTHER DIMENSION
B41B                    BEQ   LB4Ø8       BRANCH IF MORE
B41D                    STB   TMPLOC      SAVE DIMENSION COUNTER
B41F                    JSR   LB267       SYNTAX CHECK FOR A ")"
B422                    PULS  A,B         * RESTORE VARIABLE TYPE AND ARRAY
B424                    STA   VALTYP      * FLAG - LEAVE DIMENSION LENGTH ON STACK
B426                    STB   DIMFLG      *
```

DISPLAY CHARACTER SET

HEX VALUE		CHARACTER	HEX VALUE		CHARACTER	HEX VALUE		CHARACTER
Non-Inverted	Inverted		Non-Inverted	Inverted		Non-Inverted	Inverted	
ØØ	4Ø	@	18	58	X	3Ø	4Ø	Ø
Ø1	41	A	19	59	Y	31	41	1
Ø2	42	B	1A	5A	Z	32	42	2
Ø3	43	C	1B	5B	[	33	43	3
Ø4	44	D	1C	5C	\	34	44	4
Ø5	45	E	1D	5D	]	35	45	5
Ø6	46	F	1E	5E	↑	36	46	6
Ø7	47	G	1F	5F	←	37	47	7
Ø8	48	H	2Ø	6Ø		38	48	8
Ø9	49	I	21	61	!	39	49	9
ØA	4A	J	22	62	"	3A	4A	:
ØB	4B	K	23	63	#	3B	4B	;
ØC	4C	L	24	64	$	3C	4C	<
ØD	4D	M	25	65	%	3D	4D	=
ØE	4E	N	26	66	&	3E	4E	>
ØF	4F	O	27	67	'	3F	4F	?
1Ø	5Ø	P	28	68	(			
11	51	Q	29	69	)			
12	52	R	2A	6A	*			
13	53	S	2B	6B	+			
14	54	T	2C	6C	,			
15	55	U	2D	6D	-			
16	56	V	2E	6E	.			
17	57	W	2F	6F	/			