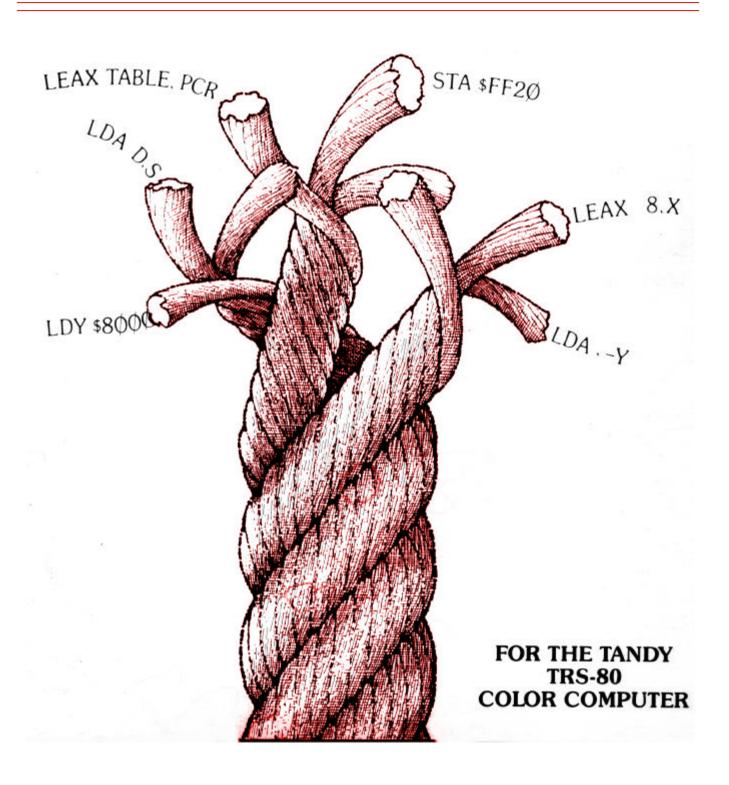
# DISK BASIC UNRAVELLED II



# TABLE OF CONTENTS

1	FOREWORD	1
2	INTRODUCTION	3
3	HOW TO USE THIS BOOK	4
4	DESCRIPTION OF DISK BASIC FILE CONTROL BLOCK STRUCTURE FILE ALLOCATION TABLE THE DIRECTORY 1793 FLOPPY DISK CONTROLLER DESCRIPTION MACHINE LANGUAGE FILE INPUT/OUTPUT DISK BASIC RAM VARIABLES/BUFFERS	5 6 9 11 13 17 18
	APPENDICES	
Α	MEMORY MAP	
В	DISASSEMBLY OF DISK BASIC 1.1	
С	DISASSEMBLY OF DISK BASIC 1.0	
D	DISK BASIC SYMBOL TABLE 1.1	
E	DISK BASIC SYMBOL TABLE 1.0	
F	DISK BASIC ROUTINES AND ENTRY POINTS	
G	DISK BASIC S DATA/ASCII TABLES	
Н	DISK BASIC ERROR ENTRY POINTS	
I	ASCII CHART	

### 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 line numbers. Each Appendix (with code) starts at Line  $\emptyset\emptyset\emptyset1$ .
- 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 Ø 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: <a href="mailto:wzydhek@internetcds.com">mailto:wzydhek@internetcds.com</a>

Special Thanks to Jean-François Morin for pointing out those Oops to me. I d like to also thank those who have either given me, or loaned me their copy of the original Unravelled Series.

# About Me

My name is Walter K. Zydhek. I ve been a Computer Hobbyist since 1984 when I received my  $1^{\rm st}$  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 the Carolwood CoCo Club. (Thanks Jean-François for reminding me of 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 whose 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 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 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

Disk Basic Unravelled will provide the reader with a complete detailed and fully commented assembly listing of the Disk Operating System (DOS) of Radio Shack s COLOR BASIC. It is not within the scope of this book to teach the neophyte how to construct a DOS or to be able to completely understand the COLOR DOS. The reader will need to have a basic knowledge of 6809 assembly language programming to be able to take full advantage of the opportunities, which this book presents. It is also assumed that the reader is familiar with the contents of the Disk Basic Users manual, which contains a general description of the overall operation of Disk Basic and much useful information concerning the physical and logical format of the tracks & sectors. Disk Basic Unravelled will allow the reader to be able to completely understand the theory behind COLOR DOS to the point of being able to modify it for his own purposes or add extra commands or functions to the DOS.

No attempt will be made to re explain the functions of BASIC and Extended Basic, which were explained in the previous two books of the BASIC Unravelled series. The reader should be aware of the fact that Color Disk Basic is not a stand-alone system. There are many direct calls into the Basic and Extended Basic ROMs. These calls are not explained in this book; the reader will have to refer to the Basic and Extended Basic Unravelled books in order to get a full explanation of these ROM calls. A complete memory map of the system operating variables is given at the beginning of the DOS assembly listing and a symbol table showing the location of all referenced routines and tables is at the end of the listing.

All of the ROMs used in the Color Computer have undergone revisions since the inception of the machine. The disk ROMs have undergone the most severe change of the three ROMs. The first disk RON (Revision 1.0) used only 6K of the available 8K RON space and the second disk RON (Revision 1.1) used approximately 6.5K of ROM with the majority of the .5K increase going to correct bugs in the first ROM and to add the DOS command to Disk Basic. That leaves 1.5K of free ROM space in the latest version of Disk Basic, which is available to the user if he has a 64K machine. It is not recommended that this free ROM space be permanently allocated by any user since the Disk Basic ROMs in the Dragon computer (a British clone of the Color Computer) use the entire 8K ROM space and have added several new disk BASIC commands. This means that the commands are also probably available to Radio Shack and version 1.2 of the BASIC ROM, which may contain some of these commands, will be coming along sometime.

The new revisions of the Color Basic and Extended Basic ROMs kept the majority of the code in the same position in the ROM. In the case of the Disk Basic revisions, however, no effort was made to keep the code in the same position. There are two reasons for this: the first is that there were so many changes that it would have been very difficult to maintain the position and secondly, there was 2K of additional ROM space available so why try to maintain the position. The total positional difference between the two versions makes it very difficult to have one assembly listing which owners of either Disk Basic ROM may use. To solve this problem, an assembly listing of both versions is contained in the book. The 1.1 version will be the most useful since it has had most of the bugs, which were in the 1.0 version corrected. The complete memory map will not be given for the 1.0 version since the memory maps for both versions of the ROM are identical.

# HOW TO USE THIS BOOK

Disk BASIC Unravelled is a commented, disassembled listing of the TRS-80 Color Computer Disk BASIC ROM. The author has never seen any kind of source listing for the Color Computer ROMs, so the comments and disassembly are 100% unique. Some of the variable label literals, which were used, have come from published memory maps of systems, which use a BASIC similar to that used in the Color Computer.

The labels used in the disassembly correspond to absolute addresses in RAM preceded by an L . The labels correspond to the ad-dresses in Version 1.0 of the ROM, which may cause some confusion when trying to cross-index the 1.0 and 1.1 versions.

Literal labels have been assigned to RAM variables (memory locations that contain data which may change) and some ROM routines and data tables. The symbol table in Appendix D will allow the user to locate the address of the literal label. If the address is between Ø and \$989, the literal is a RAM variable, the description of which will be found in appendix A, the Memory Map. If the address is between \$8000 and \$9FFF, the label will be found in the Extended BASIC listing; if it is between \$A000 and \$BFFF, the label is in the Color BASIC listing and if it is between \$C000 and \$DFFF, the label is in the Disk BASIC listing. Some of the literal values such as SKP1, SECLEN, etc. are values not associated with an address. They are defined at the beginning of the Memory Map (appendix A) in the table of EQUATES (EQU). There is an additional group of EQUates at the beginning of the Disk Basic disassembly listing (Appendices B & C).

The > symbol will occasionally appear to the left of the address of an instruction. This symbol is used to indicate that a JMP, JSR or LBxx instruction is being used when a BRA, BSR or Bxx instruction would suffice. These instructions may be replaced by their short versions in order to save a few bytes if necessary.

There are several places in the original object code where an instruction of the form LDA  $\emptyset$ ,R (where R = X,Y,U,S) has been used. These have been replaced by instructions of the form LDA ,R which are more efficient in terms of processor time (one cycle shorter).

The different versions of the ROMs provided in this book are kept in one large disk file with conditional assembly flags which allow the assembly of whichever version is desired by merely changing a single flag in the source listing. This is a convenient method of keeping track of the different versions of the ROMs but it can cause havoc with the line numbers at the extreme left of the disassembly listing. The line numbers keep track of EVERY line in the source listing regardless of whether or not that particular line is assembled. If when using the disassembly listings, you notice a gap in the line numbers it means that the missing line numbers correspond to a section of code, which was skipped during the assembly of that particular listing. This invariably means that there is a difference in the ROMs at that particular point.

### DESCRIPTION OF DISK BASIC

Disk Basic will allow the Color Computer to communicate with a floppy disk drive in order to rapidly store large amounts of data on a non volatile medium. Disk Basic is different from Extended Basic in the manner that Extended Basic provided the user with a package of graphics commands AND several useful non graphics commands, whereas Disk Basic provides ONLY disk oriented commands with no additional commands (there is approximately 1.5K of wasted space where something else could have been provided). Accordingly, any discussion of Disk Basic will center around only the TRS 80 Color Computer s DOS (Disk Operating System)

As computers have evolved over the years, one of the biggest problems faced was the storage of the large amounts of data and programs, which the computer must have access to. The amount of random access storage available to the user was relatively small compared to the total amount of storage required. Random access memory is very fast, fairly expensive and volatile (it is lost when the power is turned off). The first method of mass storage used was magnetic tape, which was non volatile and cheap, but slow. Then came the floppy disk which was non volatile, not quite so cheap, and faster than magnetic tape. Presently the floppy disk is the primary system for mass storage in microcomputers.

A floppy disk is a round piece of magnetic tape shaped like a record on which data is magnetically recorded. Somehow the data, which is stored on the disk, must be capable of being transferred to and from the computer s random access memory. This is a very complex task, which requires many things in order to be done properly. There must be a mechanical method of moving the disk and transferring the magnetic data to and from the disk. This job is performed by the disk drive. Also, there must be a method of formatting and transferring data to and from the computer s RAM and the disk drive. In the Color Computer the disk controller board does this. The majority of the work done by the disk controller is performed by the 1793 Floppy Disk Controller (FDC) which is an integrated circuit as complicated as the 6809 chip. In order to make the process orderly and logical there must be an overall controlling format or procedure for sending data to and from the 1793 (which will only provide primitive transference of blocks of data to and from the drive). The Disk Operating System (DOS) provides this overall control function by establishing a format for storing data and programs on the disk. The DOS provides a method of storing or retrieving blocks or single bytes of data to or from the disk drive.

The 1793 is capable of storing data on a disk in many different formats. For the Color Computer the 1793 is set up to save data on the disk in 35 tracks. Each track is composed of 18 sectors and each sector contains 256 bytes. The DOS treats this raw data as 68 granules with each granule containing 9 sectors, 2 granules per track. The one remaining track is used for the directory and the file allocation table (FAT).

# FILE CONTROL BLOCK STRUCTURE

The File Control Block (FCB) is used by the DOS to control the transfer of data between the computer s RAM and the disk. It consists of 25 control bytes and a 256-byte data buffer. The 25 control bytes may have different functions if the file is a random/direct, sequential input or sequential output file. The data buffer is used to collect data so that the disk I/O will only be required when there is a full sector (256 bytes) of data to be input or output to the disk. The use of this buffer speeds up the overall disk I/O by cutting down on the number of times that actual disk accesses are required.

The number of FCBs allowed is set by the FILES command, which initializes the direct page variable FCBACT (the maximum number of FCBs allowed). The DOS always sets up a system FCB directly above the last allocated FCB, which is reserved for the exclusive use of the DOS and is not accessible to the user through BASIC. The system FCB is used when the system requires an FCB for disk I/O during the execution of such commands as MERGE, COPY, SAVE, LOAD, etc. This FCB may be accessed by the user under machine language but care must be exercised to insure that none of the BASIC commands which utilize the system FCB are used when doing

The OPEN command is used to initialize the FCB for disk I/O. It keeps track of which byte, sector, track and granule is currently being accessed by the DOS for the file controlled by the FCB. When disk I/O has been completed, the FCB is deactivated with the CLOSE command. When an FCB is closed, it is available for use by another file and once the FCB is used by another file, all of the information used by the previous file is lost. Some of the information must be saved since the user may want to reopen the same file for use later on. Only six bytes from the FCB must be saved in order to be able to reinitialize an FCB. These six bytes are the file type (1), ASCII flag (1), first granule in file (2) and the number of bytes used in the last sector (2) and they are stored in the directory. A two-byte quantity is used to store number of bytes used in the last sector since the number of bytes may be any number from Ø to 256 (\$100).

Listed below are those FCB control bytes, which are common to all types of files and their relative offset from the start of the FCB.

<u>OFFSEI</u> Ø	<u>NAME</u> FCBTYP	Single byte code representing the file type under which the file was opened. It may not have any relationship to the actual type of data stored in the file; a sequential file may be opened as a random file and vice versa. The allowed codes are:
		\$10 = Sequential input, $$20$ = Sequential output, $$40$ = random, $0$ = killed file.
1	FCBDRV	Single byte quantity defining the drive number where the file is located (0 3).
2	FCBFGR	Single byte quantity defining the first granule used by the file.
3	FCBCGR	Single byte quantity defining the current granule being accessed by the FCB.

4	FCBSEC	Single byte quantity defining the current sector being accessed by the FCB (1 9). $$
18	FCBDIR	Single byte quantity defining the directory entry number for this file (0 71).
19	FCBLST	Double byte quantity containing the number of bytes used by this file in the last sector of the file.

Listed below are the definitions of the non common FCB control bytes as used by random files.

OFFSET 5	<u>NAME</u> FCBCPT	<u>DESCRIPTION</u> Unused
6	FCBPOS	Print position - always zero
7	FCBREC	Double byte quantity containing the current record number being used by the FCB.
8	FCBRLN	Double byte quantity containing the length of a record
11	FCBBUF	Double byte quantity containing a pointer to the absolute address of the start of random file buffer, which is exactly one record length long.
13	FCBSOF	Double byte quantity containing the sector offset to the current position in the record. These bytes are used to keep track of how many sectors from the beginning of a random file the current data being processed is located. These bytes are used to determine if the data in the FCB data buffer are valid for the current record number being processed. The high order byte is often set to \$FF to cause new data to be read into the FCB data buffer.
15	FCBFLG	Single byte GET / PUT flag: Ø=GET, 1=PUT.
16		Two unused bytes
21	FCBGET	Double byte quantity containing the number of characters, which have been pulled out of the current record. These bytes are set to zero every time a record is stored in (PUT) or retrieved from (GET) a file.
23	FCBPUT	Double byte quantity containing the number of characters, which have been PUT into the current record. These bytes are set to zero every time a record is stored in (PUT) or retrieved from (GET) a file.

Listed below are the definitions of the non common FCB control bytes as used by sequential files.

OFFSET 5	NAME FCBCPT	DESCRIPTION Single byte quantity pointing to the next character to be processed for input files. When this byte is incremented to zero it indicates that the data buffer needs to be refilled. For output files this byte is used to indicate that 256 bytes of the last sector in the file have been used in case a DISK FULL error occurs while searching for an unused granule6 FCBPOS  Single byte quantity containing the current print position in the file for output files, unused for input files. A carriage return in the output data stream will reset this value to zero.
7	FCBREC	Double byte quantity containing the number of whole sectors which have been input or output to a file.
9 15		Seven unused bytes.
16	FCBCFL	Single byte cache flag: ØØ=cache empty, \$FF=cache full when inputting data, the DOS treats a CR, LF sequence as a CR. Therefore the DOS must look for a LF after a CR and if it does not find a LF, it must save that character for the next time an input character is needed. The cache flag indicates whether or not an extra character, which needs to be saved (cached), has been pulled out of an input file.
17	FCBCDT	Single byte cache data byte. If the cache flag is set the cache data byte is stored here.
23	FCBDFL	Single byte data left flag for input files: $\emptyset\emptyset=$ data still left in file, \$FF=no data left in file.
24	FCBLFT	Single byte quantity containing the number of characters left in the data buffer of an input file or the number of characters stored in the data buffer of an output file.

### FILE ALLOCATION TABLE

The file allocation table (FAT) is used to keep track of whether or not a granule has been allocated to a file or if it is free. The FAT is composed of six control bytes followed by 68 data bytes—one byte for each granule. The FAT is stored on sector two of the directory track (17). A RAM image of the FAT is kept in the disk RAM for each of the four possible drives. Keeping an image of the FAT in RAM helps speed up the overall operation of the DOS by eliminating the need for disk I/O every time the DOS modifies the FAT. Saving the FAT to disk is done approximately every 19 times that a new granule is pulled from the free granule reserve. It is written to disk whenever a file is closed and there are some DOS operations, which force the FAT to be written to disk when that DOS operation allocates a free granule.

Only the DOS uses two of the six control bytes. The first FAT control byte keeps track of how many FCBs are active on the drive for a particular FAT. This byte is used to preclude the loading in of the FAT from disk when there is any active file currently using the FAT. You can imagine the disaster, which would occur if you were creating a file and had allocated some granules to your new file but had not saved the new FAT to disk when the old FAT was loaded into RAM on top of the new FAT. Your new file would be hopelessly gone. For that reason the DOS must not allow the FAT to be loaded into RAM from disk while an FCB is active for that FAT.

The second FAT control byte is used to govern the need to write data from the FAT RAM image to the disk. If the value of this byte is zero it means that the FAT RAM image is an exact copy of what is currently stored on the disk. If the value is non zero, it indicates that the data in the FAT RAM image has been changed since the last time that the FAT was written to disk. The number stored in this byte is an indicator of how many granules have been removed from the FAT since the last FAT to disk write. Some BASIC commands, such as KILL, cause an immediate FAT RAM image to disk write when granules are either freed or allocated. Other commands, which allocate granules, increment the second FAT control byte. This byte is then compared to the disk variable WFATVL and when the second control byte is >= WFATVL, the FAT is written to disk.

The FAT data bytes are used to determine whether or not a granule is free and if it has been allocated they are used to determine to which file the granule belongs. If a data byte is \$FF, it means that the granule is free and may be allocated to any file. If a granule has been allocated, it is part of a sector chain, which defines which granules belong to a certain file. The only information required to be able to trace the granule chain is the number of the first granule in the chain. If the first granule of the chain is not known, the chain cannot be traced down backwards.

A granule data byte, which has been allocated, will contain a value, which is the number of the next granule in the granule chain for that file. If the two most significant bits (6,7) of a granule data byte are set, then that granule is the last granule in a file s granule chain. The low order four bits will contain the number of sectors in the last granule, which the file uses. Even though a file may not use all of the sectors in the last granule in the chain, no other file may use the sectors. Disk space is not allocated on a sector basis, it is allocated on a granule basis and the granule may not be broken down. The smallest one-byte file

will still require a full granule to be allocated in order to store the file.

Granules are allocated in such a manner that will cause them to be relatively uniformly spread around the disk. This will lessen wear on the disk by not always allocating certain granules so that the disk drive head will not pass over certain sections of the disk too often. This is a common method used by a DOS in order to increase the life of a disk by spreading out the wear over as large a surface as possible, which could not be done if the granules were allocated on a strictly next in line numerical basis.

Bvte

# THE DIRECTORY

The directory is used by the DOS to keep track of how many files are stored on a disk. Track 17 is reserved for the directory and the file allocation table (FAT). The FAT resides on sector 2 and the directory occupies sectors 3-11. The remaining sectors are not used by the DOS in the current or past revisions to the BASIC ROMs. Each directory entry requires 32 bytes; so eight directory entries will fit in one sector for a total of 72 maximum directory entries. However, one full granule is required for each directory entry and there are only 68 granules on a disk so that only 68 directory entries (files) may exist on a disk at any time.

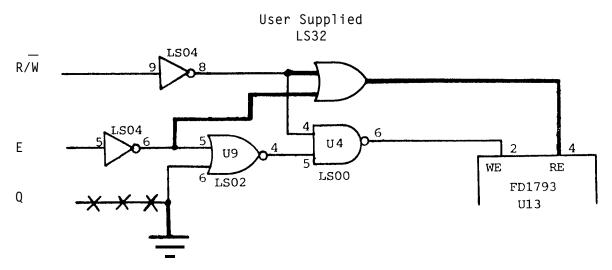
The format of the 32-byte directory entry is as follows:

Description

<u>0</u> 7	Filename, which is left justified and blank, filled. If byteØ is Ø, then the file has been KILL ed and the directory entry is available for use. If byteØ is \$FF, then the entry and all following entries have never been used.
8 10	Filename extension
11	File type: Ø=BASIC, 1=BASIC data, 2=Machine language, 3= Text editor
	source
12	ASCII flag: Ø=binary or crunched BASIC, \$FF=ASCII
13	Number of the first granule in the file
14 15	Number of bytes used in the last sector of the file
16 31	Unused (future use)

# **DIRECTORY CRASHES**

There has been much talk in the Color Computer media about so called directory crashes. These directory crashes seem to occur at random and will result in an unreadable directory, which causes the loss of all data on the directory at the time. The typical solution for this problem seems to be keeping a spare copy of the directory on an unused portion of the disk so that if a crash occurs, the spare copy may be used to restore the directory. A couple of years ago, we were having the same problem and Rodger Rosenbaum, genius extraordinaire, solved the problem by finding the bug in the disk controller which was causing it. Given below is the hardware fix, which will cure the directory crash problem. Spectral Associates does not guarantee nor does it recommend this fix and Spectral Associates will not assume any responsibility or liability for damages caused should any person or entity use or attempt to use the fix.



NOTE: This fix is only valid for the old style disk controllers. The new 5-volt only controllers do not suffer from this problem and should not be modified.

Add the wires shown with bold black lines. In order to install the fix without cutting traces on the PC board, gently lift IC s U9 and U13 out of their sockets, bend pins 6 and 4 respectively out and replace the IC s in their sockets. Then solder wires to the bent out pins.

### 1793 FLOPPY DISK CONTROLLER DESCRIPTION

The 1793 Floppy Disk Controller (FDC) is the heart of the disk controller card, which interfaces the Color Computer to the disk drive. Only the basic knowledge of the FDC in order to understand the operation of Disk Basic is presented here. If further, detailed information concerning the operation of the FDC is required, the reader is referred to the 1793 data sheet published by the Western Digital Corp. Only those functions of the 1793, which may be accessed by software the Color Computer, will be discussed.

The FDC is responsible for controlling the transference of data between the computer and the disk drive. There are many different disk drives manufactured by different companies and each drive has its own peculiarities, which require the FDC to be set up or used in a slightly different manner. The Color Computer Disk Controller board (which you plug into the cartridge slot) provides most of the support and set up functions, which the FDC requires and an eight-bit latch is used to store the FDC functions, which are software programmable, and those disk drive functions, which must be controlled directly by the computer. This control latch is located at \$FF4Ø and is a write only latch, which means that data may only be written into the latch; no provision has been given to read the contents of the latch. For this reason, the DOS has reserved a byte in the Disk RAM (DRGRAM) which is an image of the contents of \$FF4Ø so that the system software will know the status of the control latch at any time. Listed below are the functions, which may be controlled by DSKREG (\$FF4Ø).

<u>bit #</u>	<u>Description</u>
Ø	Drive select Ø
1	Drive select 1
2	Drive select 2
3	Drive motor enable: Ø=motors off, 1=motors on
4	Write pre-compensation flag: Ø=no pre-comp, 1=pre-comp
5	Density flag: Ø=single density, 1=double density
6	Drive select 3
7	Halt flag: ∅=halt disabled, 1=halt enabled

### \$FF40 Control Functions

The drive select flags directly control which drive will communicate with the computer and the motor enable flag will enable or disable the motors of all of the drives. The density flag indicates to the FDC whether the data will be stored onto the diskette in single or double density. Write pre-compensation is used to correct the problem on a double density formatted disk of certain bit patterns causing a bit to shift from its nominal write position and appear at the read data separator early or late. Write pre-compensation rectifies this problem during disk writes by shifting such a bit from its nominal position in the opposite direction to its known read shift. Write precomp is usually necessary only for data written on the tracks on the inner half of the disk. The tracks on which write pre-compensation should be enabled vary from manufacturer and the number of the track at which write precomp is enabled in the Tandy disk is 22. Write precomp is on for tracks with a number greater than 22. The halt flag is used to enable the FDC board to halt the 6809. This is used to enable the Color Computer to operate the disk drives in double density mode at the low (.89 MHz) clock speed at which the Color Computer

runs. When the halt flag is high, the DRQ (Data ReQuest) signal from the FDC will be connected to the halt input of the 6809. This will allow the DRQ signal to control the operation of the 6809 to the extent that the 6809 will not process any instructions while the FDC is processing data to or from the 6809. Writing a zero to bit 7 of will clear the halt flag \$FF40 or it will be cleared whenever the FDC generates an INTRQ (Interrupt Request) signal, which indicates that the FDC has completed its current command.

Data transfer between the computer and the disk drives through the FDC is accomplished through a series of hardware and software tricks . The slow clock speed of the Color Computer will not allow data to be transferred in the normal method of getting a byte from the computer, giving it to the FDC and then performing status checks until the FDC is not busy. There is just not enough time for this when operating at double density. Part of the Color Computer s solution is a hardware trick whereby the disk controller board will allow the FDC to halt the 68Ø9 while the FDC is storing or retrieving data. The halt flag will allow the DRQ signal from the FDC to halt the 6809 so that the 6809 will wait while the FDC is processing a data request. This trick will allow the 6809 to pass data to the FDC as fast as the FDC can take it by executing a fast loop of: grab a byte from RAM, give it to the FDC and loop back to get another byte. The analogous loop for getting data will also work. The only problem is how to get out of this loop. This problem is solved with software - when an FDC command such as WRITE SECTOR or READ SECTOR is completed an interrupt (INTRQ) is generated by the FDC. The Color Computer connects this INTFQ signal to the Non Maskable Interrupt (NMI) pin of the 68Ø9. This means that whenever an FDC command (except the \$DØ FORCE INTERRUPT) is completed, an NMI will be generated. The computer will now be able to tell that an FDC command is over; all that is left is for the computer to know where to go when the command is finished. This is accomplished by storing a jump vector (DNMIVC) in the disk RAM prior to entering the FDC data transfer loop. Another byte in disk RAM is used as a flag (NMIFLG) to indicate that the NMI jump vector should be used. If the NMIFLG is not equal to zero and an NMI is received by the 6809, Disk Basic will cause the NMIFLG to be reset and control will be transferred to the address in DNMIVC. It is exactly this method, which is used to exit from the FDC data transfer routines.

The FDC has four registers, which are used to communicate with the computer. Their functions are described below:

<u>ADDRESS</u>	<u>READ</u>	<u>WRITE</u>
\$FF48	STATUS REGISTER	COMMAND REGISTER
\$FF49	TRACK REGISTER	TRACK REGISTER
\$FF4A	SECTOR REGISTER	SECTOR REGISTER
\$FF4B	DATA REGISTER	DATA REGISTER

The track and sector registers are used by the FDC to remember where the disk drive s head is currently located. The data register is used to pass data to and from the FDC. The command register is used to pass one of the eleven possible commands to the FDC. Each of these commands has several different forms, which are explained in the FDC data sheet. The form used by Disk Basic are listed below:

<u>TYPE</u>	<u>COMMAND</u>	<u>CODE</u>
I	RESTORE	\$Ø3
I	SEEK	\$17

I	STEP	\$23
I	STEP IN	\$43
I	STEP OUT	\$53
ΙΙ	READ SECTOR	\$8Ø
ΙΙ	WRITE SECTOR	\$AØ
III	READ ADDRESS	\$CØ
III	READ TRACK	\$E4
III	WRITE TRACK	\$F4
ΙV	FORCE INTERRUPT	\$DØ

The status register is used to reflect the results of an FDC command. The contents of the status register will vary depending upon the TYPE of FDC command, which was executed. Listed below are the contents of the status register for the various TYPEs.

# STATUS REGISTER SUMMARY

	ALL TYPE I	READ	READ	READ	WRITE	WRITE
BIT	COMMANDS	ADDRESS	SECTOR	TRACK	SECTOR	TRACK
S7	NOT READY	NOT READY	NOT READY	NOT READY	NOT READY	NOT READY
S6	WRITE	Ø	Ø	Ø	WRITE	WRITE
	PROTECT				PROTECT	PROTECT
S5	HEAD	Ø	RECORD	Ø	WRITE	WRITE
	LOADED		TYPE		FAULT	FAULT
S4	SEEK ERROR	RNF	RNF	Ø	RNF	Ø
\$3	CRC ERROR	CRC ERROR	CRC ERROR	Ø	CRC ERROR	Ø
S2	TRACK Ø	LOST DATA				
SI	INDEX	DR0	DRO	DR0	DRO	DRO
S1	BUSY	BUSY	BUSY	BUSY	BUSY	BUSY

# STATUS FOR TYPE I COMMANDS

BIT NAME	MEANING		
S7 NOT READY	This bit when set indicates the drive is not ready.		
	When reset it indicates that the drive is ready. This		
	bit is an inverted copy of the Ready input and		
	logically or d with MR.		
S6 PROTECTED	When set, indicates Write Protect is activated. This		
	bit is an inverted copy at WRPT input.		
S5 HEAD LOADED	When set, it indicates the head is loaded and engaged.		
	This bit is a logical and of HLD and HLT signals.		
S4 SEEK ERROR	When set, the desired track was not verified. This bit		
	is reset to ∅ when updated.		
S3 CRC ERROR	CRC encountered in ID field.		
S2 TRACK ØØ	When set, indicates Read/Write head is positioned to		
	Track Ø. This bit is an inverted copy of the TRØØ		
	input.		
S1 INDEX	When set, indicates index mark detected from drive.		
	This bit is an inverted copy of the IP input.		
SØ BUSY	When set, command is in progress. When reset, no		
	command is in progress.		

# STATUS FOR TYPE II AND III COMMANDS

BIT NAME	MEANING
S7 NOT READY	This bit when set indicates the drive is not ready.
	When reset, it indicates that the drive is ready. This
	bit is an inverted copy of the Ready input and or d
	with MR. The Type II and Ill Commands will not execute
	unless the drive is ready.
S6 WRITE	On Read Record, Not Used. On Read Track, Not Used. On
PROTECT	any Write: It indicates a Write Protect. This bit is
	reset when updated.
S5 RECORD	On Read Record. It indicates the record type code from
TYPE/WRITE	data field address mark. 1 = Deleted Data Mark. $\emptyset$ =
FAULT	Data Mark. On any Write: It indicates a Write Fault.
	This bit is reset when updated.
S4 RECORD NOT	When set, it indicates that the desired track, sector,
FOUND (RNF)	or side was not found. This bit is reset when updated.
S3 CRC ERROR	If S4 is set, an error is found in one or more ID
	fields: otherwise it indicates error in data field.
	This bit is reset when updated.
S2 LOST DATA	When set, it indicates the computer did not respond to
	DRQ in one byte time. This bit is reset to zero when
	updated.
S1 DATA	This bit is a copy of the DRQ output. When set, it
REQUEST	indicates the DR is full on a Read Operation or the DR
	is empty on a Write operation. This bit Is reset to
	zero when updated.
SØ BUSY	When set, command is under execution. When reset, no
	command Is under execution.

The disk variable DCSTA is not a true reflection of the contents of the FDC status register. Disk Basic filters the status bits of the FDC status register and allow only those errors, which Disk Basic requires to pass through.

### MACHINE LANGUAGE FILE INPUT/OUTPUT

The DOS uses a special format for transferring binary files to and from the disk. The format is fairly simple and straightforward and allows the loading of non contiguous blocks of memory from the same file. The only problem is that Radio Shack has not provided a SAVEM function, which will allow the saving of non contiguous blocks of memory into one disk file. This minor problem can be gotten around with the help of a neat utility called JOIN which is included in the Spectral Associates Color Computer Editor Assembler, ULTRA 8ØCC. This utility will allow the concatenation of as many machine language files as the user requires into one large file. LOADM will then load all of the segments into memory and the segments may overlay one another.

Binary data is stored on the disk as one large block proceeded by a five-byte preamble. The data block is followed by five more bytes which are another preamble if there is another block of data following or the five bytes are a post-amble if there are no further data blocks. The format for the preamble and the post-amble are given below:

<u>BYTE</u>	<u>PREAMBLE</u>	<u>POSTAMBLE</u>
Ø	ØØ Preamble flag	<pre>\$FF Post-amble flag</pre>
1,2	Length of data block	Two zero bytes
3,4	Load address	EXEC address

### DISK BASIC RAM VARIABLES/BUFFERS

Disk Basic requires a substantial amount of RAM for variables and buffer space. There is not enough room in the direct page for all of the variables so Disk Basic grabs a chunk of RAM immediately above the video display RAM (starting at \$600) for its own use. It also uses nine bytes in the direct page, which leaves a total of 17 unused direct page variables for the user when Disk Basic is installed.

At the beginning of Disk RAM are two sector length (256 bytes) I/O buffers, which are primarily used to buffer data transfers to and from the disk controller. DBUFØ is the main I/O buffer and is involved in virtually all disk data transfers. The secondary I/O buffer, DBUF1 is used as a buffer during verify operations and is used as a scratchpad work area or a temporary home for the stack during certain other disk commands such as DSKI\$, DSKO\$ and DSKINI. Following these two buffers are four buffers for the File Allocation Tables and control blocks for the four drives. After these buffers are the variables, which Disk Basic uses for its own internal purposes, and a description of the function of these variables may be found in the direct page memory map found at the beginning of the disassembled list of Disk Basic.

Disk Basic also allocates two additional blocks of RAM for random file buffer and file control block storage. The area for these storage areas is directly after the disk variable RAM and just before the graphic page reserved area. The random file buffer area is used to save a one record length buffer for each active random file. The random file buffer size may be changed with the FILES command. Each time a random file is OPENed or CLOSEd one record length of memory is allocated or deallocated from the available buffer space in the random file buffer area. If there is not enough space in the random file buffer area to hold one record length, an OB (out of buffer space) error will be generated. Immediately after the random file buffer area is the area reserved for file control blocks. The number of available FCBs may be changed by the FILES command. Each FCB requires 281 bytes of RAM and there is always one more FCB (the system FCB) reserved than the number of FCBs requested by the FILES command.

0001	СØØØ	ROMPAK	EQU	\$CØØØ	
0002					
0003	0008	BS	EQU	8	BACKSPACE
0004	ØØØD	CR	EQU	\$D	ENTER KEY
0005	ØØ1B	ESC	EQU	\$1B	ESCAPE CODE
0006	ØØØA	LF	EQU	\$A	LINE FEED
0007	ØØØC	FORMF	EQU	\$C	FORM FEED
ØØØ8	0020	SPACE	EQU	\$20	SPACE (BLANK)
0009					
0010	ØØ3A	STKBUF	EQU	58	STACK BUFFER ROOM
		DEBDEL	EQU	\$45E	
0011	Ø45E		-		DEBOUNCE DELAY
0012	ØØFA	LBUFMX	EQU	250	MAX NUMBER OF CHARS IN A BASIC LINE
0013	ØØFA	MAXLIN	EQU	\$FA	MAXIMUM MS BYTE OF LINE NUMBER
0014					
0015	2600	DOSBUF	EQU	\$2600	RAM LOAD LOCATION FOR THE DOS COMMAND
0013					
	ØØØ6	FATCON	EQU	6	NUMBER OF CONTROL BYTES BEFORE FAT
	0019	FCBCON	EQU	25	NUMBER OF CONTROL BYTES BEFORE FCB
0016	0020	DIRLEN	EQU	32	NUMBER OF BYTES IN DIRECTORY ENTRY
0017	0100	SECLEN	EQU	256	LENGTH OF SECTOR IN BYTES
ØØ18	ØØ12	SECMAX	EQU	18	MAXIMUM NUMBER OF SECTORS PER TRACK
0019	1200	TRKLEN	EQU	SECMAX*SECLEN	LENGTH OF TRACK IN BYTES
0020	ØØ23	TRKMAX	EQU	35	MAX NUMBER OF TRACKS
0021	ØØ4A	FATLEN	EQU	6+(TRKMAX-1)*2	FILE ALLOCATION TABLE LENGTH
0022	0044	GRANMX	EQU	(TRKMAX-1)*2	MAXIMUM NUMBER OF GRANULES
0023	Ø119	FCBLEN	EQU	SECLEN+25	FILE CONTROL BLOCK LENGTH
			-		
0024	0010	INPFIL	EQU	\$10	INPUT FILE TYPE
ØØ25	0020	OUTFIL	EQU	\$20	OUTPUT FILE TYPE
0026	0040	RANFIL	EQU	\$40	RANDOM/DIRECT FILE TYPE
ØØ27					
ØØ35		+ CHDED EV	/TENDEI	D BASIC EQUATES	
	~~.			-	MANAGEMENT NUMBER OF ROLLS IN HIS DES RESULT MORE
0036	0018	ROWMAX	EQU	24	MAXIMUM NUMBER OF ROWS IN HI-RES PRINT MODE
ØØ37	ØØØØ	RAMLINK	EQU	Ø	DUMMY RAM LINK VECTOR
ØØ38	2000	HRESSCRN	EQU	\$2000	ADDRESS OF THE HI-RES SCREEN IN THE CPU'S MEMORY SPACE
ØØ39	CØØØ	HRESBUFF	EQU	\$CØØØ	ADDRESS OF THE GET/PUT BUFFERS IN THE CPU'S MEMORY SPACE
			-		
0040	DFFF	TMPSTACK	EQU	\$DFFF	ADDRESS OF THE HI-RES GRAPHICS STACK IN THE CPU'S MEMORY SPACE
0041	0062	EBHITOK	EQU	\$62	FIRST ENHANCED BASIC TOKEN NUMBER
0042	ØØ29	EBHISTOK	EQU	\$29	FIRST ENHANCED BASIC FUNCTION TOKEN NUMBER BUG - SHOULD BE \$28
ØØ43	ØØ2Ø	CURCHAR	EQU	SPACE	HI-RES CURSOR CHARACTER
45 UU43					
			•		
0044				IT DUEEED HEADED	FOUNTES
0044 0045		* HBUFF HG	ET/HPU	JT BUFFER HEADER	
0044 0045 0046	0000	* HBUFF HG	GET/HPU	Ø	ADDRESS OF THE NEXT BUFFER - 2 BYTES
0044 0045		* HBUFF HG	ET/HPU		
0044 0045 0046 0047	ØØØØ ØØØ2	* HBUFF HG HB.ADDR HB.NUM	GET/HPI EQU EQU	Ø 2	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES
0044 0045 0046 0047 0048	0000 0002 0003	* HBUFF HG HB.ADDR HB.NUM HB.SIZE	GET/HPI EQU EQU EQU	Ø 2 3	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES
0044 0045 0046 0047 0048 0049	ØØØØ ØØØ2	* HBUFF HG HB.ADDR HB.NUM	GET/HPI EQU EQU	Ø 2	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES
0044 0045 0046 0047 0048 0049	0000 0002 0003	* HBUFF HO HB.ADDR HB.NUM HB.SIZE HB.LEN	GET/HPI EQU EQU EQU EQU	Ø 2 3 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES
0044 0045 0046 0047 0048 0049 0050	0000 0002 0003	* HBUFF HO HB.ADDR HB.NUM HB.SIZE HB.LEN * VIDEO RE	GET/HPI EQU EQU EQU EQU EQU	Ø 2 3 5 R EQUATES	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES
0044 0045 0046 0047 0048 0049	0000 0002 0003	* HBUFF HO HB.ADDR HB.NUM HB.SIZE HB.LEN	GET/HPI EQU EQU EQU EQU EQU	Ø 2 3 5 R EQUATES	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES
0044 0045 0046 0047 0048 0049 0050	0000 0002 0003	* HBUFF HO HB.ADDR HB.NUM HB.SIZE HB.LEN * VIDEO RE	GET/HPI EQU EQU EQU EQU EQU	Ø 2 3 5 R EQUATES	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES
0044 0045 0046 0047 0048 0049 0050 0051 0052	0000 0002 0003 0005	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN * VIDEO RE * INITØ BJ COCO	GET/HPI EQU EQU EQU EQU EGISTEI IT EQU/ EQU	Ø 2 3 5 R EQUATES ATES \$80	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053	0000 0002 0003 0005	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN * VIDEO RE * INITØ BJ COCO MMUEN	GET/HPU EQU EQU EQU EQU EQU EQU EQU EQU EQU EQ	Ø 2 3 5 R EQUATES ATES \$80 \$40	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054	0000 0002 0003 0005	* HBUFF HO HB.ADDR HB.NUM HB.SIZE HB.LEN * VIDEO RE * INITØ BI COCO MMUEN IEN	GET/HPI EQU EQU EQU EGISTEI T EQU EQU EQU EQU EQU EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055	0000 0002 0003 0005 0005	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BJ COCO MMUEN IEN FEN	GET/HPI EQU EQU EQU EGISTEI T EQU EQU EQU EQU EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0055 0056 0056	0000 0002 0003 0005 0080 0040 0020 0010 0008	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10 8	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055	0000 0002 0003 0005 0005	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ B3 COCO MMUEN IEN FEN MC3 MC2	GET/HPI EQU EQU EQU EGISTEI T EQU EQU EQU EQU EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0055 0056 0056	0000 0002 0003 0005 0080 0040 0020 0010 0010 0008	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ B3 COCO MMUEN IEN FEN MC3 MC2	GET/HPI EQU	Ø 2 3 5 5 8 EQUATES ATES \$80 \$40 \$20 \$10 8 4	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058	0000 0002 0003 0005 0080 0040 0020 0010 0008 0004 0004	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ B3 COCO MMUEN IEN FEN MC3 MC2 MC1	GET/HPI EQU	Ø 2 3 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059	0000 0002 0003 0005 0080 0040 0020 0010 0010 0008	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ B3 COCO MMUEN IEN FEN MC3 MC2	GET/HPI EQU	Ø 2 3 5 5 8 EQUATES ATES \$80 \$40 \$20 \$10 8 4	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059	0000 0002 0003 0005 0080 0040 0020 0010 0008 0004 0004	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BJ COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ	GET/HPI EQU	Ø 2 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 0060 0061	0000 0002 0003 0005 0005 0040 0020 0010 0008 0004 0002 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BJ COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10 8 4 2 1 JEST ENABLED	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059	0000 0002 0003 0005 0080 0040 0020 0010 0008 0004 0004	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BJ COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ	GET/HPI EQU	Ø 2 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 0060 0061 0062	0000 0002 0003 0005 0005 0040 0020 0010 0008 0004 0004 0002 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF	GET/HPI EQU	Ø 2 3 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0056 0057 0058 0059 0060 0061 0062 0063	0000 0002 0003 0005 0005 0040 0010 0008 0004 0002 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD	GET/HPI EQU	Ø 2 3 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 0060 0061 0062 0063 0064 0065	0000 0002 0003 0005 0080 0040 0020 0010 0008 0004 0002 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD	GET/HPI EQU	Ø 2 3 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 0060 0061 0062 0063 0064 0065	0000 0002 0003 0005 0080 0040 0020 0010 0008 0004 0002 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2	GET/HPI EQU	Ø 2 3 5 5 STEEL ST	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 0060 0061 0062 0063 0064 0065	0000 0002 0003 0005 0005 0040 0040 0010 0008 0004 0002 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI1	GET/HPI EQU	Ø 2 3 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0059 0060 0061 0062 0063 0064 0065	0000 0002 0003 0005 0080 0040 0020 0010 0008 0004 0002 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2	GET/HPI EQU	Ø 2 3 5 5 STEEL ST	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0068 0061 0062 0063 0064 0066 0067 0068	0000 0002 0003 0005 0005 0040 0040 0010 0008 0004 0002 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI1	GET/HPI EQU	Ø 2 3 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0068 0061 0062 0063 0064 0065 0066 0066 0067	0000 0002 0003 0005 0005 0040 0040 0010 0008 0004 0002 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI11 EIØ	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$8Ø \$40 \$20 \$10 8 4 2 1 JEST ENABLED \$20 \$10 8 4 2 1 1	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0060 0061 0062 0063 0064 0065 0066 0066 0066 0066 0066 0066	0000 0002 0003 0005 0005 0040 0010 0008 0004 0002 0001 0008 0004 0008 0008 0004 0008	* HBUFF HO HB.ADDR HB.NUM HB.SIZE HB.LEN * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10 8 4 2 1 JEST ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 4 4 2 1 1 SZ ENABLED \$20 \$10 8 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0061 0062 0063 0064 0065 0066 0067 0068	0000 0002 0003 0005 0080 0040 0010 0008 0004 0002 0001 0008 0001 0008 0004 0008 0004	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUE TMR HBORD VBORD EI2 EI1 EIØ  * EXPANDEE BLOCK 6.0	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$440 \$20 \$10 8 4 2 1 UJEST ENABLED \$20 \$110 8 4 2 2 1 RY DEFINITIONS \$30	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE  BLOCKS \$3Ø-\$33 ARE THE HI-RES GRAPHICS SCREEN
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0060 0061 0062 0063 0064 0065 0066 0066 0066 0066 0066 0066	0000 0002 0003 0005 0005 0040 0010 0008 0004 0002 0001 0008 0004 0008 0008 0004 0008	* HBUFF HO HB.ADDR HB.NUM HB.SIZE HB.LEN * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10 8 4 2 1 JEST ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 2 1 1 SZ ENABLED \$20 \$10 8 4 4 4 2 1 1 SZ ENABLED \$20 \$10 8 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0069 0061 0062 0063 0064 0065 0066 0067 0068 0069 0070 0071	0000 0002 0003 0005 0080 0040 0010 0008 0004 0002 0001 0008 0004 0002 0001 0008	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ  * EXPANDEE BLOCK 6.0 BLOCK 6.1	GET/HPI EQU	Ø 2 3 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE  BLOCKS \$3Ø-\$33 ARE THE HI-RES GRAPHICS SCREEN HI-RES GRAPHICS SCREEN
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0068 0061 0062 0063 0064 0065 0066 0067 0066 0067 0068 0069 0071 0072 0073	0000 0002 0003 0005 0005 0040 0040 0010 0008 0004 0002 0001 0008 0004 0008 0004 0008 0004	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MC0  * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ  * EXPANDEE BLOCK 6.0 BLOCK 6.1 BLOCK 6.1	GET/HPI EQU	Ø 2 3 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE  BLOCKS \$30-\$33 ARE THE HI-RES GRAPHICS SCREEN HI-RES GRAPHICS SCREEN
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0068 0062 0063 0064 0065 0066 0067 0068 0067 0068 0069 0071 0072	0000 0002 0003 0005 0005 0040 0010 0008 0004 0002 0001 0008 0004 0002 0010 0008 0004 0002 0010	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ  * EXPANDEE BLOCK 6.0 BLOCK 6.1 BLOCK 6.2 BLOCK 6.3	GET/HPI EQU	Ø 2 3 5 5	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control  TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE  BLOCKS \$30-\$33 ARE THE HI-RES GRAPHICS SCREEN HI-RES GRAPHICS SCREEN HI-RES GRAPHICS SCREEN HI-RES GRAPHICS SCREEN
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0068 0062 0063 0064 0066 0067 0068 0067 0068 0067 0068	0000 0002 0003 0005 0005 0040 0010 0008 0004 0002 0001 0008 0004 0002 0010 0008 0001 0008 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ  * EXPANDEE BLOCK 6.Ø BLOCK 6.Ø BLOCK 6.3 BLOCK 6.4	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10 8 4 2 1 1 UEST ENABLED \$20 \$10 8 4 2 1 1 CRY DEFINITIONS \$30 \$31 \$32 \$33 \$34	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control ROM map control  TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE  BLOCKS \$30-\$33 ARE THE HI-RES GRAPHICS SCREEN GET/PUT BUFFER
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0060 0062 0063 0064 0065 0066 0066 0067 0066 0067 0068 0071 0072 0073 0074 0075	0000 0002 0003 0005 0005 0040 0010 0008 0004 0002 0001 0008 0004 0002 0010 0008 0004 0002 0010 0008	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ  * EXPANDEE BLOCK 6.Ø BLOCK 6.Ø BLOCK 6.1 BLOCK 6.2 BLOCK 6.3 BLOCK 6.4 BLOCK 6.5	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10 8 4 2 1 USEST ENABLED \$20 \$10 8 4 2 1 1 CRY DEFINITIONS \$30 \$31 \$32 \$33 \$34 \$35	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control ROM map control  TIMER HORIZONTAL BORDER VERTICAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE  BLOCKS \$3Ø-\$33 ARE THE HI-RES GRAPHICS SCREEN GET/PUT BUFFER STACK AREA FOR HI-RES GRAPHICS COMMAND
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0068 0062 0063 0064 0066 0067 0068 0067 0068 0067 0068	0000 0002 0003 0005 0005 0040 0010 0008 0004 0002 0001 0008 0004 0002 0010 0008 0001 0008 0001	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ  * EXPANDEE BLOCK 6.Ø BLOCK 6.Ø BLOCK 6.3 BLOCK 6.4	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10 8 4 2 1 1 UEST ENABLED \$20 \$10 8 4 2 1 1 CRY DEFINITIONS \$30 \$31 \$32 \$33 \$34	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control ROM map control  TIMER HORIZONTAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE  BLOCKS \$30-\$33 ARE THE HI-RES GRAPHICS SCREEN GET/PUT BUFFER
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0061 0062 0063 0064 0065 0066 0067 0068 0067 0068 0070 0071 0072 0075 0076 0077	0000 0002 0003 0005 0005 00005 00005 0000 0000	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ  * EXPANDEE BLOCK 6.0 BLOCK 6.1 BLOCK 6.2 BLOCK 6.4 BLOCK 6.5 BLOCK 6.5 BLOCK 6.6	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$440 \$20 \$10 8 4 2 1 USEST ENABLED \$20 \$110 8 4 2 2 1 1 RY DEFINITIONS \$30 \$31 \$32 \$33 \$34 \$35 \$36	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control  TIMER HORIZONTAL BORDER VERTICAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE  BLOCKS \$3Ø-\$33 ARE THE HI-RES GRAPHICS SCREEN GET/PUT BUFFER STACK AREA FOR HI-RES GRAPHICS COMMAND CHARACTER POINTERS
0044 0045 0046 0047 0048 0049 0050 0051 0052 0053 0054 0055 0056 0057 0058 0060 0062 0063 0064 0065 0066 0066 0067 0066 0067 0068 0071 0072 0073 0074 0075	0000 0002 0003 0005 0005 0040 0010 0008 0004 0002 0001 0008 0004 0002 0010 0008 0004 0002 0010 0008	* HBUFF HG HB.ADDR HB.NUM HB.SIZE HB.LEN  * VIDEO RE * INITØ BI COCO MMUEN IEN FEN MC3 MC2 MC1 MCØ  * INTERRUF TMR HBORD VBORD EI2 EI1 EIØ  * EXPANDEE BLOCK 6.Ø BLOCK 6.Ø BLOCK 6.1 BLOCK 6.2 BLOCK 6.3 BLOCK 6.4 BLOCK 6.5	GET/HPI EQU	Ø 2 3 5 5 R EQUATES ATES \$80 \$40 \$20 \$10 8 4 2 1 USEST ENABLED \$20 \$10 8 4 2 1 1 CRY DEFINITIONS \$30 \$31 \$32 \$33 \$34 \$35	ADDRESS OF THE NEXT BUFFER - 2 BYTES NUMBER OF THIS BUFFER - 1 BYTES NUMBER OF BYTES IN THE BUFFER - 2 BYTES NUMBER OF BYTES IN THIS HEADER  1 = Color Computer compatible 1 = MMU enabled 1 = GIME chip IRQ output enabled 1 = GIME chip FIRQ output enabled 1 = RAM at XFEXX is constant 1 = standard SCS ROM map control ROM map control ROM map control  TIMER HORIZONTAL BORDER VERTICAL BORDER VERTICAL BORDER SERIAL DATA KEYBOARD CARTRIDGE  BLOCKS \$3Ø-\$33 ARE THE HI-RES GRAPHICS SCREEN GET/PUT BUFFER STACK AREA FOR HI-RES GRAPHICS COMMAND

0080		* BLOCKS	\$48-\$41	F ARE USED FOR	THE BASIC OPERATING SYSTEM
0081	0038	BLOCK7.Ø	EQU	\$38	
ØØ82	ØØ39	BLOCK7.1	EQU	\$39	
ØØ83	ØØ3A	BLOCK7.2	EQU	\$3A	
ØØ84	ØØ3B	BLOCK7.3	EQU	\$3B	
ØØ85	ØØ3C	BLOCK7.4	EQU	\$3C	
			-		
ØØ86	ØØ3D	BLOCK7.5	EQU	\$3D	
ØØ87	ØØ3E	BLOCK7.6	EQU	\$3E	
ØØ88	ØØ3F	BLOCK7.7	EQU	\$3F	
0089					
0028		* PSEUDO	PSEUD0	OPS	
ØØ29	0021	SKP1	EQU	\$21	OP CODE OF BRN SKIP ONE BYTE
0030	ØØ8C	SKP2	EQU	\$8C	OP CODE OF CMPX # - SKIP TWO BYTES
ØØ31	ØØ86	SKP1LD	EQU	\$86	OP CODE OF LDA # - SKIP THE NEXT BYTE
ØØ32	2200	*	LQU	400	AND LOAD THE VALUE OF THAT BYTE INTO ACCA THIS
ØØ32		*			IS USUALLY USED TO LOAD ACCA WITH A NON ZERO VALUE
		•			13 030ALLI 03LD TO LOAD ACCA WITH A NOW ZERO VALUE
0090					
0091				_	
0092 0000			ORG	Ø	
0093	0000		SETDP	Ø	
0094					
0095 0000		ENDFLG	RMB	1	STOP/END FLAG: POSITIVE=STOP, NEG=END
0096 0001		CHARAC	RMB	1	TERMINATOR FLAG 1
0097 0002		ENDCUR	RMB	1	TERMINATOR FLAG 2
0098 0003		TMPLOC	RMB	1	SCRATCH VARIABLE
0099 0004		IFCTR	RMB	1	IF COUNTER - HOW MANY IF STATEMENTS IN A LINE
0100 0005					
		DIMFLG	RMB	1	*DV* ARRAY FLAG Ø=EVALUATE, 1=DIMENSIONING
0101 0006		VALTYP	RMB	1	*DV* *PV TYPE FLAG: Ø=NUMERIC, \$FF=STRING
0102 0007		GARBFL	RMB	1	*TV STRING SPACE HOUSEKEEPING FLAG
0103 0008		ARYDIS	RMB	1	DISABLE ARRAY SEARCH: ØØ=ALLOW SEARCH
0104 0009		INPFLG	RMB	1	*TV INPUT FLAG: READ=Ø, INPUT∽Ø
Ø1Ø5 ØØØA		RELFLG	RMB	1	*TV RELATIONAL OPERATOR FLAG
Ø1Ø6 ØØØB		TEMPPT	RMB	2	*PV TEMPORARY STRING STACK POINTER
Ø1Ø7 ØØØD		LASTPT	RMB	2	*PV ADDR OF LAST USED STRING STACK ADDRESS
Ø1Ø8 ØØØF		TEMPTR	RMB	2	TEMPORARY POINTER
Ø1Ø9 ØØ11		TMPTR1	RMB	2	TEMPORARY DESCRIPTOR STORAGE (STACK SEARCH)
Ø11Ø					#2 (MANTISSA ONLY)
0111 0013		FPA2	RMB	4	FLOATING POINT ACCUMULATOR #2 MANTISSA
0112 0017		BOTSTK	RMB	2	BOTTOM OF STACK AT LAST CHECK
0113 0019		TXTTAB	RMB	2	*PV BEGINNING OF BASIC PROGRAM
Ø114 ØØ1B		VARTAB	RMB	2	*PV START OF VARIABLES
Ø115 ØØ1D		ARYTAB	RMB	2	*PV START OF ARRAYS
Ø116 ØØ1F		ARYEND	RMB	2	*PV END OF ARRAYS (+1)
0117 0021		FRETOP	RMB	2	*PV START OF STRING STORAGE (TOP OF FREE RAM)
0118 0023		STRTAB	RMB	2	*PV START OF STRING VARIABLES
Ø119 ØØ25		FRESPC	RMB	2	UTILITY STRING POINTER
Ø12Ø ØØ27		MEMSIZ	RMB	2	*PV TOP OF STRING SPACE
				2	
0121 0029		OLDTXT	RMB		SAVED LINE NUMBER DURING A "STOP"
Ø122 ØØ2B		BINVAL	RMB	2	BINARY VALUE OF A CONVERTED LINE NUMBER
Ø123 ØØ2D		OLDPTR	RMB	2	SAVED INPUT PTR DURING A "STOP"
Ø124 ØØ2F		TINPTR	RMB	2	TEMPORARY INPUT POINTER STORAGE
0125 0031		DATTXT	RMB	2	*PV 'DATA' STATEMENT LINE NUMBER POINTER
Ø126 ØØ33		DATPTR	RMB	2	*PV 'DATA' STATEMENT ADDRESS POINTER
0127 0035		DATTMP	RMB	2	DATA POINTER FOR 'INPUT' & 'READ'
Ø128 ØØ37		VARNAM	RMB	2	*TV TEMP STORAGE FOR A VARIABLE NAME
0129 0039		VARPTR	RMB	2	*TV POINTER TO A VARIABLE DESCRIPTOR
Ø13Ø ØØ3B		VARDES	RMB	2	TEMP POINTER TO A VARIABLE DESCRIPTOR
Ø131 ØØ3D		RELPTR	RMB	2	POINTER TO RELATIONAL OPERATOR PROCESSING ROUTINE
Ø131 ØØ35		TRELFL	RMB	1	TEMPORARY RELATIONAL OPERATOR FLAG BYTE
		INCLIL	מויוא	*	TEH ORAKI KELATIONAL OFERATOR TEAU DITE
Ø133		+ FI O A T T 11	0 0071	E ACCUMULATORS	#2 A R E ADE MOCTLY
Ø134					#3,4 & 5 ARE MOSTLY
Ø135				CH PAD VARIABL	
Ø136					#3 :PACKED: (\$40-\$44)
0137 0040		V4Ø	RMB	1	
0138 0041		V41	RMB	1	
0139 0042		V42	RMB	1	
0140 0043		V43	RMB	1	
0141 0044		V44	RMB	1	
Ø142					#4 :PACKED: (\$45-\$49)
Ø143 ØØ45		V45	RMB	1	
0144 0046		V45 V46	RMB	1	
0145 0047		V47	RMB	1	
0146 0048		V48	RMB	2	WE DIOWED (**** ****)
Ø147		** FLOATI	NG POIN	NI ACCUMULATOR	#5 :PACKED: (\$4A \$4E)

Ø148	004A	V4A	RMB	1			
Ø149	ØØ4B	V4B	RMB	2			
Ø15Ø	ØØ4D	V4D	RMB	2			
	טדט				ATOD .	μα	
Ø151		** FLOATIN			AIUK		
Ø152	ØØ4F	FPØEXP	RMB	1		*PV	FLOATING POINT ACCUMULATOR #Ø EXPONENT
Ø153	ØØ5Ø	FPAØ	RMB	4		*PV	FLOATING POINT ACCUMULATOR #Ø MANTISSA
	0054	FPØSGN	RMB				FLOATING POINT ACCUMULATOR #Ø SIGN
Ø155	0055	COEFCT	RMB	1		POLY	NOMIAL COEFFICIENT COUNTER
Ø156	ØØ56	STRDES	RMB	5		TEMP	ORARY STRING DESCRIPTOR
Ø157		FPCARY		1			TING POINT CARRY BYTE
	9935						TING FOINT CARRY BITE
Ø158		** FLOATIN			AIUR :		
Ø159	ØØ5C	FP1EXP	RMB	1		*PV	FLOATING POINT ACCUMULATOR #1 EXPONENT
Ø16Ø	ØØ5D	FPA1	RMB	4		*PV	FLOATING POINT ACCUMULATOR #1 MANTISSA
Ø161		FP1SGN		1			FLOATING POINT ACCUMULATOR #1 SIGN
	ווטשש	IFIJUN	KIID	1		F V	I LUATING FUINT ACCUMULATOR #1 31GN
Ø162							
Ø163	0062	RESSGN	RMB	1		SIGN	OF RESULT OF FLOATING POINT OPERATION
Ø164	0063	FPSBYT	RMB	1		FIOA	TING POINT SUB BYTE (FIFTH BYTE)
	0064	COEFPT		2			NOMIAL COEFFICIENT POINTER
Ø166	0066	LSTTXT	RMB	2		CURR	RENT LINE POINTER DURING LIST
Ø167	ØØ68	CURLIN	RMB	2		*PV	CURRENT LINE # OF BASIC PROGRAM, \$FFFF = DIRECT
Ø168		DEVCFW		1			TAB FIELD WIDTH
Ø169		DEVLCF		1			TAB ZONE
Ø17Ø	006C	DEVPOS	RMB	1		*TV	PRINT POSITION
Ø171	ØØ6D	DEVWID	RMB	1		*TV	PRINT WIDTH
Ø172		PRTDEV		1			PRINT DEVICE: Ø=NOT CASSETTE, -1=CASSETTE
Ø173	וטשט	DEVNUM	RMB	1		*PV	DEVICE NUMBER: -3=DLOAD, -2=PRINTER,
Ø174		*					-1=CASSETTE, Ø=SCREEN, 1-15=DISK
Ø175	0070	CINBFL	RMB	1		*PV	CONSOLE IN BUFFER FLAG: ØØ=NOT EMPTY, \$FF=EMPTY
Ø176				1			WARM START FLAG: \$55=WARM, OTHER=COLD
		RSTFLG					
Ø177	0072	RSTVEC	RMB	2		*PV	WARM START VECTOR - JUMP ADDRESS FOR WARM START
Ø178	0074	TOPRAM	RMB	2		*PV	TOP OF RAM
Ø179			RMB	2			RE: UNUSED VARIABLES
		ETI CTA					
Ø18Ø		FILSTA		1			FILE STATUS FLAG: Ø=CLOSED, 1=INPUT, 2=OUTPUT
Ø181	ØØ79	CINCTR	RMB	1		*PV	CONSOLE IN BUFFER CHAR COUNTER
Ø182	ØØ7A	CINPTR	RMB	2		*PV	CONSOLE IN BUFFER POINTER
Ø183		BLKTYP		1			CASS BLOCK TYPE: Ø=HEADER, 1=DATA, \$FF=EOF
Ø184	007D	BLKLEN	RMB	1		*10	CASSETTE BYTE COUNT
Ø185	ØØ7E	CBUFAD	RMB	2		*TV	CASSETTE LOAD BUFFER POINTER
Ø186	aasa	CCKSUM	RMB	1			CASSETTE CHECKSUM BYTE
Ø187		CSRERR		1			ERROR FLAG/CHARACTER COUNT
Ø188	ØØ82	CPULWD	RMB	1		*TV	PULSE WIDTH COUNT
Ø189	ØØ83	CPERTM	RMB	1		*TV	BIT COUNTER
	ØØ84	СВТРНА		1			BIT PHASE FLAG
Ø191	ØØ85	CLSTSN	RMB	1			LAST SINE TABLE ENTRY
Ø192	ØØ86	GRBLOK	RMB	1		*TV	GRAPHIC BLOCK VALUE FOR SET, RESET AND POINT
Ø193	ØØ87	IKEYIM	RMB	1		*TV	INKEY\$ RAM IMAGE
	ØØ88	CURPOS		2			CURSOR LOCATION
0195	ØØ8A	ZER0	RMB	2		*PV	DUMMY - THESE TWO BYTES ARE ALWAYS ZERO
Ø196	ØØ8C	SNDTON	RMB	1		*TV	TONE VALUE FOR SOUND COMMAND
Ø197		SNDDUR	RMB	2			DURATION VALUE FOR SOUND COMMAND
Ø198	DD 0 D	SINDBOIL	MID	-			DOINTION THESE FOR SOONS CONTINUE
		44 TUESE -	VT-0 -	DE 40055	DC: 111		OM
Ø199		** THESE B	YIES A	KE MOVED	DOMN		
0200		***				INIT	DESCRIPTION
0201		*				VALU	
	ØØ8F	CMPMID	RMB	1		18	*PV 1200/2400 HERTZ PARTITION
	0090	CMPØ		1		24	*PV UPPER LIMIT OF 1200 HERTZ PERIOD
0204	0091	CMP1	RMB	1		10	*PV UPPER LIMIT OF 2400 HERTZ PERIOD
0205	0092	SYNCLN	RMB	2		128	*PV NUMBER OF \$55'S TO CASSETTE LEADER
	0094					11	*PV CURSOR BLINK DELAY
		BLKCNT		1			
0207		LPTBTD	RMB	2		88	*PV BAUD RATE CONSTANT (600)
0208	ØØ97	LPTLND	RMB	2		1	*PV PRINTER CARRIAGE RETURN DELAY
	ØØ99	LPTCFW		1		16	*PV TAB FIELD WIDTH
	ØØ9A	LPTLCF		1		112	*PV LAST TAB ZONE
0211	ØØ9B	LPTWID	RMB	1		132	*PV PRINTER WIDTH
Ø212	ØØ9C	LPTPOS	RMB	1		Ø	*PV LINE PRINTER POSITION
	ØØ9D	EXECJP		2			A *PV JUMP ADDRESS FOR EXEC COMMAND
		LALGUE	WILD	_		LD4P	W 1 A DOWN WARE 23 LOW EVER COLINAINA
0214							
Ø215		** THIS RO	UTINE	PICKS UP	THE N	EXT IN	PUT CHARACTER FROM
Ø216		** BASIC.	THE AD	DRESS OF	THE N	EXT BA	SIC BYTE TO BE
Ø217		** INTERPR					and and a second second
		THIERPK	LILU I	3 STUKED	AT CH	ANAD.	
Ø218							
Ø219	ØØ9F ØC A7	GETNCH	INC	<charad+1< td=""><td>l</td><td>*PV</td><td>INCREMENT LS BYTE OF INPUT POINTER</td></charad+1<>	l	*PV	INCREMENT LS BYTE OF INPUT POINTER
	ØØA1 26 Ø2			GETCCH			BRANCH IF NOT ZERO (NO CARRY)
	ØØA3 ØC A6			<charad< td=""><td></td><td></td><td>INCREMENT MS BYTE OF INPUT POINTER</td></charad<>			INCREMENT MS BYTE OF INPUT POINTER
מררו	טא טע כחעע		1110	-CHARAD		ΓV	THOUSENERS AND DISE OF THEOL LOTHICK

Ø222	ØØA5	В6		GETCCH	FCB	\$B6	*PV OP CODE OF LDA EXTENDED
Ø223	ØØA6			CHARAD		2	*PV THESE 2 BYTES CONTAIN ADDRESS OF THE CURRENT
Ø224				*			CHARACTER WHICH THE BASIC INTERPRETER IS
Ø225				*			PROCESSING
	ØØA8	75 1/	۱ 1 ۸		JMP	BROMHK	JUMP BACK INTO THE BASIC RUM
	סאשש	/ E A/	4 IA		UNF	אחויוחא	JUMP DACK INTO THE DASIC RUM
Ø227	00 A D			VAD	DMD		LOW ORDER FOUR BYTES OF THE BRODUST
	ØØAB			VAB	RMB	1	= LOW ORDER FOUR BYTES OF THE PRODUCT
	ØØAC			VAC	RMB	1	= OF A FLOATING POINT MULTIPLICATION
Ø23Ø	ØØAD			VAD	RMB	1	= THESE BYTES ARE USE AS RANDOM DATA
Ø231	ØØAE			VAE	RMB	1	= BY THE RND STATEMENT
Ø232							
Ø233				* EXTENDED	BASI	C VARIABLES	
	ØØAF			TRCFLG	RMB	1	*PV TRACE FLAG Ø=OFF ELSE=ON
	ØØBØ			USRADR	RMB	2	*PV ADDRESS OF THE START OF USR VECTORS
	ØØB2				RMB		*PV FOREGROUND COLOR
				FORCOL		1	
	ØØB3			BAKCOL	RMB	1	*PV BACKGROUND COLOR
	ØØB4			WCOLOR	RMB	1	*TV WORKING COLOR BEING USED BY EX BASIC
	ØØB5			ALLCOL	RMB	1	*TV ALL PIXELS IN THIS BYTE SET TO COLOR OF VB3
0240	ØØB6			PMODE	RMB	1	*PV PMODE'S MODE ARGUMENT
Ø241	ØØB7			ENDGRP	RMB	2	*PV END OF CURRENT GRAPHIC PAGE
Ø242	ØØB9			HORBYT	RMB	1	*PV NUMBER OF BYTES/HORIZONTAL GRAPHIC LINE
Ø243	ØØBA			BEGGRP	RMB	2	*PV START OF CURRENT GRAPHIC PAGE
0244	ØØBC			GRPRAM	RMB	1	*PV START OF GRAPHIC RAM (MS BYTE)
	ØØBD			HORBEG	RMB	2	*DV* *PV HORIZ COORD - START POINT
	ØØBF			VERBEG	RMB	2	*DV* *PV VERT COORD - START POINT
	ØØC1			CSSYAL	RMB	1	*PV SCREEN'S COLOR SET ARGUMENT
	ØØC2				RMB	1	
				SETFLG			*PV PRESET/PSET FLAG: Ø=PRESET, 1=PSET
	ØØC3			HOREND	RMB	2	*DV* *PV HORIZ COORD - ENDING POINT
	ØØC5			VEREND	RMB	2	*DV* *PV VERT COORD - ENDING POINT
	ØØC7			HORDEF	RMB	2	*PV HORIZ COORD - DEFAULT COORD
Ø252	ØØC9			VERDEF	RMB	2	*PV VERT COORD - DEFAULT COORD
Ø253							
Ø254				* EXTENDED	BASI	C SCRATCH PAD VA	RIABLES
0255	ØØCB			VCB	RMB	2	
	ØØCD			VCD	RMB	2	
	ØØCF			VCF	RMB	2	
	ØØD1			VD1	RMB	2	
	ØØD3			VD3	RMB	1	
	ØØD4			VD4	RMB	1	
Ø261	ØØD5			VD5	RMB	1	
Ø262	ØØD6			VD6	RMB	1	
Ø263	ØØD7			VD7	RMB	1	
Ø264	ØØD8			VD8	RMB	1	
Ø265	ØØD9			VD9	RMB	1	
	ØØDA			VDA	RMB	1	
Ø267	DDDA			• 571	IIIID	-	
	ØØDB			CHGFLG	RMB	1	*TV FLAG TO INDICATE IF GRAPHIC DATA HAS BEEN CHANGED
	ØØDC						
				TMPSTK	RMB	2	*TV STACK POINTER STORAGE DURING PAINT
	ØØDE			OCTAVE	RMB	1	*PV OCTAVE VALUE (PLAY)
	ØØDF			VOLHI	RMB	1	*DV* *PV VOLUME HIGH VALUE (PLAY)
	ØØEØ			VOLLOW	RMB	1	*DV* *PV VOLUME LOW VALUE (PLAY)
	ØØE1			NOTELN	RMB	1	*PV NOTE LENGTH (PLAY)
	ØØE2			TEMP0	RMB	1	*PV TEMPO VALUE (PLAY)
	ØØE3			PLYTMR	RMB	2	*TV TIMER FOR THE PLAY COMMAND
Ø276	ØØE5			DOTYAL	RMB	1	*TV DOTTED NOTE TIMER SCALE FACTOR
	ØØE6			HRMODE	EQU	*	SUPER EXTENDED BASIC HI-RES MODE
	ØØE6			DLBAUD	RMB	1	*DV* *PV DLOAD BAUD RATE CONSTANT \$BØ=3ØØ, \$2C=12ØØ
	ØØE7			HRWIDTH	EQU	*	SUPER EXTENDED BASIC HI-RES TEXT MODE
	ØØE7			TIMOUT	RMB	1	*DV* *PV DLOAD TIMEOUT CONSTANT
	ØØE8					1	*DV* *PV ANGLE VALUE (DRAW)
				ANGLE	RMB		
	ØØE9			SCALE	RMB	1	*DV* *PV SCALE VALUE (DRAW)
Ø283							
Ø284				* DSKCON V	ARIABI	LES	
Ø285	ØØEA			DCOPC	RMB	1	*PV DSKCON OPERATION CODE Ø-3
Ø286	ØØEB			DCDRV	RMB	1	*PV DSKCON DRIVE NUMBER Ø 3
				DCTRK	RMB	1	*PV DSKCON TRACK NUMBER Ø 34
ע משע	ØØEC			DSEC	RMB	1	*PV DSKCON SECTOR NUMBER 1-18
	ØØEC ØØED						
Ø288	ØØED				RMR	2	*PV DSKCON DATA POINTER
Ø288 Ø289	ØØED ØØEE			DCBPT	RMB DMB	2	*PV DSKCON DATA POINTER  *PV DSKCON STATUS RYTE
Ø288 Ø289 Ø29Ø	ØØED ØØEE ØØFØ				RMB RMB	2	*PV DSKCON DATA POINTER *PV DSKCON STATUS BYTE
Ø288 Ø289 Ø29Ø Ø291	ØØED ØØEE ØØFØ			DCBPT DCSTA	RMB	1	*PV DSKCON STATUS BYTE
Ø288 Ø289 Ø29Ø Ø291 Ø292	ØØED ØØEE ØØFØ			DCBPT			
0288 0289 0290 0291 0292 0293	00ED 00EE 00F0 00F1			DCBPT DCSTA	RMB RMB	1 2	*PV DSKCON STATUS BYTE TEMPORARY FCB POINTER
0288 0289 0290 0291 0292 0293 0294	ØØED ØØEE ØØFØ			DCBPT DCSTA	RMB	1	*PV DSKCON STATUS BYTE
0288 0289 0290 0291 0292 0293	00ED 00EE 00F0 00F1			DCBPT DCSTA	RMB RMB	1 2	*PV DSKCON STATUS BYTE TEMPORARY FCB POINTER

<b>700</b> 5			
0296	*		DACIC EVDACI/DOCDACIO
Ø297 Ø298	^		BASIC EXBASI(DOSBASIC
0299 0100	SW3VEC RMB	3	\$XXXX \$XXXX \$3B3B SWI3 VECTOR
0300 0103	SW2VEC RMB	3	\$XXXX \$XXXX \$3B3B SWI2 VECTOR
Ø3Ø1 Ø1Ø6	SWIVEC RMB	3	\$XXXX \$XXXX \$XXXX SWI VECTOR
0302 0109	NMIVEC RMB	3	\$XXXX \$XXXX \$D7AE NMI VECTOR
Ø3Ø3 Ø1ØC	IRQVEC RMB	3	\$A9B3 \$894C \$D7BC IRQ VECTOR
Ø3Ø4 Ø1ØF	FRQVEC RMB	3	\$AØF6 \$AØF6 \$AØF6 FIRQ VECTOR
Ø3Ø5			
0306 0112	TIMVAL		
0307 0112	USRJMP RMB	3	JUMP ADDRESS FOR BASIC'S USR FUNCTION
Ø3Ø8	* RMB	2	TIMER VALUE FOR EXBAS
0309	* RMB	1	UNUSED BY EXBAS OR DISK BASIC
0310 0115	RVSEED RMB	1	* FLOATING POINT RANDOM NUMBER SEED EXPONENT
Ø311 Ø116	RMB	4	* MANTISSA: INITIALLY SET TO \$804FC75259
Ø312 Ø11A	CASFLG RMB	1	UPPER CASE/LOWER CASE FLAG: \$FF=UPPER, Ø=LOWER
Ø313 Ø11B	DEBVAL RMB	2	KEYBOARD DEBOUNCE DELAY (SET TO \$45E)
Ø314 Ø11D	EXPJMP RMB	3	JUMP ADDRESS FOR EXPONENTIATION
Ø315	**		INITIALLY SET TO ERROR FOR BASIC, \$8489 FOR EX BASIC
Ø316	*** COMMA	ND INTERDRETATIO	NU VECTOR TARLE
0317	*** CUMMA	ND INTERPRETATION	JN VECTUR TABLE
0318	44 EOND CEIC OF	10 DVTF TADLEC	
0319	** FOUR SETS OF	IN BAIE LABEE2:	
Ø32Ø Ø321			
Ø322	** TUE   ACT   CE	TADIE MIICT DE	FOLLOWED BY A ZERO BYTE
Ø323			ND 8,9) POINT TO THE JUMP TABLE FOR
Ø324			R TABLES, THESE VECTORS POINT TO A
Ø325			TO THE CORRECT JUMP TABLE.
Ø326			FIED THIS SCHEME SO THAT THE USER
Ø327			NY ADDITIONAL TABLES WILL HAVE TO BE
Ø328	* ACCESSED FROM		
Ø329			
Ø33Ø	* BYTE	DESCRIPTION	
Ø331	* Ø	NUMBER OF RESER	VED WORDS
Ø332	* 1,2	LOOKUP TABLE OF	RESERVED WORDS
Ø333	* 3,4	JUMP TABLE FOR	COMMANDS (FIRST TABLE)
Ø334	*	VECTOR TO EXPAN	SION COMMAND HANDLERS (ALL BUT FIRST TABLE)
Ø335	* 5	NUMBER OF SECON	DARY FUNCTIONS
Ø336	* 6,7	LOOKUP TABLE OF	SECONDARY FUNCTIONS (FIRST TABLE)
Ø337	*		SION SECONDARY COMMAND HANDLERS (ALL BUT
Ø338	*	FIRST TABLE)	
Ø339	* 8,9		SECONDARY FUNCTIONS
0340	* 10	Ø BYTE - END OF	TABLE FLAG (LAST TABLE ONLY)
0341	2011/50 81/8		DIOTOLO TIDIF
0342 0120	COMVEC RMB	10	BASIC'S TABLE
0343 012A	RMB	10	EX BASIC'S TABLE
0344 0134	RMB	10	DISC BASIC'S TABLE (UNUSED BY EX BASIC)
Ø345 Ø346	**** HCD FHNCTIO	ON VECTOR ADDRES	SES (EX BASIC ONLY)
Ø347 Ø13E		ON VECTOR ADDRES	USR Ø VECTOR
0348 0140	RMB RMB	2	USR 1
0349 0142	RMB	2	USR 2
Ø35Ø Ø144	RMB	2	USR 3
Ø351 Ø146	RMB	2	USR 4
Ø352 Ø148	RMB	2	USR 5
Ø353 Ø14A	RMB	2	USR 6
Ø354 Ø14C	RMB	2	USR 7
Ø355 Ø14E	RMB	2	USR 8
Ø356 Ø15Ø	RMB	2	USR 9
Ø357			
Ø358	*** THE ABOVE 20	BYTE USR ADDR	VECTOR TABLE IS MOVED TO
0359	*** \$95F-\$972 B	Y DISC BASIC. TH	E 20 BYTES FROM \$13E-\$151
0360	*** ARE REDEFINE	ED AS FOLLOWS:	
0361			
0362	* RMB	10	USER (SPARE) COMMAND INTERPRETATION TABLE SPACE
Ø363	* FCB	Ø	END OF COMM INTERP TABLE FLAG
Ø364	* RMB	9	UNUSED BY DISK BASIC
Ø365			
Ø366		ND INTERPRETATION	
Ø367	*	BYTE	BASIC EX BASIDISK BASIC
Ø368	*	Ø	53 BASIC TABLE
Ø369	^	1,2	\$AA66

Ø37Ø	* 3,4	\$AB67
Ø371	* 5	20
Ø372	* 6,7	\$AB1A
Ø373	* 8,9	\$AA29
Ø374	0,3	TANKES
	* α	OF TV DASIC TABLE
0375		25 EX BASIC TABLE
Ø376	* 1,2	\$8183
Ø377	* 3,4	\$813C
Ø378	* 5	14
Ø379	* 6,7	\$821E
Ø38Ø	* 8,9	\$8168
Ø381	-,-	
Ø382	* Ø	19 (20 2.1) DISK BASIC TABLE
Ø383	~	\$C17F
	-,-	
Ø384	3,4	\$C2CØ
Ø385	* 5	6
Ø386	* 6,7	\$C2Ø1
Ø387	* 8,9	\$C236
Ø388		
Ø389		
Ø39Ø Ø152	KEYBUF RMB 8	KEYBOARD MEMORY BUFFER
Ø391 Ø15A	POTVAL RMB 1	LEFT VERTICAL JOYSTICK DATA
Ø392 Ø15B	RMB 1	LEFT HORIZONTAL JOYSTICK DATA
Ø393 Ø15C	RMB 1	RIGHT VERTICAL JOYSTICK DATA
Ø394 Ø15D	RMB 1	RIGHT HORIZONTAL JOYSTICK DATA
Ø395		
Ø396		- INITIALIZED TO RTS BY COLOR BASIC
Ø397	* 25 SETS OF 3 BYTE	NSTRUCTIONS WHICH ARE CALLED BY COLOR BASIC
Ø398	* EXTENDED AND DISK	ASIC. THEIR PURPOSE IS TO ALLOW ENHANCEMENTS (SUCH
Ø399	* AS EX BASIC AND DO	BASIC) AS MORE ROMS ARE ADDED TO THE
0400	* SYSTEM BY EFFECTIV	LY ALLOWING MORE CODE TO BE ADDED TO THE
0401		ROMS. THIS NEW CODE IS LOCATED IN THE NEW ROMS
0402		GET TO THE NEW CODE IS IN BYTES 1 & 2 OF THE
0403		WILL CONTAIN A \$7E WHICH IS THE FIRST BYTE OF
0404	* THE JMP INSTRUCTION	
0405	* THE FIRST ADDRESS	N THIS TABLE IS THE ADDRESS IN BASIC WHICH
0406	* CALLS THE RAM VECT	R, THE SECOND ADDRESS IS THE VALUE WHICH
0407	* EX BASIC PUTS IN T	E RAM VECTOR (IF ANY) AND THE THIRD ADDRESS
Ø4Ø8	* IS THE VALUE WHICH	DISK BASIC PUTS THERE (IE ANY)
Ø4Ø8	* IS THE VALUE WHICH	DISK BASIC PUTS THERE (IF ANY)
0409	* IS THE VALUE WHICH	DISK BASIC PUTS THERE (IF ANY)
Ø4Ø9 Ø41Ø		
Ø4Ø9 Ø41Ø Ø411	*	2.0 2.1 1.0 1.1
0409 0410 0411 0412 015E	* RVECØ RMB 3	2.Ø 2.1 1.Ø 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND
Ø4Ø9 Ø41Ø Ø411	*	2.0 2.1 1.0 1.1
0409 0410 0411 0412 015E	* RVECØ RMB 3	2.Ø 2.1 1.Ø 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND
0409 0410 0411 0412 015E 0413 0161 0414 0164	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3 RVEC3 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3 RVEC3 RMB 3 RVEC3 RMB 3 RVEC4 RMB 3	2.Ø 2.1 1.Ø 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3 RVEC3 RMB 3 RVEC4 RMB 3 RVEC5 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$CB4A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3 RVEC3 RMB 3 RVEC4 RMB 3 RVEC5 RMB 3 RVEC5 RMB 3 RVEC6 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C844 \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3 RVEC3 RMB 3 RVEC4 RMB 3 RVEC5 RMB 3 RVEC6 RMB 3 RVEC6 RMB 3 RVEC6 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C844 \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C58C CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3 RVEC3 RMB 3 RVEC4 RMB 3 RVEC5 RMB 3 RVEC5 RMB 3 RVEC6 RMB 3 RVEC7 RMB 3 RVEC7 RMB 3 RVEC7 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C889 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C5BF \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3 RVEC3 RMB 3 RVEC3 RMB 3 RVEC4 RMB 3 RVEC5 RMB 3 RVEC5 RMB 3 RVEC6 RMB 3 RVEC7 RMB 3 RVEC7 RMB 3 RVEC8 RMB 3 RVEC9 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C844 \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C58C CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3 RVEC3 RMB 3 RVEC4 RMB 3 RVEC5 RMB 3 RVEC5 RMB 3 RVEC6 RMB 3 RVEC7 RMB 3 RVEC7 RMB 3 RVEC7 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C889 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C5BF \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179	* RVECØ RMB 3 RVEC1 RMB 3 RVEC2 RMB 3 RVEC3 RMB 3 RVEC4 RMB 3 RVEC5 RMB 3 RVEC5 RMB 3 RVEC6 RMB 3 RVEC6 RMB 3 RVEC7 RMB 3 RVEC7 RMB 3 RVEC8 RMB 3 RVEC9 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A4426 \$CA3B \$CAE9 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC8 RMB 3  RVEC8 RMB 3  RVEC9 RMB 3  RVEC1Ø RMB 3  RVEC1Ø RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$CB4A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182	*  RVEC0 RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CA6P CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C667 \$C664 INPUTTING A BASIC LINE
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC9 RMB 3  RVEC1Ø RMB 3  RVEC1Ø RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C844 \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A446 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C859 \$C839 BREAK CHECK \$A390 \$C687 \$C6E4 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CAE4 TERMINATING BASIC LINE
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0423 017F 0424 0188	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A446 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C6B7 \$C6E4 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CAE4 TERMINATING BASIC LINE INPUT
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0423 017F 0424 0182 0425 0185 0426 0188	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A4406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A420 \$C43B \$CA59 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C687 \$C664 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CA44 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0428 018E	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C818 \$C848 PRINT DEVICE NUMBER CHECK \$A4406 \$C818 \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CA69 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C667 \$C664 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CA44 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C667 \$C664 RESERVED FOR ON ERROR GOTO COMMAND
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0188 0427 0188 0428 018E 0429 0191	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC11 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A4406 \$C818 \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CA69 CLOSE ALL FILES \$A442D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C667 \$C664 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CA64 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C687 \$C664 RESERVED FOR ON ERROR GOTO COMMAND
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0428 018E	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C818 \$C848 PRINT DEVICE NUMBER CHECK \$A4406 \$C818 \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CA69 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C667 \$C664 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CA44 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C667 \$C664 RESERVED FOR ON ERROR GOTO COMMAND
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0188 0427 0188 0428 018E 0429 0191	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC11 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A4406 \$C818 \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CA69 CLOSE ALL FILES \$A442D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C667 \$C664 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CA64 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C687 \$C664 RESERVED FOR ON ERROR GOTO COMMAND
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0427 0188 0428 018E 0429 0191 0430 0194 0431 0197	*  RVEC0 RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$CB4A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A42C \$CA3B \$CAE9 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C659 \$C8A9 BREAK CHECK \$A390 \$C667 \$C6E4 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CAE4 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C6B7 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A5C5 \$829C \$C990 \$CA3E RUN \$BD22 \$87EF
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0427 0188 0428 018E 0429 0191 0430 0194 0431 0197	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC19 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A4406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A420 \$C43B \$CA5P CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$C65B \$C035 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C6B7 \$C664 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CA64 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C667 \$C664 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A5D2 \$87EF \$AD9E \$82B9 \$C880 BASIC'S COMMAND INTERPRETATION LC
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0428 018E 0429 0191 0430 0194 0431 0197 0432 019A	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC1Ø RMB 3  RVEC1Ø RMB 3  RVEC1Ø RMB 3  RVEC1Ø RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$CB4A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A4406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A420 \$C61B \$C61B \$C61B PRINT DEVICE NUMBER CHECK \$A420 \$C61B PRINT DEVICE NUMBER CHECK \$A420 \$C61B PRINT PRINT \$B061 \$C65B PRINT \$B061 PCC5B PRINT \$B061 PCC5B PRINT \$B061 PCC5B PRINT \$B061 PCC5B
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0428 018E 0429 0191 0430 0194 0431 0197 0432 019A 0433 019D 0434 01A0	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC21 RMB 3  RVEC22 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$CB4A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C818 \$C848 PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CA6P CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$B061 \$CC5B \$CB35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A4390 \$C667 \$C664 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CA64 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C6B7 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$8BF0 \$C24D \$C265 ERROR DRIVER \$ABC49 \$8BF0 \$C24D \$C265 ERROR DRIVER \$ABC55 \$829C \$C990 \$CA3E RUN \$BD22 \$87EF ASCII TO FLOATING POINT CONVERSION \$ASCII TO FLOATING POINT CONVERSION \$ABC40 \$ABC40 \$CB0 \$ABC40 \$CB0 \$CB0 \$CB0 \$CB0 \$CB0 \$CB0 \$CB0 \$CB
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0427 0188 0429 0191 0430 0194 0431 0197 0432 019A 0433 019D 0434 01A0	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC66 RMB 3  RVEC67 RMB 3  RVEC7 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC21 RMB 3  RVEC21 RMB 3  RVEC21 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C818 \$C848 PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CA69 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$B061 \$CC5B \$CB35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C667 \$C664 INPUTTING A BASIC LINE \$A549 \$CA60 \$C607 \$C664 INPUTTING A BASIC LINE \$A54BF \$CA36 \$CA64 TERMINATING BASIC LINE INPUT \$A55CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C6B7 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A5C5 \$829C \$C990 \$CA3E RUN \$BD22 \$87EF AD9E \$8289 \$C880 BASIC'S COMMAND INTERPRETATION LOTED ACCUMANT COMMANDS \$ASC4 \$C880 BASIC'S COMMAND INTERPRETATION LOTED ACCUMANT COMMANDS \$CLS \$C8B0 \$CS85' SECONDARY TOKEN HANDLER
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 018B 0428 018E 0429 0191 0430 0194 0431 0197 0432 0197 0433 0190 0433 019D 0434 01A0	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC22 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A4406 \$C818 \$CA49 CLOSE ALL FILES \$A442D \$8286 \$CA48 \$CA9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 \$RINT \$B061 \$CC5B \$CD35 INPUT \$B061 \$CC5B \$CB35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C6B7 \$C6E4 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CAE4 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$A223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C6B7 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A5C9 \$CA60 \$C690 \$CA3E RUN \$BD22 \$87EF AD9E \$8290 \$CA3E RUN \$BD22 \$87EF AD9E \$82B9 \$CA3E RUN \$B162 \$EXBAS' SECONDARY TOKEN HANDLER \$A864 \$CA910 \$CSBAS' SECONDARY TOKEN HANDLER \$CASA \$CASA' RENUM TOKEN CHECK
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 018B 0428 018E 0429 0191 0430 0194 0431 0197 0432 019A 0433 019D 0434 01A0 0435	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC12 RMB 3  RVEC12 RMB 3  RVEC13 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC21 RMB 3  RVEC22 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$CB4A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CA89 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CA5P CLOSE ONE FILE \$B918 \$8E90 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$B061 \$C659 \$C849 BREAK CHECK \$A390 \$C667 \$C664 INPUTTING A BASIC LINE \$A4BF \$CA60 \$CA60 \$CA60 \$CO90 EOF COMMAND \$A5CE \$CA60 \$CA60 \$CO90 EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C667 \$C664 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A5C9 \$C690 \$CA3E RUN \$BD22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BD22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BB22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BB22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BB22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BB32 \$CA3E RUN \$BCA3E RUN \$BCA3E RUN \$BCA3E RUN \$CA5E RUN \$
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 018B 0428 018E 0429 0191 0430 0194 0431 0197 0432 0197 0433 0190 0433 019D 0434 01A0	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC22 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A4406 \$C818 \$CA49 CLOSE ALL FILES \$A442D \$8286 \$CA48 \$CA9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 \$RINT \$B061 \$CC5B \$CD35 INPUT \$B061 \$CC5B \$CB35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C6B7 \$C6E4 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CAE4 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$A223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C6B7 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A5C9 \$CA60 \$C690 \$CA3E RUN \$BD22 \$87EF AD9E \$8290 \$CA3E RUN \$BD22 \$87EF AD9E \$82B9 \$CA3E RUN \$B162 \$EXBAS' SECONDARY TOKEN HANDLER \$A864 \$CA910 \$CSBAS' SECONDARY TOKEN HANDLER \$CASA \$CASA' RENUM TOKEN CHECK
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 018B 0428 018E 0429 0191 0430 0194 0431 0197 0432 019A 0433 019D 0434 01A0 0435	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC9 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC12 RMB 3  RVEC12 RMB 3  RVEC13 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC21 RMB 3  RVEC22 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$CB4A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CA89 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CA5P CLOSE ONE FILE \$B918 \$8E90 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$B061 \$C659 \$C849 BREAK CHECK \$A390 \$C667 \$C664 INPUTTING A BASIC LINE \$A4BF \$CA60 \$CA60 \$CA60 \$CO90 EOF COMMAND \$A5CE \$CA60 \$CA60 \$CO90 EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C667 \$C664 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A5C9 \$C690 \$CA3E RUN \$BD22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BD22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BB22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BB22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BB22 \$87EF \$AD9E \$8289 \$CA3E RUN \$BB32 \$CA3E RUN \$BCA3E RUN \$BCA3E RUN \$BCA3E RUN \$CA5E RUN \$
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 016A 0417 016D 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0427 0188 0428 018E 0429 0191 0430 0194 0431 0197 0432 019A 0433 019D 0434 01A0 0435 0436 0437	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC22 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$B918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C859 \$C8A9 BREAK CHECK \$A390 \$C6B7 \$C6E4 INPUTTING A BASIC LINE \$A4BF \$CA36 \$CAE4 TERMINATING BASIC LINE INPUT \$A5CE \$CA60 \$C90C EOF COMMAND \$B223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C6B7 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A5C9 \$C990 \$CA3E RUN \$BD22 \$87EF \$AD9E \$82B9 \$CA3E RUN \$BC8D \$AC46 \$C6CD \$CACA \$CA
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 0417 0160 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0429 0191 0430 0194 0431 0197 0432 019A 0433 019D 0434 01A0 0435 0436 0437 0438 01A3 0439 01A6	*  RVEC0 RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC22 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C684A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C848 INPUT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES \$A420 \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$8918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C687 \$C664 INPUTTING A BASIC LINE \$A48F \$CA36 \$CAE4 TERMINATING BASIC LINE INPUT \$A549 \$C667 \$C6E4 INPUTTING A BASIC LINE INPUT \$A549 \$C667 \$C6E4 INPUTTING A BASIC LINE INPUT \$A55CE \$CA60 \$C90C EOF COMMAND \$A223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C667 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN AC46 \$C687 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN AC46 \$C687 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN BASIC \$8025 \$876F \$AD9E \$8290 \$CA3E RUN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$ASC1 TO FLOATING POINT CONVERSION \$CLS \$B821 \$8304 \$CC29A \$C282 EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 0116 0418 0177 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0429 0191 0430 0194 0431 0197 0432 0190 0433 0190 0434 01A0 0435 0436 0437 0438 01A3 0439 01A6 0440 0441 01A9	*  RVECØ RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC66 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC22 RMB 3  RVEC22 RMB 3  RVEC22 RMB 3  STRSTK RMB 8*5	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C84A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A446 \$C81B \$C84B PRINT DEVICE NUMBER CHECK \$A426 \$CA38 \$CAE9 CLOSE ALL FILES \$A42D \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$8918 \$8E90 \$8E90 PRINT \$8061 \$CC58 \$C035 INPUT \$8061 \$CC58 \$C035 INPUT \$A549 \$C859 \$C849 BREAK CHECK \$A390 \$C667 \$C6E4 INPUTTING A BASIC LINE \$A48F \$CA36 \$CAE4 TERMINATING BASIC LINE INPUT \$A56CE \$CA60 \$C90C EOF COMMAND \$A223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C667 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A529 \$C890 \$CA3E RUN \$A501 TO FLOATING POINT CONVERSION \$AC46 \$C667 \$C664 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A510 CLS \$A3910 \$C687 \$C684 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A511 TO FLOATING POINT CONVERSION \$A612 \$C867 \$C664 RESERVED FOR ON ERROR GOTO COMMAN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$A511 TO FLOATING POINT CONVERSION \$A612 \$C867 \$C860 BASIC'S COMMAND INTERPRETATION LOCATION CLS \$A614 \$C667 \$C667 \$C667 \$C667 \$C667 \$C667 \$C677 \$C77 \$C
0409 0410 0411 0412 015E 0413 0161 0414 0164 0415 0167 0416 0417 0160 0418 0170 0419 0173 0420 0176 0421 0179 0422 017C 0423 017F 0424 0182 0425 0185 0426 0188 0427 0188 0429 0191 0430 0194 0431 0197 0432 019A 0433 019D 0434 01A0 0435 0436 0437 0438 01A3 0439 01A6	*  RVEC0 RMB 3  RVEC1 RMB 3  RVEC2 RMB 3  RVEC3 RMB 3  RVEC4 RMB 3  RVEC5 RMB 3  RVEC6 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC7 RMB 3  RVEC10 RMB 3  RVEC10 RMB 3  RVEC11 RMB 3  RVEC12 RMB 3  RVEC14 RMB 3  RVEC15 RMB 3  RVEC15 RMB 3  RVEC16 RMB 3  RVEC16 RMB 3  RVEC17 RMB 3  RVEC17 RMB 3  RVEC18 RMB 3  RVEC19 RMB 3  RVEC20 RMB 3  RVEC21 RMB 3  RVEC22 RMB 3	2.0 2.1 1.0 1.1 \$A5F6 \$C426 \$C44B OPEN COMMAND \$A5B9 \$C838 \$C888 DEVICE NUMBER VALIDITY CHECK \$A35F \$C843 \$C893 SET PRINT PARAMETERS \$A282 \$8273 \$C684A \$CC1C CONSOLE OUT \$A176 \$8CF1 \$C58F \$C5BC CONSOLE IN \$A3ED \$C818 \$C848 INPUT DEVICE NUMBER CHECK \$A406 \$C81B \$C848 INPUT DEVICE NUMBER CHECK \$A426 \$CA3B \$CAE9 CLOSE ALL FILES \$A420 \$8286 \$CA4B \$CAF9 CLOSE ONE FILE \$8918 \$8E90 \$8E90 PRINT \$B061 \$CC5B \$CD35 INPUT \$A549 \$C687 \$C664 INPUTTING A BASIC LINE \$A48F \$CA36 \$CAE4 TERMINATING BASIC LINE INPUT \$A549 \$C667 \$C6E4 INPUTTING A BASIC LINE INPUT \$A549 \$C667 \$C6E4 INPUTTING A BASIC LINE INPUT \$A55CE \$CA60 \$C90C EOF COMMAND \$A223 \$8846 \$CDF6 \$CED2 EVALUATE AN EXPRESSION \$AC46 \$C667 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN AC46 \$C687 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN AC46 \$C687 \$C6E4 RESERVED FOR ON ERROR GOTO COMMAN BASIC \$8025 \$876F \$AD9E \$8290 \$CA3E RUN \$AC49 \$88F0 \$C24D \$C265 ERROR DRIVER \$ASC1 TO FLOATING POINT CONVERSION \$CLS \$B821 \$8304 \$CC29A \$C282 EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN CHECK \$CLS \$EXBAS' SECONDARY TOKEN HANDLER EXBAS' RENUM TOKEN CHECK \$CLS \$EXBAS' SECONDARY

$\alpha \Lambda \Lambda \Lambda$	Ø2DA	LINHDR	RMB	2	LINE INPUT BUFFER HEADER
Ø445	02DC	LINBUF	RMB	LBUFMX+1	BASIC LINE INPUT BUFFER
Ø446	Ø3D7	STRBUF	RMB	41	STRING BUFFER
Ø447					
Ø448	0400	VIDRAM	RMB	200	VIDEO DISPLAY AREA
Ø449					
					E GEORGE (DION DAGES ONLY)
Ø45Ø		*START OF	ADD11	IONAL KAM VARIABI	LE STORAGE (DISK BASIC ONLY)
Ø451	0600	DBUFØ	RMB	SECLEN	I/O BUFFER #Ø
Ø452		DBUF1	RMB	SECLEN	I/O BUFFER #1
Ø453	0800	FATBLØ	RMB	FATLEN	FILE ALLOCATION TABLE - DRIVE Ø
Ø454	<b>Θ</b> 84Δ	FATBL1	RMB	FATLEN	FILE ALLOCATION TABLE - DRIVE 1
Ø455		FATBL2	RMB	FATLEN	FILE ALLOCATION TABLE - DRIVE 2
Ø456	Ø8DE	FATBL3	RMB	FATLEN	FILE ALLOCATION TABLE - DRIVE 3
Ø457	Ø928	FCBV1	RMB	16*2	FILE BUFFER VECTORS (15 USER, 1 SYSTEM)
Ø458	0948	RNBFAD	RMB	2	START OF FREE RANDOM FILE BUFFER AREA
Ø459	Ø94A	FCBADR	RMB	2	START OF FILE CONTROL BLOCKS
Ø46Ø		DNAMBF		8	DISK FILE NAME BUFFER
0461	Ø954	DEXTBF	RMB	3	DISK FILE EXTENSION NAME BUFFER
Ø462	Ø957	DFLTYP	RMB	1	*DV* DISK FILE TYPE: Ø=BASIC, 1=DATA, 2=MACHINE
Ø463		*			LANGUAGE, 3=TEXT EDITOR SOURCE FILE
				_	·
Ø464	0958	DASCFL	RMB	1	*DV* ASCII FLAG: Ø=CRUNCHED OR BINARY, \$FF=ASCII
Ø465	Ø959	DRUNFL	RMB	1	RUN FLAG: (IF BIT 1=1 THEN RUN, IF BIT Ø=1, THEN CLOSE
Ø466		*			ALL FILES BEFORE RUNNING)
	ao e v		DMD	1	
Ø467		DEFDRV	RMB	1	DEFAULT DRIVE NUMBER
Ø468	Ø95B	FCBACT	RMB	1	NUMBER OF FCBS ACTIVE
Ø469		DRESFL		1	RESET FLAG: ◇Ø WILL CAUSE A 'NEW' & SHUT DOWN ALL FCBS
Ø47Ø		DLOADFL		1	LOAD FLAG: CAUSE A 'NEW' FOLLOWING A LOAD ERROR
Ø471	Ø95E	DMRGFL	RMB	1	MERGE FLAG: Ø=NØ MERGE, \$FF=MERGE
Ø472	Ø95F	DUSRVC	RMB	20	DISK BASIC USR COMMAND VECTORS
	2331				
Ø473				ORK AREA FOR DIR	ECTURY SEARCH
Ø474		* EXISTI	NG FI	LE	
Ø475	0973	V973	RMB	1	SECTOR NUMBER
Ø476		V974			
				2	RAM DIRECTORY IMAGE ADDRESS
Ø477	Ø976	V976	RMB	1	FIRST GRANULE NUMBER
Ø478		* UNUSED	FILE		
Ø479	Ø977	V977	RMB	1	SECTOR NUMBER
Ø48Ø	W978	V978	RMB	2	RAM DIRECTORY IMAGE ADDRESS
Ø481					
Ø482	Ø97A	WFATVL	RMB	2	WRITE FAT VALUE: NUMBER OF FREE GRANULES WHICH MUST BE TAKEN
	<b>5</b> 5771		IUID	_	
Ø483					FROM THE FAT TO TRIGGER A WRITE FAT TO DISK SEQUENCE
Ø484	Ø97C	DFFLEN	RMB	2	DIRECT ACCESS FILE RECORD LENGTH
0/185					CURRENT TRACK NUMBER, DRIVES Ø,1,2,3
	И97F	DRØTRK	RMB	4	
	Ø97E	DRØTRK	RMB	4	
Ø486	Ø982	NMIFLG	RMB	1	NMI FLAG: Ø=DON'T VECTOR ≪Ø=YECTOR OUT
	Ø982				
Ø486 Ø487	Ø982	NMIFLG	RMB	1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI
Ø486 Ø487 Ø488	Ø982 Ø983	NMIFLG DNMIVC *	RMB RMB	1 2	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET
Ø486 Ø487 Ø488 Ø489	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li></ul>	NMIFLG DNMIVC * RDYTMR	RMB RMB	1 2 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER
Ø486 Ø487 Ø488 Ø489 Ø49Ø	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM	RMB RMB RMB RMB	1 2 1 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø)
Ø486 Ø487 Ø488 Ø489	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li></ul>	NMIFLG DNMIVC * RDYTMR	RMB RMB RMB RMB	1 2 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø)
Ø486 Ø487 Ø488 Ø489 Ø49Ø Ø491	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL	RMB RMB RMB RMB RMB	1 2 1 1 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON
Ø486 Ø487 Ø488 Ø489 Ø49Ø Ø491 Ø492	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR	RMB RMB RMB RMB	1 2 1 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE
0486 0487 0488 0489 0490 0491 0492 0493	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR *	RMB RMB RMB RMB RMB	1 2 1 1 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA
Ø486 Ø487 Ø488 Ø489 Ø49Ø Ø491 Ø492	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR	RMB RMB RMB RMB RMB	1 2 1 1 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE
0486 0487 0488 0489 0490 0491 0492 0493 0494	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR *	RMB RMB RMB RMB RMB	1 2 1 1 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA
0486 0487 0488 0489 0490 0491 0492 0493 0494	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR *	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.
0486 0487 0488 0489 0490 0491 0492 0493 0494 0495	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR *	RMB RMB RMB RMB RMB	1 2 1 1 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA
0486 0487 0488 0489 0490 0491 0492 0493 0494 0495 0496 0497	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * *	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.
0486 0487 0488 0489 0490 0491 0492 0493 0494 0495	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * *	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.
0486 0487 0488 0489 0490 0491 0492 0493 0493 0495 0496 0497	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * *	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.
0486 0487 0488 0489 0490 0491 0492 0493 0494 0495 0496 0497 0498	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF	RMB RMB RMB RMB RMB RMB	1 1 1 1 1 1 1 SECLEN	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0491 0492 0493 0494 0495 0496 0497 0498 0499	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0490 0491 0492 0493 0494 0495 0496 0497 0498	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF	RMB RMB RMB RMB RMB RMB	1 1 1 1 1 1 1 SECLEN	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0491 0492 0493 0494 0495 0496 0497 0499 0500 0501	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFERS	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0491 0492 0493 0494 0495 0496 0497 0498 0499 0501 0501	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI	RMB RMB RMB RMB RMB RMB	1 1 1 1 1 1 1 SECLEN	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0499 0491 0492 0493 0494 0495 0496 0497 0498 0499 0500 0501 0502	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFERS	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0491 0492 0493 0494 0495 0496 0497 0498 0499 0501 0501	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFERS	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0491 0492 0493 0494 0495 0497 0498 0499 0500 0501 0502	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFERS	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0495 0496 0497 0498 0499 0501 0501 0503 0504 0505	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFERS	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0495 0496 0497 0500 0501 0502 0503 0504 0505 0506	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFERS	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0491 0492 0493 0494 0495 0496 0497 0500 0501 0502 0503 0505 0506 0507	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0495 0496 0497 0500 0501 0502 0503 0504 0505 0506	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0491 0492 0493 0494 0495 0496 0497 0500 0501 0502 0503 0504 0506 0507 0508	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0496 0497 0498 0500 0501 0502 0503 0504 0505 0506 0507 0508	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO	RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0495 0497 0498 0501 0502 0503 0504 0505 0506 0507 0508 0508 0508 0508	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO *VARIABLE *ARRAY STO	RMB RMB RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0496 0497 0498 0500 0501 0502 0503 0504 0505 0506 0507 0508	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO	RMB RMB RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0495 0496 0501 0501 0503 0504 0505 0506 0507 0508 0509 0510 0501	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO *VARIABLE *ARRAY STO	RMB RMB RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0496 0496 0501 0502 0503 0504 0505 0506 0507 0508 0508 0510 0511 0511	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO *VARIABLE *ARRAY STO	RMB RMB RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0495 0496 0498 0500 0501 0502 0503 0506 0507 0508 0509 0511 0512	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO *VARIABLE *ARRAY STO * FREE MEM	RMB RMB RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0496 0496 0501 0502 0503 0504 0505 0506 0507 0508 0508 0510 0511 0511	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO *VARIABLE *ARRAY STO	RMB RMB RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0489 0491 0492 0493 0494 0495 0496 0497 0500 0501 0502 0503 0504 0505 0506 0507 0508 0509 0511 05512	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO *VARIABLE *ARRAY STO * FREE MEM	RMB RMB RMB RMB RMB RMB RMB RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0495 0497 0498 0500 0501 0502 0503 0504 0505 0506 0507 0508 0508 0509 0510 0511 0512 0514	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO *VARIABLE *ARRAY STO * * FREE MEM * *STACK	RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0495 0497 0498 0501 0501 0503 0504 0505 0506 0507 0508 0508 0508 0509 0510 0511 0512 0513 0514	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO *VARIABLE *ARRAY STO * FREE MEM	RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS
0486 0487 0488 0499 0491 0492 0493 0494 0495 0497 0498 0500 0501 0502 0503 0504 0505 0506 0507 0508 0508 0509 0510 0511 0512 0514	<ul><li>Ø982</li><li>Ø983</li><li>Ø985</li><li>Ø986</li><li>Ø987</li><li>Ø988</li></ul>	NMIFLG DNMIVC * RDYTMR DRGRAM DVERFL ATTCTR * * DFLBUF *RANDOM FI *FILE CONT *GRAPHIC F *BASIC PRO *VARIABLE *ARRAY STO * * FREE MEM * *STACK	RMB	1 2 1 1 1 1 1 SECLEN SERVED AREA LOCKS AND BUFFER: ESERVED AREA	NMI FLAG: Ø=DON'T VECTOR <>Ø=YECTOR OUT NMI VECTOR: WHERE TO JUMP FOLLOWING AN NMI INTERRUPT IF THE NMI FLAG IS SET MOTOR TURN OFF TIMER RAM IMAGE OF DSKREG (\$FF4Ø) VERIFY FLAG: Ø=OFF, \$FF=ON READ/WRITE ATTEMPT COUNTER: NUMBER OF TIMES THE DISK WILL ATTEMPT TO RETRIEVE OR WRITE DATA BEFORE IT GIVES UP AND ISSUES AN ERROR.  INITIALIZED TO SECLEN BY DISKBAS

```
*USER PROGRAM RESERVED AREA
Ø519
0520
                     *END OF RAM
Ø521
0522 8000
                     ORG
                                $8000
0523
0524 8000
                                      $2000
                                                       EXTENDED BASIC ROM
                                RMR
Ø525 AØØØ
                                RMB
                                      $2000
                                                       COLOR BASIC ROM
Ø526 CØØØ
                     ROMPAK
                                EQU
                     DOSBAS
                                      $2000
                                                       DISK BASIC ROM/ENHANCED BASIC INIT CODE
Ø527 CØØØ
                                RMR
Ø528 EØØØ
                                RMR
                                      $1FØØ
                                                       ENHANCED BASIC
Ø529
                     * START OF ADDITIONAL VARIABLES USED BY SUPER EXTENDED BASIC
Ø53Ø
                     H.CRSLOC
                                                       CURRENT LOCATION OF CURSOR
Ø531 FEØØ
                                RMB
                                     2
Ø532 FEØ2
                     H.CURSX
                                RMR
                                                       X POSITION OF CURSOR
                                                       Y POSITION OF CURSOR
Ø533 FEØ3
                     H.CURSY
                                RMB
                                      1
                                                       COLUMNS ON HI-RES ALPHA SCREEN
                     H.COLUMN
                                RMR
Ø534 FEØ4
                                      1
                                                       ROWS ON HI-RES ALPHA SCREEN
Ø535 FEØ5
                     H.ROW
                                RMR
                                      1
Ø536 FEØ6
                     H.DISPEN
                                RMB
                                      2
                                                       END OF HI-RES DISPLAY SCREEN
Ø537 FEØ8
                     H.CRSATT
                                RMB
                                                       CURRENT CURSOR'S ATTRIBUTES
                                      1
                                RMR
                                                       UNUSED
Ø538 FEØ9
                                      1
Ø539 FEØA
                     H.FCOLOR
                                RMB
                                      1
                                                       FOREGROUND COLOR
                     H.BCOLOR
                                                       BACKGROUND COLOR
Ø54Ø FEØB
                                RMB
                                      1
                     H.ONBRK
                                RMB
                                                       ON BRK GOTO LINE NUMBER
Ø541 FEØC
                                      2
0542 FF0F
                     H.ONFRR
                                RMB
                                      2
                                                       ON ERR GOTO LINE NUMBER
Ø543 FE1Ø
                     H.ERROR
                                RMB
                                      1
                                                       ERROR NUMBER ENCOUNTERED OR $FF (NO ERROR)
Ø544 FE11
                     H.ONERRS
                                RMB
                                      2
                                                       ON ERR SOURCE LINE NUMBER
                                RMB
                                                       LINE NUMBER WHERE ERROR OCCURRED
Ø545 FE13
                     H.ERLINE
                                      2
                     H.ONBRKS
                                                       ON BRK SOURCE LINE NUMBER
Ø546 FE15
                                RMR
                                      2
Ø547 FE17
                     H.ERRBRK
                                RMB
                                      1
                                                       STILL UNKNOWN, HAS TO DO WITH ERR, BRK
                                                       PRINT COUNT, CHARACTERS TO BE HPRINTED
Ø548 FE18
                     H.PCOUNT
                                RMB
                                      1
                                                       PRINT BUFFER, HPRINT CHARS. STORED HERE
Ø549 FE19
                     H. PBUF
                                RMB
                                      80
Ø55Ø FE69
                                RMB
                                      132
                                                       UNUSED
                                                       INTERRUPT VALID FLAG. Ø=NOT VALID, $55=VALID
Ø551 FEED
                     INT.FLAG
                                RMR
                     * TABLE OF JUMP VECTORS TO INTERRUPT SERVICING ROUTINES
Ø552
Ø553 FEEE
                     INT.JUMP
Ø554 FEEE
                     INT.SWI3
                                RMB
                                      3
Ø555 FEF1
                     INT.SWI2
                                RMR
                                      3
                     INT.FIRQ
Ø556 FEF4
                                RMR
                                      3
Ø557 FEF7
                     INT.IRO
                                RMR
                                      3
Ø558 FEFA
                     INT.SWI
                                RMB
                                      3
Ø559 FEFD
                     INT.NMI
                                RMB
                                      3
0560
Ø561
                     * I/O AREA
Ø562
Ø563 FFØØ
                     PIAØ
                                EQU
                                                       PERIPHERAL INTERFACE ADAPTER ONE
Ø564
Ø565 FFØØ
                     втти
                                KEYBOARD ROW 1 AND RIGHT JOYSTICK SWITCH 1
                                KEYBOARD ROW 2 AND LEFT JOYSTICK SWITCH 1
Ø566
                     BIT1
                                KEYBOARD ROW 3 AND RIGHT JOYSTICK SWITCH 2
                     BIT2
0567
Ø568
                     BIT3
                                KEYBOARD ROW 4 AND LEFT JOYSTICK SWITCH 2
Ø569
                     BIT4
                                KEYBOARD ROW 5
Ø57Ø
                     BIT5
                                KEYBOARD ROW 6
                                KEYBOARD ROW 7
Ø571
                     BIT6
Ø572
                     BIT7
                                JOTSTICK COMPARISON IINPUT
Ø573
Ø574 FFØ1
                                CONTROL OF HSYNC (63.5ps)
                                                              \emptyset = IRQ* TO CPU DISABLED
                     BITØ
                                                              1 = IRO* TO CPU ENABLED
                                INTERRUPT
0575
Ø576
                     BIT1
                                CONTROL OF INTERRUPT
                                                              \emptyset = FLAG SET ON FALLING EDGE OF HS
Ø577
                                POLARITY
                                                              1 = FLAG SET ON RISING EDGE OF HS
Ø578
                     BIT2
                                NORMALLY 1
                                                              \emptyset = CHANGES FEØØ TO DATA DIRECTION
Ø579
                     BIT3
                                SEL 1
                                                              LSB OF TWO ANALOG MUX SELECT LINES
Ø58Ø
                     BIT4
                                ALWAYS 1
                     BIT5
                                ALWAYS 1
Ø581
0582
                                NOT USED
                     BIT6
Ø583
                     BIT7
                                HORIZONTAL SYNC INTERRUPT FLAG
Ø584
Ø585 FFØ2
                     BITØ
                                KEYBOARD COLUMN 1
                                KEYBOARD COLUMN 2
Ø586
                     BIT1
Ø587
                     BIT2
                                KEYBOARD COLUMN 3
                                KEYBOARD COLUMN 4
Ø588
                     BIT3
                                KEYBOARD COLUMN 5
Ø589
                     BIT4
Ø59Ø
                     BIT5
                                KEYBOARD COLUMN 6
Ø591
                     BIT6
                                KEYBOARD COLUMN 7 / RAM SIZE OUTPUT
```

DISK BASIC UNRAVELLED II

Ø592		BIT7	KEYBOARD COLUMN 8			
Ø593		DI17	KEIBOARD COLUMN O			
Ø594 Ø595	FFØ3	BITØ	CONTROL OF VSYNC (16. INTERRUPT	667ms)	Ø = IRQ* TO CPU DISA 1 = IRO* TO CPU ENAB	
Ø596		BIT1	CONTROL OF INTERRUPT		Ø = FLAG SET ON FALL	ING EDGE OF FS
Ø597 Ø598		BIT2	POLARITY NORMALLY 1		1 = FLAG SET ON RISI Ø = CHANGES FFØ2 TO	
Ø599		BIT3	SEL 2		MSB OF TWO ANALOG MU	
0600		BIT4	ALWAYS 1			
0601		BIT5	ALWAYS 1			
0602		BIT6	NOT USED			
0603		BIT7	FIELD SYNC INTERRUPT	FLAG		
Ø6Ø4	ΓΓα4		DMD 20	DTAG T	MACEC	
Ø6Ø6	FFØ4 FF2Ø	DA	RMB 28	PIAØ I	MAGES	
Ø6Ø7		PIA1	EQU *	PERIPH	IERAL INTERFACE ADAPTE	R TWO
Ø6Ø8						
0609	FF2Ø	BITØ	CASSETTE DATA INPUT			
0610		BIT1	RS-232C DATA OUTPUT			
Ø611		BIT2	6 BIT D/A LSB			
Ø612 Ø613		BIT3 BIT4	6 BIT D/A 6 BIT D/A			
Ø614		BIT5	6 BIT D/A			
Ø615		BIT6	6 BIT D/A			
Ø616		BIT7	6 BIT D/A MSB			
Ø617						
Ø618	FF21	BITØ	CONTROL OF CD		Ø = FIRQ* TO CPU DIS	
Ø619 Ø62Ø		BIT1	(RS-232C STATUS) CONTROL OF INTERRUPT		1 = FIRQ* TO CPU ENA Ø = FLAG SET ON FALL	
Ø621		DIII	POLARITY		1 = FLAG SET ON FALL	
Ø622		BIT2	NORMALLY 1		$\emptyset = CHANGES FF2\emptyset TO$	
Ø623		BIT3	CASSETTE MOTOR CONTRO	L	$\emptyset = OFF$ $1 = ON$	
Ø624		BIT4	ALWAYS 1			
Ø625		BIT5	ALWAYS 1			
Ø626		BIT6	NOT USED			
Ø627 Ø628		BIT7	CD INTERRUPT FLAG			
Ø629	FF22	BITØ	RS-232C DATA INPUT			
Ø63Ø		BIT1	SINGLE BIT SOUND OUTP	UT		
Ø631		BIT2	RAM SIZE INPUT			
Ø632		BIT3	RGB MONITOR SENSING I	NPUT	CSS	0.5.1
Ø633 Ø634		BIT4 BIT5	VDG CONTROL OUTPUT VDG CONTROL OUTPUT		GMØ & UPPER/LOWER CA GM1 & INVERT	2F*
Ø635		BIT6	VDG CONTROL OUTPUT		GM2	
Ø636		BIT7	VDG CONTROL OUTPUT		A*/G	
Ø637						
	FF23	BITØ	CONTROL OF CARTRIDGE		Ø = FIRQ* TO CPU DIS	
Ø639		DIT1	INTERRUPT		1 = FIRQ* TO CPU ENA Ø = FLAG SET ON FALL	
Ø64Ø Ø641		BIT1	CONTROL OF INTERRUPT POLARITY		1 = FLAG SET ON RISI	
Ø642		BIT2	NORMALLY 1		$\emptyset$ = CHANGES FF22 TO	
Ø643		BIT3	SOUND ENABLE			
Ø644		BIT4	ALWAYS 1			
Ø645		BIT5	ALWAYS 1			
Ø646 Ø647		BIT6 BIT7	NOT USED CARTRIDGE INTERRUPT F	LAC		
Ø648		211/	SARINIDUL INILKKUFI F	LAU		
	FF24		RMB 28	PIA1 I	MAGES	
Ø65Ø	FF4Ø	PIA2				
	FF4Ø	DSKREG	RMB 1	DISK C	CONTROL REGISTER	
Ø652	EE 1 0	рιτα	DRIVE SELECT Ø			
Ø653 Ø654	FF4Ø	BITØ BIT1	DRIVE SELECT D			
Ø655		BIT2	DRIVE SELECT 2			
Ø656		BIT3	DRIVE MOTOR ENABLE		Ø = MOTORS OFF	1 = MOTORS ON
Ø657		BIT4	WRITE PRECOMPENSATION		$\emptyset = NO PRECOMP$	1 = PRECOMP
Ø658		BIT5	DENSITY FLAG		$\emptyset = SINGLE$	1 = DOUBLE
Ø659		BIT6	DRIVE SELECT 3		a - DICABLED	1 - ENADIED
Ø66Ø Ø661		BIT7	HALT FLAG		Ø = DISABLED	1 = ENABLED
	FF41		RMB 7	DSKREG	I IMAGES	
Ø663						
Ø664	5540		ISK CONTROLLER INTERNA			
Ø665	FF48	FDCREG	RMB 1	STATUS	COMMAND REGISTER	

Ø666						
Ø667		COMMANDS	TVDE	COMMAND	CODE	
Ø668		COMMANDS	I	RESTORE	\$Ø3	
Ø669			I	SEEK	\$17	
Ø67Ø			Ī	STEP	\$23	
Ø671			I	STEP IN	\$43	
Ø672			Ī	STEP OUT	\$53	
Ø673			ΙΙ	READ SECTOR	\$80	
Ø674			II	WRITE SECTOR	\$AØ	
Ø675			III	READ ADDRESS	\$CØ	
Ø676			III	READ TRACK	\$E4	
Ø677			III	WRITE TRACK	\$F4	
Ø678			IV	FORCE INTERRUPT	\$DØ	
Ø679			••	TOROL INTERROTT	455	
Ø68Ø		STATUS	BIT	TYPE I	READ ADDRESS/SECTOR/TRACK	WRITE SECTOR/TRACK
Ø681		0.700	SØ	BUSY	BUSY	BUSY
Ø682			S1	INDEX	DRQ	DRQ
Ø683			S2	TRACK Ø	LOST DATA	LOST DATA
Ø684			\$3	CRC ERROR	CRC ERROR (EXCEPT TRACK)	CRC ERROR (EXCEPT TRACK)
Ø685			S4	SEEK ERROR	RNF (EXCEPT TRACK)	RNF (EXCEPT TRACK)
Ø686			S5	HEAD LOADED	RECORD TYPE (SECTOR ONLY)	WRITE FAULT
Ø687			S6	WRITE PROTECT		WRITE PROTECT
Ø688			S7	NOT READY	NOT READY	NOT READY
Ø689						
Ø69Ø	FF49		RMB	1	TRACK REGISTER	
Ø691	FF4A		RMB	1	SECTOR REGISTER	
Ø692			RMB	1	DATA REGISTER	
Ø693	FF4C		RMB	4	FDCREG IMAGES	
Ø694						
Ø695	FF5Ø		RMB	16	UNUSED SPACE	
Ø696	FF6Ø		RMB	1	X COORDINATE FOR X-PAD	
Ø697	FF61		RMB	1	Y COORDINATE FOR X-PAD	
Ø698	FF62		RMB	1	STATUS REGISTER FOR X-PAD	
Ø699	FF63		RMB	5	UNUSED	
0700		* RS-232 P		1 PAK		
0701			RMB	1	READ/WRITE DATA REGISTER	
0702			RMB	1	STATUS REGISTER	
0703			RMB	1	COMMAND REGISTER	
Ø7Ø4			RMB	1	CONTROL REGISTER	
Ø7Ø5			RMB	4		
Ø7Ø6			RMB	13	COUND (CDEFOU CARTRIDGE DECE	-
Ø7Ø7			RMB	1	SOUND/SPEECH CARTRIDGE RESE	
Ø7Ø8			RMB	1	SOUND/SPEECH CARTRIDGE READ	
Ø7Ø9	FF/F		RMB	1	MULTI-PAK PROGRAMMING REGIS	IER
Ø71Ø Ø711	ΕΕΟΜ		RMB	64	RESERVED FOR FUTURE EXPANSI	ON
Ø711	LLOM		KIID	04	RESERVED FOR FUTURE EXPANSI	ON
Ø712		* VIDEO CO	NTDOL	DECISTEDS		
Ø714	FF9Ø	INITØ	RMB	1	INITIALIZATION REGISTER Ø	
Ø715	1136	111110	KIID	-	THITTALIZATION REGISTER D	
Ø716	FF9Ø	BITØ	MCØ		ROM MAP CONTROL (SEE	TABLE BELOW)
Ø717		BIT1	MC1		ROM MAP CONTROL (SEE	
Ø718		BIT2	MC2		STANDARD SCS	,
Ø719		BIT3	MC3		1 = DRAM AT ØxFEXX I	S CONSTANT
Ø72Ø		BIT4	FEN		1 = CHIP FIRQ OUTPUT	
Ø721		BIT5	IEN		1 = CHIP IRQ OUTPUT	
Ø722		BIT6	M/P		1 = MMU ENABLED	
Ø723		BIT7	COCO		1 = COCO 1 & 2 COMPA	TIBLE
Ø724						
Ø725			MC1	MCØ	ROM MAPPING	
Ø726			Ø	Х	16K INTERNAL, 16K EXTERNAL	
Ø727			1	Ø	32K INTERNAL	
Ø728			1	1	32L EXTERNAL (EXCEPT FOR VE	CTORS)
Ø729						
	FF91	INIT1	RMB	1	INITIALIZATION REGISTER 1	
Ø731						
Ø732	FF91	BITØ	TR		MMU TASK REGISTER SE	LECT
Ø733		BIT1				
Ø734		BIT2				
Ø735		BIT3				
Ø736		BIT4	TINC		TIMED THRUT OF FOT	1-70 0-60
Ø737		BIT5	TINS		TIMER INPUT SELECT:	ı=/ωns, ω=63ns
Ø738		BIT6				
Ø739		BIT7				

DISK BASIC UNRAVELLED II

```
Ø814
                                      64Øx2
Ø815
                                      512x4
                                                                       Ø
                                                                              Ø
                                                        1
                                                               1
                                                                                     1
Ø816
                                      512x2
                                                               Ø
                                                        1
                                                                       Ø
                                                                              Ø
                                                                                     Ø
                                      32Øx16
Ø817
                                                        1
                                                               1
                                                                       1
                                                                              1
                                                                                     Ø
Ø818
                                      32Øx4
                                                        1
                                                               Ø
                                                                       1
                                                                              Ø
                                                                                     1
Ø819
                                      256x16
                                                                                     Ø
                                                                       Ø
                                                        1
                                                               1
                                                                              1
                                      256x4
0820
                                                        1
                                                               Ø
                                                                       Ø
                                                                              Ø
                                                                                     1
Ø821
                                      256x2
                                                        Ø
                                                               1
                                                                       Ø
                                                                              Ø
                                                                                     Ø
Ø822
                                                                                     Ø
                                      160x16
                                                        1
                                                                       1
                                                                              1
0823
Ø824
                     * COCO MODE SELECTION
Ø825
                                                        DISPLAY MODE
                                                                              REG. FF22
Ø826
                                                        ٧2
                                                               ۷1
                                                                       ٧Ø
                                                                                                       3
                                                                                             5
                                                                                                  4
                                                                              7
                                                                                     6
                     ALPHA
                                                                                                        CSS
Ø827
                                                        Ø
                                                               Ø
                                                                       Ø
                                                                              Ø
                                                                                     Х
                                                                                             Х
                                                                                                  Ø
Ø828
                     ALPHA INVERTED
                                                        Ø
                                                               Ø
                                                                       Ø
                                                                              Ø
                                                                                     Х
                                                                                                  Ø
                                                                                                        CSS
                     SEMIGRAPHICS 4
                                                        Ø
                                                               Ø
Ø829
                                                                       Ø
                                                                              Ø
                                                                                                  Ø
                                                                                     х
                                                                                                        Х
                     64x64 COLOR GRAPHICS
Ø83Ø
                                                                                     Ø
                                                                                                        CSS
                                                        Ø
                                                               Ø
                                                                                             Ø
                                                                                                  Ø
                                                                       1
                                                                              1
Ø831
                     128x64 GRAPHICS
                                                        Ø
                                                               Ø
                                                                              1
                                                                                     α
                                                                                             Ø
                                                                                                  1
                                                                                                        CSS
Ø832
                     128x64 COLOR GRAPHICS
                                                        Ø
                                                                       Ø
                                                                                     Ø
                                                                                                  Ø
                                                                                                        CSS
Ø833
                     128x96 GRAPHICS
                                                                                                        CSS
                                                        Ø
                                                                                     Ø
                                                               1
                                                                       1
                                                                              1
                                                                                                  1
                                                                                             1
                     128x96 COLOR GRAPHICS
Ø834
                                                        1
                                                               Ø
                                                                      Ø
                                                                              1
                                                                                     1
                                                                                             Ø
                                                                                                  Ø
                                                                                                        CSS
Ø835
                     128x96 GRAPHICS
                                                        1
                                                               Ø
                                                                       1
                                                                              1
                                                                                     1
                                                                                             Ø
                                                                                                  1
                                                                                                        CSS
Ø836
                     128x96 COLOR GRAPHICS
                                                                                                        CSS
                                                               1
                                                                              1
                                                                                     1
                                                                                             1
Ø837
                     256x192 GRAPHICS
                                                                       Ø
                                                               1
                                                                                                        CSS
                                                                              1
                                                                                     1
                                                                                             1
Ø838
Ø839
                     * ALPHANUMERIC MODES
Ø84Ø
                                TEXT SCREEN MEMORY
Ø841
                                      EVEN BYTE
Ø842
                                                        CHARACTER BIT Ø
                                      BITØ
Ø843
                                      BIT1
                                                        CHARACTER BIT 1
                                                        CHARACTER BIT 2
Ø844
                                      BIT2
Ø845
                                      BIT3
                                                        CHARACTER BIT 3
                                                        CHARACTER BIT 4
Ø846
                                      BIT4
Ø847
                                      BIT5
                                                        CHARACTER BIT 5
Ø848
                                      BIT6
                                                        CHARACTER BIT 6
Ø849
                                      BIT7
Ø85Ø
                                      ODD BYTE
Ø851
                                                        BGNDØ BACKGROUND COLOR BIT (PALETTE ADDR)
0852
                                      RITØ
                                                        BGND1 BACKGROUND COLOR BIT (PALETTE ADDR)
Ø853
                                      BIT1
Ø854
                                      BIT2
                                                        BGND2
                                                               BACKGROUND COLOR BIT (PALETTE ADDR)
Ø855
                                      BIT3
                                                        FGBDØ FOREGROUND COLOR BIT (PALETTE ADDR)
                                                        FGND1
                                                               FOREGROUND COLOR BIT (PALETTE ADDR)
Ø856
                                      RIT4
                                                               FOREGROUND COLOR BIT (PALETTE ADDR)
Ø857
                                      BIT5
                                                        FGND2
Ø858
                                      BIT6
                                                        UNDLN CHARACTERS ARE UNDERLINED
                                                        BLINK CHARACTERS BLINK AT 1/2 SEC. RATE
Ø859
                                      BIT7
                                * ATTRIBUTES NOT AVAILABLE WHEN COCO = 1
0860
                     * GRAPHICS MODES
Ø861
Ø862
                                 16 COLOR MODES: (CRES1=1, CRESØ = Ø)
                                      BYTE FROM DRAM
Ø863
                                                        PAØ, SECOND PIXEL
Ø864
                                      BITØ
Ø865
                                      BIT1
                                                        PA1, SECOND PIXEL
Ø866
                                      BIT2
                                                        PA2, SECOND PIXEL
                                                        PA3, SECOND PIXEL
Ø867
                                      BIT3
Ø868
                                      BIT4
                                                        PAØ, FIRST PIXEL
Ø869
                                      BIT5
                                                        PA1, FIRST PIXEL
Ø87Ø
                                      BIT6
                                                        PA2, FIRST PIXEL
                                                        PA3, FIRST PIXEL
Ø871
                                      BIT7
Ø872
                                 4 COLOR MODES: (CRES1=0, CRES0 = 1)
                                      BYTE FROM DRAM
Ø873
                                                        PAØ, FOURTH PIXEL
Ø874
                                      BITØ
                                                        PA1, FOURTH PIXEL
Ø875
                                      BIT1
Ø876
                                      BIT2
                                                        PAØ, THIRD PIXEL
                                                        PA1, THIRD PIXEL
Ø877
                                      BIT3
                                                        PAØ, SECOND PIXEL
Ø878
                                      BIT4
Ø879
                                      BIT5
                                                        PA1, SECOND PIXEL
                                                        PAØ, FIRST PIXEL
Ø88Ø
                                      BIT6
Ø881
                                      BIT7
                                                        PA1, FIRST PIXEL
                                 2 COLOR MODES: (CRES1=0, CRES0 = 0)
Ø882
Ø883
                                      BYTE FROM DRAM
Ø884
                                                        PAØ, EIGHTH PIXEL
                                      BITØ
                                                        PAØ, SEVENTH PIXEL
Ø885
                                      BIT1
Ø886
                                      BIT2
                                                        PAØ, SIXTH PIXEL
Ø887
                                      BIT3
                                                        PAØ, FIFTH PIXEL
```

Ø888 Ø889 Ø890 Ø891 Ø892 Ø893 Ø894 Ø895 Ø899 Ø900 Ø901 Ø902 Ø903 Ø904 Ø905 Ø906 Ø907 Ø908 Ø909		* PALETTE	ADDRES	BIT4 BIT5 BIT6 BIT7 SSES ADDRESS FFBØ FFB1 FFB2 FFB3 FFB4 FFB5 FFB6 FFB7 FFB8 FFBB FFBB FFBB FFBB FFBB FFBB	PAØ, T PAØ, S	ORTH PIHIRD PIECOND PIRST PIRST PIPA2 Ø Ø Ø Ø I I I I Ø Ø Ø Ø I I I I Ø Ø Ø Ø I I I I I Ø Ø Ø Ø I I I I I I Ø Ø Ø Ø I	XEL	PAØ Ø 1 Ø 1 Ø 1 Ø 1 Ø 1 Ø 1 Ø 1 Ø 1 Ø 1
Ø91Ø Ø911	FF9A	V.BORDER	RMB	1	BORDER	REGIST	ER	
0912 0913 0914 0915 0916 0917 0918 0919 0920		BITØ BIT1 BIT2 BIT3 BIT4 BIT5 BIT6 BIT7	BLUØ GRNØ REDØ BLU1 GRN1 RED1	1	BUNDER	BLUE L GREEN RED LS BLUE M GREEN RED MS	SB LSB B ISB MSB	
Ø922			RMB	1	RESERV			
Ø923 Ø924	FF9C	V.SCROLL	RMB	1	VERTIC	AL SCRO	LL REGI	STER
0925 0926 0927 0928 0929 0930 0931 0932 0933	FF9C	BITØ BIT1 BIT2 BIT3 BIT4 BIT5 BIT6 BIT7 * IN COCO	VSCØ VSC1 VSC2 VSC3	THE VSC'S MUST	BE INIT	IALIZED	) TO \$ØF	
Ø934 Ø935	EEOD	V.OFSET1	RMB	1	VEDTIC	۸۱ NEES	SET 1 RE	CISTED
Ø936				•	V LIVI 10	ME UITS	'FI T KE	MISILK
Ø937	FF9D	BITØ	Y11					
Ø938 Ø939		BIT1 BIT2	Y12 Y13					
Ø94Ø		BIT3	Y14					
Ø941		BIT4	Y15					
Ø942		BIT5	Y16					
Ø943		BIT6	Y17					
Ø944 Ø945		BIT7	Y18					
Ø946	FF9F	V.OFSETØ	RMB	1	VERTIC	AI OFFS	SET Ø RE	GISTER
Ø947				-				
Ø948	FF9E	BITØ	Y3					
Ø949		BIT1	Y4					
Ø95Ø		BIT2	Y5					
Ø951		BIT3 BIT4	Y6					
Ø952 Ø953		BIT5	Y7 Y8					
Ø954		BIT6	Y9					
Ø955		BIT7	Y1Ø					
Ø956				Y9-Y15 ARE NOT				
Ø957		SAM BITS	⊦Ø-F6	. ALSO IN COCO	MUDE, Y	16-Y18	2HUULD	BE 1, ALL OTHERS Ø
Ø958 Ø959	FFQF	H.OFSETØ	RMB	1	HODITO	NTAI OF	ESET W	REGISTER
Ø96Ø	11.71	מו שבו אוויייי	KIID	-	1101(120	MIAL UF	ו שבו ש	REGISTER
Ø961	FF9F	BITØ	ΧØ		HORIZO	NTAL OF	FSET AD	DRESS

0962 0963 0964 0965 0966		BIT1 BIT2 BIT3 BIT4 BIT5 BIT6	X1 X2 X3 X4 X5 X6		HORIZO HORIZO HORIZO HORIZO	ONTAL OF ONTAL OF ONTAL OF ONTAL OF ONTAL OF	FSET AL FSET AL FSET AL FSET AL	DDRESS DDRESS DDRESS DDRESS		
Ø968		BIT7	XVEN			ONTAL VI				
Ø969				A HORIZONTAL SCR					RDLESS (	OF THE
Ø97Ø				CRES BITS SELEC						
Ø971				ER THAN THE DISP						
Ø972		'WINDOW'	(THE	DISPLAYED SCREE	N) BY M	MEANS OF	THE HO	ORIZONTA	AL OFFS	ET
Ø973				RACTER MODE, THE				CHARACT	TERS RE	GARDLESS
Ø974		OF ATTRI	BUTE	(OR 64, IF DOUBL	E-WIDE	IS SELE	CTED).			
Ø975										
Ø976	FFAØ	MMUREG	RMB	16	MEMORY	Y MANAGE	MENT UN	NIT REG	ISTERS	(6 BITS)
Ø977		- DELATION	CUID	DETUEEN DATA IN	TACK DE		AND 051	IEDATED	4 D D D E C	•
Ø978		* RELATION	IZHIP	BETWEEN DATA IN						
Ø979				BIT	D5	D4	D3	D2	D1	DØ
Ø98Ø Ø981				CORRESPONDING MEMORY ADDRESS	A 1 Q	A17	A16	A15	A14	A13
Ø982				FILFIORT ADDRESS	AIO	AI/	AIO	AID	A14	AIS
Ø983		* DATA FRO	M THE	MMU IS THEN USE	D AS TH	IF IIPPER	6 ADDE	PESS LIN	JFS (Δ1	3-418)
Ø984		FOR MEMO			D A3 11	IL OITEK	O ADDI	(L33 L1)	1L5 (A1.	J A10)
Ø985				ADDRESS RANGE	TR	A15	A14	A13	MMU LO	CATION
Ø986				X0000 - X1FFF	Ø	Ø	Ø	Ø	FFAØ	
Ø987				X2000 - X3FFF	Ø	Ø	Ø	1	FFA1	
Ø988				X4000 - X5FFF	Ø	Ø	1	Ø	FFA2	
Ø989				X6000 - X7FFF	Ø	Ø	1	1	FFA3	
Ø99Ø				X8000 - X9FFF	Ø	1	Ø	Ø	FFA4	
0991				XAØØØ - XBFFF	Ø	1	Ø	1	FFA5	
Ø992				XCØØØ - XDFFF	Ø	1	1	Ø	FFA6	
Ø993				XEØØØ - XFFFF	Ø	1	1	1	FFA7	
Ø994				V2222 V4555		~	~	~		
Ø995				X0000 - X1FFF	1	Ø	Ø	Ø	FFA8	
Ø996				X2000 - X3FFF	1	Ø	Ø	1	FFA9	
Ø997 Ø998				X4000 - X5FFF X6000 - X7FFF	1 1	Ø Ø	1 1	Ø 1	FFAA FFAB	
Ø999				X8000 - X7FFF X8000 - X9FFF	1	1	Ø	Ø	FFAC	
1000				XAØØØ - XBFFF	1	1	Ø	1	FFAD	
1000				XCØØØ - XDFFF	1	1	1	Ø	FFAE	
1002				XEØØØ - XFFFF	1	1	1	1	FFAF	
1003				A-222 A	-	-	-	-		
1004	FFBØ	PALETREG	RMB	16	COLOR	PALETTE	REGIST	TERS (6	BITS)	
1005										
1006				DATA BIT	D5	D4	D3	D2	D1	DØ
1007				RGB OUTPUT	R1	G1	B1	RØ	GØ	BØ
1008				COMP. OUTPUT	I 1	ΙØ	P3	P2	P1	PØ
1009		+ FOD COCO	COMP	ATIDILITY THE E	01.1.01.11	10 0110111	D DE 16	\		A I T 7 A T T O NI
1010 1011				ATIBILITY, THE F FOR PAL VERSION,					I INTII	ALIZATION
1011		(NGD VAL		GREEN	\$12	. IADLL	TOK CON	11.02115		
1013				YELLOW	\$36					
1014				BLUE	\$09					
1015			FFB3		\$24					
1016			FFB4	BUFF	\$3F					
1017			FFB5	CYAN	\$10					
1018			FFB6	MAGENTA	\$2D					
1019				ORANGE	\$26					
1020				BLACK	\$00					
1021				GREEN	\$12					
1022				BLACK	\$00					
1023				BUFF	\$3F					
1024 1025				BLACK GREEN	\$ØØ \$12					
1025				BLACK	\$00					
1027				ORANGE	\$26					
1027				J.WIIIME	720					
1029	FFCØ	SAMREG	EQU	*	SAM CO	ONTROL R	EGISTER	RS		
1030					2			•		
1031	FFCØ	VØCLR	RMB	1	CLEAR	COCO GR	APHICS	MODE V	ð	
1032	FFC1	VØSET	RMB	1	SET CO	OCO GRAP	HICS MO	DDE VØ		
1033		V1CLR	RMB	1		COCO GR			1	
1034		V1SET	RMB	1		OCO GRAP			_	
1000	EEC /	V2CLR	RMB	1	CLEAR	COCO GR	APHICS	MODE V2	2	
1035	FFC4									

1036	FFC5	V2SET	RMB	1	SET COCO GRAPHICS MODE V2
1037	FFC6	FØCLR	RMB	1	CLEAR COCO GRAPHICS OFFSET FØ
1038	FFC7	FØSET	RMB	1	SET COCO GRAPHICS OFFSET FØ
1039	FFC8	F1CLR	RMB	1	CLEAR COCO GRAPHICS OFFSET F1
1040	FFC9	F1SET	RMB	1	SET COCO GRAPHICS OFFSET F1
1041	FFCA	F2CLR	RMB	1	CLEAR COCO GRAPHICS OFFSET F2
1042	FFCB	F2SET	RMB	1	SET COCO GRAPHICS OFFSET F2
1043	FFCC	F3CLR	RMB	1	CLEAR COCO GRAPHICS OFFSET F3
1044	FFCD	F3SET	RMB	1	SET COCO GRAPHICS OFFSET F3
1045	FFCE	F4CLR	RMB	1	CLEAR COCO GRAPHICS OFFSET F4
1046	FFCF	F4SET	RMB	1	SET COCO GRAPHICS OFFSET F4
1047	FFDØ	F5CLR	RMB	1	CLEAR COCO GRAPHICS OFFSET F5
1048	FFD1	F5SET	RMB	1	SET COCO GRAPHICS OFFSET F5
1049	FFD2	F6CLR	RMB	1	CLEAR COCO GRAPHICS OFFSET F6
1050	FFD3	F6SET	RMB	1	SET COCO GRAPHICS OFFSET F6
1051	FFD4		RMB	4	RESERVED
1052	FFD8	R1CLR	RMB	1	CLEAR CPU RATE, (Ø.89 MHz)
1053	FFD9	R1SET	RMB	1	SET CPU RATE, (1.78 MHz)
1054	FFDA		RMB	4	RESERVED
1055	FFDE	ROMCLR	RMB	1	ROM DISABLED
1056	FFDF	ROMSET	RMB	1	ROM ENABLED
1057					
1058	FFEØ		RMB	18	RESERVED FOR FUTURE MPU ENHANCEMENTS
1059		*	INTER	RUPT VECTORS	
1060	FFF2	SWI3	RMB	2	
1061	FFF4	SWI2	RMB	2	
1062	FFF6	FIRQ	RMB	2	
1063	FFF8	IRQ	RMB	2	
1064	FFFA	SWI	RMB	2	
1065	FFFC	NMI	RMB	2	
1066	FFFE	RESETV	RMB	2	

```
aaa1
                 аа ға
                                          DHITOK
                                                      EOU
                                                                                                              HIGHEST 1.1 DISK TOKEN
                 00 32
0002
                                          CYEAR
                                                       EQU
аааз
0004
aaas
0006
                                          **** FILE ALLOCATION TABLE FORMAT
aaaz
0008
0009
                                          * THE FILE ALLOCATION TABLE (FAT) CONTAINS THE STATUS OF THE GRANULES ON A DISKETTE.
* THE FAT CONTAINS 6 CONTROL BYTES FOLLOWED BY 68 DATA BYTES (ONE PER GRANULE). ONLY THE
aa1a
0011
                                          * FIRST TWO OF THE SIX CONTROL BYTES ARE USED. A VALUE OF $FF IS SAVED IN UNALLOCATED * GRANULES. IF BITS 6 & 7 OF THE DATA BYTE ARE SET, THE GRANULE IS THE LAST GRANULE
ØØ12
0013
0014
                                          * IN A FILE AND BITS 0-5 ARE THE NUMBER OF USED SECTORS IN THAT GRANULE. IF BITS 6 & 7 * ARE NOT SET, THE DATA BYTE CONTAINS THE NUMBER OF THE NEXT GRANULE IN THE FILE.
0015
ØØ16
                                           * OFFSETS TO FAT CONTROL BYTES
0017
                                                                                                              ACTIVE FILE COUNTER : DISK TO RAM FAT IMAGE DISABLE VALID DATA FLAG: Ø=DISK DATA VALID, <\!\!> Ø = NEW FAT DATA - DISK DATA INVALID
ØØ18
                 00 00
0019
                 00 01
                                          FAT1
                                                      EOU
0020
0021
                                                              2 TO 5
                                                                                                              NOT USED
                 ØØ Ø6
                                           FATCON
                                                      EQU
                                                                                                              OFFSET TO START OF FAT DATA (68 BYTES)
0022
0023
ØØ24
0025
                                          **** DIRECTORY ENTRY FORMAT
0026
ØØ27
                                           * THE DIRECTORY IS USED TO KEEP TRACK OF HOW MANY FILES ARE STORED ON A DISKETTE
* AND WHERE THE FILE IS STORED ON THE DISK. THE FIRST GRANULE USED BY THE FILE WILL
* ALLOW THE FAT TO TRACK DOWN ALL OF THE GRANULES USED BY THE FILE. IF THE FIRST
0028
ØØ29
0030
ØØ31
                                           * BYTE OF THE DIRECTORY ENTRY IS ZERO, THE FILE HAS BEEN KILLED;
0032
                                           * IF THE FIRST BYTE IS $FF THEN THE DIRECTORY ENTRY HAS NEVER BEEN USED.
ØØ33
                                                               RYTE
                                                                                                              DESCRIPTION
ØØ34
ØØ35
                                                                                                              FILE NAME
FILE EXTENSION
0036
                00 00
                                          DIRNAM
                                                      EOU
                                                              Ø
                 ØØ Ø8
                                          DIREXT
0037
                                                       EOU
ØØ38
                 ØØ ØB
                                          DIRTYP
                                                               11
                                                                                                              FILE TYPE
                                                       EQU
0039
                 ØØ ØC
                                          DIRASC
                                                       EOU
                                                              12
                                                                                                              ASCII FLAG
0040
                                                                                                              FIRST GRANULE IN FILE
                                           DIRGRN
                                                                                                              NUMBER OF BYTES IN LAST SECTOR
0041
                 ØØ ØE
                                          DIRLST
                                                      EOU
                                                              14
0042
                                                               16 TO 31
0043
                                           **** FILE CONTROL BLOCK FORMAT
0045
0046
αα47
                                             THE FILE STRUCTURE OF COLOR TRS DOS IS CONTROLLED BY A FILE CONTROL BLOCK (FCB)
0048
                                          * THE FILE SINCUINE DE COURT HS DUSIS CONTROLLED HS A FILE CONTROL BLOCK FOR THE FOR CONTROL BYTES CONTROL BYTES AND A SECTOR LONG (256 BYTES) DATA BUFFER.

* THE CONTROL BYTES CONTROL THE ORDERLY FLOW OF DATA FROM THE COMPUTER'S RAM TO

* THE DISKETTE AND VICE VERSA. THE OPEN COMMAND INITIALIZES THE FCB; THE INPUT,

* OUTPUT, WRITE, PRINT, GET AND PUT COMMANDS TRANSFER DATA THROUGH THE FCB AND

* THE CLOSE COMMAND TURNS OFF THE FCB.
ØØ49
0050
0051
0052
ØØ53
0054
                                          * TABLES OF OFFSETS TO FCB CONTROL BYTES
0056
ØØ57
                                          ***** RANDOM FILE
0058
                                                              BYTE
                                                                                                              DESCRIPTION
                                          FCBTYP
                                                                                                              FILE TYPE: $40=RANDOM/DIRECT, Ø=CLOSED
0059
                 00 00
                                                       EOU
9969
                 00 01
                                           FCBDRV
                                                       EQU
                                                                                                              DRIVE NUMBER
                                                                                                              FIRST GRANULE IN FILE
                 00 02
                                           FCBFGR
0061
                                                       EOU
                                                                                                               CURRENT GRANULE BEING USED
ØØ62
0063
                 00 04
                                           FCBSEC
                                                      EOU
                                                                                                              CURRENT SECTOR BEING USED (1-9)
                                           ECRPOS
                                                                                                              CURRENT PRINT POSITION - ALWAYS ZERO IN RANDOM FILES
9965
                 98 96
                                                       FOII
                                                                                                              CURRENT RECORD NUMBER
ØØ66
                 ØØ Ø7
                                           FCBREC
                                                                                                              RANDOM FILE RECORD LENGTH
0067
                 ØØ Ø9
                                           FCBRLN
                                                       E0U
                                                                                                              PRINTER TO START OF THIS FILE'S RANDOM ACCESS BUFFER SECTOR OFFSET TO CURRENT POSITION IN RECORD GET/PUT FLAG: Ø=PUT, 1=PUT
                                           FCBBUF
ØØ68
                 ØØ ØB
                                                       EQU
                                                              11
aasa
                 aa ab
                                           FCRSOF
                                                       FOII
                                                               13
                 ØØ ØF
                                           FCBFLG
0070
                                                       EQU
                                                               15
                                                                                                              OF THE CURRENT RECORD OF THE CURRENT RECORD
                                                               16,17
0071
                                           FCBDIR
                 00 12
                                                      EQU
0072
                                                              18
0073
                                           FCBLST
0074
                 00 15
                                           FCBGET
                                                      EOU
                                                              21
ØØ75
                                                                                                               'PUT' RECORD COUNTER: POINTER TO WHERE IN THE RECORD THE NEXT
0076
                 ØØ 17
                                          FCBPUT
                                                      EQU
                                                              23
                                                                                                              BYTE WILL BE 'PUT'
OFFSET TO START OF FCB DATA BUFFER (256 BYTES)
0077
0078
                 ิติต 19
                                          FCBCON
                                                      EQU
                                                             25
ØØ79
ดดลด
                                           **** SEQUENTIAL FILE
                                                                                                              DESCRIPTION
0081
                                                              BYTE
0082
                 аа аа
                                          ECRTYP
                                                       FOII
                                                                                                              FILE TYPE: $10=INPUT, $20=OUTPUT, 0=CLOSED DRIVE NUMBER
                00 01
                                           FCBDRV
0083
                                                       EOU
ØØ84
                                           FCBFGR
                                                                                                              FIRST GRANULE IN FILE
                                                                                                              CURRENT GRANULE BEING USED
0085
                 00 03
                                           FCBCGR
                                                       EOU
                                                              3
                                                                                                              CURRENT SECTOR BEING USED (1-9)
ØØ86
                                           FCBSEC
                                                       EQU
                                                                                                              INPUT FILE: CHARACTER POINTER - POINTS TO NEXT CHARACTER IN
ØØ87
                                          ECRCPT
                                                       FOII
                                                               5
                                                                                                               FILE TO BE PROCESSED.
ØØ88
                                                                                                              OUTPUT FILE: FULL SECTOR FLAG - IF IT IS 1 WHEN THE FILE IS CLOSED IT MEANS 256 BYTES OF THE LAST SECTOR HAVE BEEN USED.
ดดลจ
0090
ØØ91
                 ØØ Ø6
ØØ Ø7
                                          FCBPOS
FCBREC
                                                      FOII
                                                                                                              CURRENT PRINT POSITION
CURRENT RECORD NUMBER: HOW MANY WHOLE SECTORS HAVE BEEN
0092
                                                       EQU
ØØ93
                                                                                                              INPUT OR OUTPUT TO A FILE.
                                                                                                              UNUSED
                                                              9 TO 15
0094
                                           FCBRLN
ØØ95
                                                       EQU
                                                                                                              CACHE FLAG: ØØ=CACHE EMPTY, $FF=CACHE FULL
                                                                                                              CACHE DATA BYTE
0096
                 ØØ 11
                                          FCBCDT
                                                      EOU
                                                              17
```

ØØ97	00 12	FCBDIR	EQU	18	DIRECTORY ENTRY NUMBER (0-71)
0098	00 13	FCBLST	EQU	19	NUMBER OF BYTES IN LAST SECTOR OF FILE
ØØ99 Ø1ØØ	ØØ 17	* FCBDFL	EQU	21,22 23	UNUSED INPUT FILE ONLY: DATA LEFT FLAG: Ø=DATA LEFT, \$FF=NO DATA (EMPTY)
0101	ØØ 18	FCBLFT	EQU	24	NUMBER OF CHARACTERS LEFT IN BUFFER (INPUT FILE)
0102	gg 10	*	FOII	25	NUMBER OF CHARS STORED IN BUFFER (OUTPUT FILE)
Ø1Ø3 Ø1Ø4	ØØ 19	FCBCON	EQU	25	OFFSET TO FCB DATA BUFFER (256 BYTES)
0105			ORG	\$CØØØ	
Ø1Ø6	CGGG 44 4D	DOCDAC	F00	IDKI	
Ø1Ø7 Ø1Ø8	C000 44 4B C002 20 08	DOSBAS LCØØ2	FCC BRA	'DK' LCØØC	
0109	0002 20 00	20002	5.01	2000	
0110	CØØ4 D7 5F	DCNVEC	FDB	DSKCON	DSKCON POINTER
Ø111 Ø112	CØØ6 ØØ EA CØØ8 DF 4C	DSKVAR DSINIT	FDB FDB	DCOPC DOSINI	ADDRESS OF DSKCON VARIABLES DISK INITIALIZATION VECTOR
Ø113	CØØA DF ØØ	DOSVEC	FDB	DOSCOM	DOS COMMAND VECTOR
0114		755		TUE DAM HOED DV DAGU DAGA	
Ø115 Ø116	CØØC 8E Ø6 ØØ	LCØØC	LDX	THE RAM USED BY DISK BASIC #DBUFØ	POINT X TO START OF DISK RAM
Ø117	CØØF 6F 8Ø	LCØØF	CLR	, X+	CLEAR A BYTE
Ø118	CØ11 8C Ø9 89			#DFLBUF	END OF DISK'S RAM?
Ø119 Ø12Ø	CØ14 26 F9 CØ16 8E C1 Ø9		BNE LDX	LC00F #LC109	NO - KEEP CLEARING POINT X TO ROM IMAGE OF COMMAND INTERPRETATION TABLE
Ø121	CØ19 CE Ø1 34		LDU	#COMVEC+2Ø	POINT U TO RAM ADDRESS OF SAME
Ø122	CØ1C C6 ØA		LDB	#10	10 BYTES PER TABLE
Ø123 Ø124	CØ1E BD A5 9A CØ21 CC B2 77		JSR LDD	LA59A #LB277	MOVE (B) BYTES FROM (X) TO (U) SYNTAX ERROR ADDRESS
Ø124	CØ24 ED 43		STD	\$03,U	* SET JUMP TABLE ADDRESSES OF THE USER COMMAND
Ø126	CØ26 ED 48		STD	,	* INTERPRETATION TABLE TO POINT TO SYNTAX ERROR
Ø127 Ø128	CØ28 6F C4 CØ2A 6F 45		CLR	,U \$05,U	CLEAR BYTE Ø OF USER TABLE (DOESN'T EXIST FLAG) SET NUMBER OF SECONDARY USER TOKENS TO ZERO
Ø129	CØ2C CC CF ØA		LDD	#DXCVEC	* SAVE NEW
Ø13Ø	CØ2F FD Ø1 2D		STD	COMVEC+13	* POINTERS TO EXBAS
Ø131 Ø132	CØ32 CC CF 32 CØ35 FD Ø1 32		LDD STD	#DXIVEC COMVEC+18	* COMMAND AND SECONDARY * COMMAND INTERPRETATION ROUTINES
Ø132	C035 FD 01 32	**** MOV		NEW RAM VECTORS FROM ROM TO RAM	" COMMAND INTERFRETATION ROUTINES
Ø134	CØ38 CE Ø1 5E		LDU	#RVECØ	POINT U TO 1ST RAM VECTOR
Ø135 Ø136	CØ3B 86 7E CØ3D B7 Ø1 AØ	LCØ3B	LDA STA	#\$7E RVEC22	OP CODE OF JMP INSTRUCTION SET 1ST BYTE OF 'GET'/'PUT' RAM VECTOR TO 'JMP'
Ø130	CØ4Ø A7 CØ		STA	,U+	SET 1ST BYTE OF GET / FOT KAN VECTOR TO SMI
Ø138	CØ42 EC 81		LDD	, X++	GET RAM VECTOR FROM ROM
Ø139 Ø14Ø	CØ44 ED C1 CØ46 8C C1 39		STD	,U++ #LC139	STORE IT IN RAM COMPARE TO END OF ROM VALUES
Ø141	CØ49 26 FØ		BNE	LCØ3B	BRANCH IF NOT ALL VECTORS MOVED
Ø142	CØ4B 8E C2 B2		LDX	#DVEC22	GET ROM VALUE OF 'GET'/'PUT' RAM VECTOR
Ø143 Ø144	CØ4E BF Ø1 A1 CØ51 8E C8 BØ		STX LDX	RVEC22+1 #DVEC20	SAVE IT IN RAM GET DISK COMMAND INTERPRETATION LOOP RAM VECTOR
Ø145	CØ54 BF Ø1 9B		STX	RVEC20+1	SAVE IN RAM VECTOR TABLE
Ø146		**** INI		ZE DISK BASIC'S USR VECTORS	
Ø147 Ø148	CØ57 8E Ø9 5F CØ5A 9F BØ		LDX STX	#DUSRVC USRADR	POINT X TO START OF DISK BASIC USR VECTORS SAVE START ADDRESS IN USRADR
Ø149	CØ5C CE B4 4A		LDU	#LB44A	POINT U TO ADDRESS OF 'FUNCTION CALL' ERROR
Ø15Ø	CØ5F C6 ØA		LDB	#\$ØA	10 USER VECTORS TO INITIALIZE
Ø151 Ø152	CØ61 EF 81 CØ63 5A	LCØ61	STU DECB	,X++	SET USR VECTOR TO 'FC' ERROR DECREMENT USR VECTOR COUNTER
Ø152	CØ64 26 FB		BNE	LCØ61	BRANCH IN NOT DONE WITH ALL 10 VECTORS
Ø154	CØ66 8E D8 A1		LDX	#DNMISV	GET ADDRESS OF NMI SERVICING ROUTINE
Ø155 Ø156	CØ69 BF Ø1 ØA CØ6C 86 7E		STX LDA	NMIVEC+1 #\$7E	SAVE IT IN NMI VECTOR OP CODE OF JMP
Ø157	CØ6E B7 Ø1 Ø9		STA	NMIVEC	MAKE THE NMI VECTOR A JMP
Ø158	CØ71 8E D8 AF		LDX	#DIRQSV	GET ADDRESS OF DISK BASIC IRQ SERVICING ROUTINE
Ø159 Ø16Ø	CØ74 BF Ø1 ØD CØ77 86 13		STX LDA	IRQVEC+1 #\$13	SAVE IT IN IRQVEC = INITIALIZE WRITE FAT
Ø161	CØ79 B7 Ø9 7A		STA	WFATVL	= TO DISK TRIGGER VALUE
Ø162	C07C 7F 08 00		CLR	FATBLØ	* * INITIALIZE THE ACTIVE ELLE COUNTED OF
Ø163 Ø164	CØ7F 7F Ø8 4A CØ82 7F Ø8 94			FATBL1 FATBL2	* INITIALIZE THE ACTIVE FILE COUNTER OF * EACH FAT TO ZERO. THIS WILL CAUSE THE FATS
Ø165	CØ85 7F Ø8 DE		CLR	FATBL3	* TO THINK THERE ARE NO ACTIVE FILES
Ø166 Ø167	CØ88 8E Ø9 89 CØ8B BF Ø9 48			#DFLBUF RNBFAD	= GET THE STARTING ADDRESS OF THE = RANDOM FILE BUFFER FREE AREA AND DAVE IT AS THE
Ø168	לע וט טטעט 40	*	311	MIDI AD	= START ADDRESS OF FREE RAM FOR RANDOM FILE BUFFERS
Ø169	CØ8E 3Ø 89 Ø1 ØØ			\$Ø100,X	SAVE 256 BYTES FOR RANDOM FILE BUFFERS INITIALLY
Ø17Ø Ø171	CØ92 BF Ø9 4A CØ95 3Ø Ø1			FCBADR \$01,X	SAVE START ADDRESS OF FCBS * ADD ONE AND SAVE THE STARTING
Ø171 Ø172	CØ97 BF Ø9 28			FCBV1	* ADDRESS OF FCB1
Ø173	CØ9A 6F ØØ			FCBTYP,X	CLEAR THE FIRST BYTE OF FCB 1 (CLOSE FCB)
Ø174 Ø175	CØ9C 3Ø 89 Ø1 19 CØAØ BF Ø9 2A			FCBLEN, X FCBV1+2	POINT X TO FCB 2 SAVE ITS STARTING ADDRESS IN FCB VECTOR TABLE
Ø176	CØA3 6F ØØ			FCBTYP,X	CLEAR THE FIRST BYTE OF FCB 2 (CLOSE FCB)
Ø177	CØA5 3Ø 89 Ø1 19		LEAX	FCBLEN, X	* POINT X TO SYSTEM FCB - THIS FCB WILL ONLY
Ø178 Ø179	CØA9 BF Ø9 2C	*	STX	FCBV1+4	* BE USED TO COPY, LOAD, SAVE, MERGE, ETC SAVE ITS ADDRESS IN THE FCB VECTOR TABLE
Ø18Ø	CØAC 6F ØØ			FCBTYP,X	CLEAR THE FIRST BYTE OF SYSTEM FCB (CLOSE FCB)
Ø181	CØAE 86 Ø2			#\$Ø2	* SET THE NUMBER OF ACTIVE RESERVED
Ø182 Ø183	CØBØ B7 Ø9 5B CØB3 3Ø 89 Ø1 19			FCBACT FCBLEN,X	* FILE BUFFERS TO 2 (1,2) POINT X TO ONE PAST THE END OF SYSTEM FCB
Ø184	CØB7 1F 1Ø		TFR		SAVE THE ADDRESS IN ACCD
Ø185	CØB9 5D		TSTB	LCARD	ON AN EVEN 256 BYTE BOUNDARY?
Ø186 Ø187	CØBA 27 Ø1 CØBC 4C		BEQ INCA	LCØBD	YES NO - ADD 256 TO ADDRESS
Ø188	CØBD 85 Ø1	LCØBD	BITA	#\$01	* CHECK TO SEE IF ACCD IS ON AN EVEN
Ø189	CØBF 27 Ø1			LCØC2	* 512 BYTE (ONE GRAPHIC PAGE) BOUNDARY - ADD
Ø19Ø Ø191	CØC1 4C CØC2 1F 89	LCØC2	INCA TFR	A,B	* 256 (INCA) TO IT IF NOT COPY ACCA TO ACCB
Ø192	CØC4 CB 18			#\$18	SAVE ENOUGH ROOM FOR 4 GRAPHICS PAGES (PCLEAR 4)
2232					

Ø193	CØC6 D7 19	S	STB TXTTAB ISR L96EC DA BEGGRP IDDA #\$Ø6 STA ENDGRP ISR [DSINIT] ISR LCØFØ INDCC #SAF IDX #LC139-1 ISR STRINOUT IDX #DKWMST ISTX RSTVEC	SAVE NEW START OF BASIC ADDRESS
Ø194	CØC8 BD 96 EC	J	ISR L96EC	INITIALIZE EXBAS VARIABLES & DO A NEW
Ø195 Ø196	CØCB 96 BA CØCD 8B Ø6	L A	DA BEGGRP	GET THE START OF CURRENT GRAPHICS PAGE ADD 1.5K (6 X 256 = ONE GRAPHICS PAGE)
Ø190		S	STA ENDGRP	SAVE NEW END OF GRAPHICS PAGE
Ø198	CØCF 97 B7 CØD1 AD 9F CØ Ø8 CØD5 8D 19	J	JSR [DSINIT]	INITIALIZE SWI2,3 JUMP ADDRESSES
		В	SSR LCØFØ	GO INITIALIZE THE FLOPPY DISK CONTROLLER
	CØD7 1C AF	A	NDCC #\$AF	TURN ON IRQ AND FIRQ
Ø2Ø1 Ø2Ø2	CØD9 8E C1 38 CØDC BD B9 9C	L	DX #LC139-1	POINT X TO DISK BASIC COPYRIGHT MESSAGE PRINT COPYRIGHT MESSAGE TO SCREEN
	CØDF 8E CØ E7	ı	DX #DKWMST	GET DISK BASIC WARM START ADDRESS
	CØE2 9F 72	S	STX RSTVEC	SAVE IT IN RESET VECTOR
0205	CØE4 7E AØ E2	J	IMP LAØE2	JUMP BACK TO BASIC
0206				
	CØE7 12 CØE8 8D Ø6	DKWMST N	IOP BSR LCØFØ	WARM START INDICATOR INITIALIZE THE FLOPPY DISK CONTROLLER
	CØEA BD D2 D2		ISR LD2D2	CLOSE FILES AND DO MORE INITIALIZATION
	CØED 7E 8Ø CØ		IMP XBWMST	JUMP TO EXBAS' WARM START
Ø211			CLR NMIFLG	RESET NMI FLAG
	CØF3 7F Ø9 85		CLR RDYTMR	RESET DRIVE NOT READY TIMER
	CØF6 7F Ø9 86 CØF9 7F FF 4Ø	C	CLR DRGRAM	RESET RAM IMAGE OF DSKREG (MOTORS OFF) RESET DISK CONTROL REGISTER
	CØFC 86 DØ	Ĺ	DA #\$DØ	FORCE INTERRUPT COMMAND OF 1793
Ø216	CØFE B7 FF 48	S	STA FDCREG	SEND IT TO 1793
	C1Ø1 1E 88	E	CLR DSKREG  LDA #5DØ  STA FDCREG  CXG A,A  CXG A,A  LDA FDCREG	* DELAY
	C103 1E 88 C105 B6 FF 48	E	IXG A,A	* DELAY SOME MORE GET 1793 STATUS (CLEAR REGISTER)
	C108 39	R	RTS	del 1793 STATUS (CLEAR REGISTER)
Ø221				
Ø222	0100 14		SIC COMMAND INTERP TABLES	OR DICK DACTO 4 4 COMMINGS
	C109 14 C10A C1 92	LC109 F	TCB	20 DISK BASIC 1.1 COMMANDS DISK BASIC'S COMMAND DICTIONARY
	C1ØC C2 38		DB LC238	COMMAND JUMP TABLE
	C10E 06		CB Ø6	6 DISK BASIC SECONDARY FUNCTIONS
	C1ØF C2 19		DB LC219	SECONDARY FUNCTION TABLE
Ø228 Ø229	C111 C2 4E	F	DB LC24E	SECONDARY FUNCTION JUMP TABLE
Ø23Ø		* RAM HOOF	KS FOR DISK BASIC	
Ø231	C113 C4 4B C8 88 C8 93	R I C 1 1 3 F	DB DVECØ DVEC1 DVEC2	
Ø232	C119 CC 1C C5 BC C8 48	3 F	DB DVEC3,DVEC4,DVEC5	
Ø233	C11F C8 4B CA E9 CA F9	) F	DB DVEC3, DVEC4, DVEC5 DB DVEC6, DVEC7, DVEC8 DB XVEC9, DVEC10, DVEC11 DB DVEC12, DVEC13, DVEC14 DB DVEC15, DVEC12	
Ø235	C12B C6 E4 CA E4 C9 Ø	, . . F	DB DVEC12,DVEC13,DVEC14	
Ø236	C131 CE D2 C6 E4	F	DB DVEC15, DVEC12	
0237	C135 C2 65 CA 3E	F	DB DVEC17, DVEC18	
Ø238 Ø239		* DISK RAS	SIC COPYRIGHT MESSAGE	
	C139 44 49 53 4B 2Ø 45		CC 'DISK EXTENDED COLOR BASIC 1.1'	
	C13F 58 54 45 4E 44 45			
	C145 44 20 43 4F 4C 4F C14B 52 20 42 41 53 49			
Ø243		,		
Ø245	C156 ØD		FCB CR	
	C157 43 4F 50 59 52 49		DB 'COPYRIGHT (C) 198'	
	C15D 47 48 54 20 28 43 C163 29 20 31 39 38	•		
Ø249	C168 32		CB CYEAR	
	C169 20 42 59 20 54 41 C16F 4E 44 59	L F	CC ' BY TANDY'	
	C172 ØD	F	FCB CR	
	C173 55 4E 44 45 52 20		CC 'UNDER LICENSE FROM MICROSOFT'	
	C179 4C 49 43 45 4E 53			
	C17F 45 20 46 52 4F 4E C185 20 4D 49 43 52 4F			
Ø256 Ø257	C18B 53 4F 46 54			
Ø258	C18F ØD ØD ØØ	F	DB CR,CR,Ø	
Ø259				
Ø26Ø Ø261		* DISK RAS	SIC COMMAND DICTIONARY TABLE	
Ø262		*		TOKEN #
	C192 44 49 D2		CC 'DI',\$8Ø+'R'	CE
	C195 44 52 49 56 C5		FCC 'DRIV',\$80+'E'	CF Da
	C19A 46 49 45 4C C4 C19F 46 49 4C 45 D3		FCC 'FIEL',\$8Ø+'D' FCC 'FILE',\$8Ø+'S'	DØ D1
Ø267	C1A4 4B 49 4C CC	F	CC 'KIL',\$80+'L'	D2
Ø268	C1A8 4C 4F 41 C4	F	CC 'LOA',\$8Ø+'D'	D3
ย269 ด27ด	C1AC 4C 53 45 D4 C1BØ 4D 45 52 47 C5 C1B5 52 45 4E 41 4D C5 C1BB 52 53 45 D4	F	'LSE',\$80+' ' 'MERG' \$80+'F'	D4 D5
0270	C1B5 52 45 4E 41 4D C	, 5 F	CC 'RENAM',\$8Ø+'E'	D6
	0400 50 50 45 04		CC 'RSE' \$80+'T'	D7
0272	CIBB 52 53 45 D4	F	00 102 ,705 1	
Ø272 Ø273	C1BB 52 53 45 D4 C1BF 53 41 56 C5	F	CC 'SAV',\$80+'E'	D8
0272 0273 0274	C1BB 52 53 45 D4 C1BF 53 41 56 C5 C1C3 57 52 49 54 C5	F F F	CCC 'SAV',\$80+'E' CCC 'WRIT',\$80+'E'	D8 D9
0272 0273 0274 0275 0276	C1BB 52 53 45 D4 C1BF 53 41 56 C5 C1C3 57 52 49 54 C5 C1CB 56 45 52 49 46 D5 C1CE 55 4E 4C 4F 41 C4	F F F F F	CC 'SAV',880+'E' CC 'WRIT',880+'E' CC 'VERIF',880+'P' CC 'UNLOA',\$80+'P'	D8
0272 0273 0274 0275 0276 0277	C1BB 52 53 45 D4 C1BF 53 41 56 C5 C1C3 57 52 49 54 C5 C1C8 56 45 52 49 46 D5 C1CE 55 4E 4C 4F 41 C4 C1D4 44 53 4B 49 4E C5	F F F F 1 F	CCC 'SAV',\$88+'E' CCC 'WRIT',\$80+'E' CCC 'VERIF',\$80+'Y' CCC 'UNLOA',\$80+'D' CCC 'DSKIN',\$80+'I'	D8 D9 DA DB DC
0272 0273 0274 0275 0276 0277 0278	C1BF 52 53 45 D4 C1BF 53 41 56 C5 C1C3 57 52 49 54 C5 C1C8 56 45 52 49 46 D9 C1CE 55 4E 4C 4F 41 C4 C1D4 44 53 4B 49 4E C5 C1DA 42 41 43 4B 55 D6	F F F 1 F 9 F	CCC 'SAV',\$88+'E' CCC 'WRIT',\$88+'E' CCC 'WRIF',\$88+'Y' CCC 'UNLOA',\$88+'D' CCC 'DSNIN',\$88+'I' CCC 'BACKU',\$88+'P'	D8 D9 DA DB DC DD
	C1BF 53 41 56 C5 C1C3 57 52 49 54 C5 C1C8 56 45 52 49 46 D9 C1CE 55 4E 4C 4F 41 C4 C1D4 44 53 4B 49 4E C9 C1DA 42 41 43 4B 55 D0 C1EØ 43 4F 50 D9			D8 D9 DA DB DC DD DD
0272 0273 0274 0275 0276 0277 0278 0279 0280 0281		F	FCC 'DSKI',\$8Ø+'\$' FCC 'DSKO',\$8Ø+'\$'	D8 D9 DA DB DC DD
Ø28Ø Ø281 Ø282	C1E4 44 53 4B 49 A4	F F	CC 'DSKI',\$8Ø+'\$'	D8 D9 DA DB DC DD DE
0280 0281 0282 0283	C1E4 44 53 4B 49 A4 C1E9 44 53 4B 4F A4	F F	FCC 'DSKI',\$8Ø+'\$' FCC 'DSKO',\$8Ø+'\$'	D8 D9 DA DB DC DD DE DF EØ
0280 0281 0282 0283 0284 0285	C1E4 44 53 4B 49 A4 C1E9 44 53 4B 4F A4	F F F * DISK BAS	FCC 'DSKI',\$8Ø+'\$' FCC 'DSKO',\$8Ø+'\$'	D8 D9 DA DB DC DD DE E E Ø
0280 0281 0282 0283 0284 0285 0286	C1E4 44 53 4B 49 A4 C1E9 44 53 4B 4F A4 C1EE 44 4F D3	F F F * DISK BAS	CCC 'DSKI',\$80+'\$' CCC 'DSKO',\$80+'\$' CCC 'DO',\$80+'\$' SIC COMMAND JUMP TABLE	D8 D9 DA DB DC DD DE E DF E Ø E 1  COMMAND / TOKEN #
0280 0281 0282 0283 0284 0285	C1E4 44 53 4B 49 A4 C1E9 44 53 4B 4F A4	* DISK BAS	CCC 'DSKI',\$80+'\$' CCC 'DSKO',\$80+'\$' CCC 'DO',\$80+'\$'	D8 D9 DA DB DC DD DE E E Ø

0294 0295 0296 0297 0298 0299 0301 0301 0302 0303 0304 0305 0306 0307	C1F5 DØ BC C1F7 D1 5C C1F9 C6 EF C1FB CA 48 C1FD D1 Ø2 C1FF CA 39 C2Ø1 DØ 1B C2Ø3 D1 Ø1 C2Ø5 C9 EØ C2Ø7 DØ 66 C2Ø9 D7 4E C2Ø8 D2 33 C2ØD D5 99 C2ØF D2 62 C211 D3 B9 C213 D4 ED C215 D5 62 C217 D6 EC	FDB FIELD FDB FILES FDB KILL FDB LOAD FDB LSET FDB MERGE FDB RENAME FDB RSET FDB WRITE FDB WRITE FDB VERIFY FDB DSKINI FDB BACKUP FDB DSKI FDB DSKI FDB DSKI FDB DSKI FDB DSKO FDB DOS	FIELD / DØ FILES / D1 KILL / D2 LOAD / D3 LSET / D4 MERGE / D5 RENAME / D6 RSET / D7 SAVE / DB WRITE / D9 VERIFY / DA UNLOAD / DB DSKINI /DC BACKUP / DD COPY / DE DSKI\$ / DF DSKO\$ / EØ DOS / E1
Ø3Ø8 Ø3Ø9		* SECONDARY FUNCTION DICTIONARY TABLE	
Ø312 Ø313 Ø314	C219 43 56 CE C21C 46 52 45 C5 C22Ø 4C 4F C3 C223 4C 4F C6 C226 4D 4B 4E A4 C22A 41 D3	* LC219 FCC 'CV', \$80+'N' FCC 'FRE', \$80+'E' FCC 'LO', \$80+'C' FCC 'LO', \$80+'F' FCC 'MKN', \$80+'S' FCC 'A', \$80+'S'	TOKEN # A2 A3 A4 A5 A6 A7
Ø318 Ø319		* DISK BASIC SECONDARY FUNCTION JUMP TABLE	
0320 0321 0322 0323 0324 0325 0326	C22C CD F4 C22E CE 9C C23Ø CE 1Ø C232 CE 37 C234 CE Ø2 C236 B2 77	* DISK BASIC SECONDARY FUNCTION JUMP TABLE  * LC22C FDB CVN FDB FREE FDB LOC FDB LOF FDB MKN\$ FDB AS	FUNCTION / TOKEN # CVN / A2 FREE / A3 LOC / A4 LOF / A5 MKN\$ / A6 AS / A7
0327 0328 0329 0330 0331 0332 0333 0334 0336 0337 0348 0349 0341 0342 0343 0344 0345 0346 0347 0348	C238 81 E1 C23A 22 Ø8 C23C 8E C1 F1 C23F 8Ø CE C241 7E AD D4 C244 81 E1 C246 18 23 EØ 2D	*DISK BASIC COMMAND INTERPRETATION HANDLER LC238 CMPA #DHITOK BHI LC244	*COMPARE TO HIGHEST DISK BASIC TOKEN *AND BRANCH IF HIGHER POINT X TO DISK BASIC COMMAND JUMP TABLE SUBTRACT OUT LOWEST DISK BASIC COMMAND TOKEN JUMP TO BASIC'S COMMAND HANDLER COMPARE TO HIGHEST DISK BASIC TOKEN 'SYMTAX' EPPOR IE ON SK BASIC TOKEN
0365 0366 0367 0368 0369 0371 0372 0373 0374 0375 0376 0377	C290 42 52 C292 44 46 C294 4F 42 C296 57 50 C298 46 4E C29A 46 53 C29C 41 45 C29C 41 45 C29E 46 4F C2AØ 53 45 C2AØ 53 45 C2AØ 54 66 C2AØ 55 52	* DISK BASIC ERROR MESSAGES LC290 FCC 'BR' FCC 'OB' FCC 'WP' FCC 'FN' FCC 'FS' FCC 'AE' FCC 'F0' FCC 'SE' FCC 'SE' FCC 'VF' FCC 'ER'	27 BAD RECORD 28 DISK FULL 29 OUT OF BUFFER SPACE 38 WRITE PROTECTED 31 BAD FILE NAME 32 BAD FILE STRUCTURE 33 FILE ALREADY EXISTS 34 FIELD OVERFLOM 35 SET TO NON-FIELDED STRING 36 VERIFICATION ERROR 37 WRITE OR INPUT PAST END OF RECORD
0379 0380 0381 0382 0383 0384	C2A6 42 41 53 C2A9 20 20 20 C2AC 44 41 54 C2AF 42 49 4E	* DISK FILE EXTENSIONS BASEXT FCC 'BAS' DEFEXT FCC ' ' DATEXT FCC 'DAT' BINEXT FCC 'BIN'	BASIC FILE EXTENSION BLANK (DEFAULT) FILE EXTENSION DATA FILE EXTENSION BINARY FILE EXTENSION

### CLS RAM VECTOR ### C282 34 11 DVEC22 PSNS X,CC ### SAVE X, REG AND STATUS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### C289 26 84 BNE LC28F ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, WITH CALLING ADDRESS ### SAVE X, REG AND X, RESTORE X, REG X, TATUS AND RETURN ### SAVE X, REG AND X, RESTORE X, RESTO	Ø  NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### ### ### ### ### ### ### ### ### ##	Ø  NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### CASES BC 97 5F CMPX #H.975F COMING FROM EXRAS' GET/PUT? ### BNE LC28F NO ### BNE LC28F NO ### NUMBER SIGN (GET#, PUT#)? ##	Ø  NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### NUMBER SIGN (GET#, PUT#)? ### NUMBER SIGN (GET#, PUT#)? ### BEQ LC2C1	Ø  NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### BEQ LC2C1	Ø  NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### RESTORE X REG, STATUS AND RETURN ### RESTORE X REG, STATUS AND REGISTERS OFF ### RESTORE X RES	Ø  NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### ### ### ### ### ### ### ### ### ##	Ø  NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### ### ### ### ### ### ### ### ### ##	Ø  NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
8396   C2C3 BD C8 2E	Ø  NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### ### ### ### ### ### ### ### ### ##	Ø NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### 838 C2C8 6F 88 15 ### 15 CLR FGBET,X # RESET THE GET ### 16399 C2C8 6F 88 16 ### 16 CLR FGBET+1,X # DATA POINTER ### 16400 C2CE 6F 88 17 ### 16400 C2CE 6F 88 17 ### 16400 C2CE 6F 88 17 ### 16400 C2CE 6F 88 18 ### 16400 C2CE 4F 6F 66 ### 16400 C2CE 4F 6F 6F 6F 6C CLR FGBPUT+1,X ### 17	Ø NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
8489	Ø NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### CAPT COUNTY FOR THE PROPRY CAPT AND THE PR	Ø NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
0402   C204 6F 06	Ø NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### ### ### ### ### ### ### ### ### ##	Ø NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
0405   C2DA 9D A5	Ø NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
0406   C2DC 27 0C	Ø NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
0407   C2DE BD B2 6D	NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
04499   C2E4   F 10	NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### ### ### ### ### ### ### ### ### ##	NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
0411   C2EB ED 07	NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
### Q413   C2EC 27 10	NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
## 414 C2EE BD C6 85 ## C2F1 EC 09 ## LDD FCBRLN,X ## GET RANDOM FILE RECORD NUMBER ## C2F3 AE 0B ## LDX FCBBUF,X ## BUFFER POINTER AND SAVE THEM ON THE ## RECORD LENGTH AND RAV ## RECORD LENGTH COUNTER AND RANDOM BUI ## RECORD LENGTH COUNTER AND RANDOM BUI ## RECORD LENGTH COUNTER AND RANDOM BUI ## POINTER WHICH ARE MAINTAINED ON THE ## POINT X TO (RECORD NUMBER -1) ## SAVE PRODUCT ON THE STACK ## POINT X TO (RECORD NUMBER -1) ## POINT X TO (RECORD NUMBER ## POINT X TO (RECORD NUMBER -1) ## POINT X TO (RECORD NUMBER -1) ## POINT X TO (RECORD NUMBER -1) ## POINT X TO (RECORD NUMBER ## THE STACT THE SECTOR DEPOINT NUMBER ## THE FIRST BYTE OF THE SECTOR USED BY ## THE FIRST BYTE OF THE SECTOR USED BY ## THE START THE BYTE IS LOCATED) ## POINT X TO (RECORD NUMBER -1) ## THE FIRST BYTE OF THE SECTOR DEPOINT NUMBER ## THE FIRST BYTE OF THE SECTOR DEPOINT NUMBER ## T	NDOM FILE STACK - MPORARY FER STACK D NUMBER -1)
## GET RANDOM FILE RECORD LENGTH AND RA ## SUFFER POINTER AND SAVE THEM ON THE ## RECORD LENGTH COUNTER AND RANDOM BUT ## POINT X TO (RECORD NUMBER -1) ## RECORD LENGTH X (RECORD MULT (UNSIGNED) RECORD MU	STACK - MPORARY FER STACK D NUMBER -1)
### ### ### ### ### ### ### ### ### ##	MPORARY FER STACK D NUMBER -1)
## RECORD LENGTH COUNTER AND RANDOM BUF ## RECORD LENGTH COUNTER AND RANDOM BUF ## POINTER WHICH ARE MAINTAINED ON THE ## POINTER WHICH ARE MAINTAINED ## POINTER WHICH ## POINTER M	FER STACK D NUMBER -1)
## POINTER WHICH ARE MAINTAINED ON THE ### POINTER WHICH ARE MAINTAINED ON THE ####################################	STACK D NUMBER -1)
0421         C2F9         BD 9F B5         JSR         L9FB5         MULT (UNSIGNED) RECORD LENGTH X (RECOF 0422 C2FC 34 60         PSHS U,Y         SAVE PRODUCT ON THE STACK           0423         C2FE A6 E0         LDA ,S+         CHECK MS BYTE OF PRODUCT           0424         C300 26 09         BNE LC30B         'BR' ERROR IF NOT ZERO (RECORD NUMBER PULS X           0425         C302 35 10         PULS X         * PULL THE BOTTOM 3 PRODUCT BYTES OFF 0426           0426         C304 35 04         PULS B         * TOP TWO IN X, BOTTOM IN ACCB; ACCB FOR USED BY THE FIRST BYTE OF THE SECTOR USED BY ACCB FOR USED BY ACC	
0422         C2FC 34 60         PSHS U,Y         SAVE PRODUCT ON THE STACK           0423         C2FE A6 E0         LDA ,S+         CHECK MS BYTE OF PRODUCT           0424         C300 26 09         BNE LC308         'BR' ERROR IF NOT ZERO (RECORD NUMBER           0425         C302 35 10         PULS X         * PULL THE BOTTOM 3 PRODUCT BYTES OFF           0426         C304 35 04         PULS B         * TOP TWO IN X, BOTTOM IN ACCB; ACCB #           0427         *         * THE FIRST BYTE OF THE SECTOR USED BY           0428         *         * (X) CONTAINS THE SECTOR OFFSET (IN W           0429         *         * FROM THE START THE BYTE IS LOCATED)           0430         C306 8C 02 64         LC306 CMPX #(TRKMAX-1)         612 SECTORS MAX IN A RANDOM FILE           0431         C309 25 05         BLO LC310         BRANCH IF RECORD LENGTH O.K.	
0423   C2FE A6 E0	T00 D10)
## ## ## ## ## ## ## ## ## ## ## ## ##	TOO DIO)
0426     C304     35     04     PULS     B     * TOP TWO IN X, BOTTOM IN ACCB; ACCB; F       0427     *     *     * THE FIRST BYTE OF THE SECTOR USED B)       0428     *     * (X) CONTAINS THE SECTOR OFFSET (IN W       0429     *     * FROM THE START THE BYTE IS LOCATED)       0430     C306     BC 02     64     LC306     CMPX     #(TRKMAX-1)     612     SECTORS MAX IN A RANDOM FILE       0431     C309     25     85     BLO     LC310     BRANCH IF RECORD LENGTH O.K.	
0427         *         * THE FIRST BYTE OF THE SECTOR USED BY 4428         *         * (X) CONTAINS THE SECTOR OFFSET (IN W 4428)         *         * (X) CONTAINS THE SECTOR OFFSET (IN W 4428)         * FROM THE START THE BYTE IS LOCATED)         * FROM THE START THE BYTE IS LOCATED) <td< td=""><td></td></td<>	
0428     *     * (X) CONTAINS THE SECTOR OFFSET (IN No. 12)       0429     *     * FROM THE START THE BYTE IS LOCATED)       0430     C306     8C 02 64     * KC306     CMPX #(TRKMAX-1)     612 SECTORS MAX IN A RANDOM FILE       0431     C309     25 05     BLO     LC310     BRANCH IF RECORD LENGTH O.K.	
0430 C306 8C 02 64 LC306 CMPX #(TRKMAX-1) 612 SECTORS MAX IN A RANDOM FILE 0431 C309 25 05 BLO LC310 BRANCH IF RECORD LENGTH O.K.	
0431 C309 25 05 BLO LC310 BRANCH IF RECORD LENGTH O.K.	
Ø432 C3ØB C6 36 LC3ØB LDB #2*27 'BAD RECORD' ERROR	
0433 C30D 7E AC 46 JMP LAC46 JUMP TO ERROR HANDLER	
Ø434         C31Ø DE F1         LC31Ø         LDU         FCBTMP         POINT U TO FCB           Ø435         C312         AC 4D         CMPX         FCBSOF,U         * COMPARE SAVED SECTOR OFFSET TO THE (	IIDDENT SECTOD OFFSET
9436 C314 10 27 00 B7 LBEQ LC3CF * BEING PROCESSED - DO NOT PROCESS A N	
0437 C318 34 14 PSHS X,B SAVE BYTE AND SECTOR OFFSET TO RECORD	
0438         C31A         A6 4F         LDA         FCBFLG,U         * CHECK FCB GET/PUT FLAG AND           0439         C31C         27 06         BEQ         LC324         * BRANCH IF IT WAS A GET	
Ø439         C31C         27 Ø6         BEQ         LC324         * BRANCH IF IT WAS A GET           Ø44Ø         C31E         6F 4F         CLR         FCBFLG,U         FORCE GET/PUT TO 'PUT'	
0441 C320 C6 03 LDB #\$03 DSKCON WRITE OP CODE	
0442 C322 8D 33 BSR LC357 GO WRITE A SECTOR - SAVE 'PUT' DATA ON	DISK
0443 * CONVERT THE SECTOR OFFSET TO A GRANULE AND SECTOR NUMBER 0444 C324 EC 61 LC324 LDD \$01,S * GET THE NUMBER OF SECTORS TO THE STA	RT OF
0445 C326 BD C7 84 JSR LC784 * THIS RECORD NUMBER AND CONVERT THEM	
0446 C329 34 04 PSHS B SAVE GRANULE OFFSET ON THE STACK	
Ø447     C32B BD C7 79     JSR LC779     MULTIPLY GRANULE NUMBER X 9 - CONVERT       Ø448     C32E 5Ø     NEGB     * NEGATE LS BYTE OF GRANULE OFFSET AND	
0449 C32F EB 63 ADDB \$03,S * LS BYTE OF SECTOR OFFSET - ACCB = SI	
Ø45Ø * * NUMBER (Ø-8) CORRESPONDING TO THE SE	CTOR NUMBER WITHIN A
Ø451** GRANULE OF THE LAST SECTOR OF THE SIØ452C331 5CINCB= ADD ONE - SECTORS SAVED IN THE FCB;	
Ø453 C332 E7 44 STB FCBSEC,U = AT 1 NOT Ø - SAVE IT IN THE FCB	
0454 C334 E6 42 LDB FCBFGR,U GET FIRST GRANULE IN FILE	
0455 C336 BD C7 55 JSR LC755 POINT X TO FAT 0456 C339 33 06 LEAU FATCON,X POINT U TO FAT DATA	
0457 C33B A6 E4 LDA ,S GET NUMBER OF GRANULES OFFSET TO RECOF	D
Ø458 C33D 4C INCA ADD ONE (COMPENSATE FOR DECA BELOW)	
0459 C33E 30 C4 LC33E LEAX ,U POINT X TO FAT DATA 0460 C340 3A ABX POINT X TO CORRECT GRANULE	
9461 C341 4A DECA DECREMENT GRANULE COUNTER	
0462 C342 27 37 BEQ LC37B BRANCH IF CORRECT GRANULE FOUND	
Ø463         C344         E7         E4         STB         ,S         SAVE GRANULE ADDRESS ON STACK           Ø464         C346         E6         84         LDB         ,X         GET NEXT GRANULE IN FILE	
0465 C348 C1 C0 CMPB #\$C0 LAST GRANULE IN FILE?	
Ø466 C34A 25 F2 BLO LC33E NO - KEEP LOOKING	
9467  * THE CRANILE RETURN SEARCHER FOR IS NOT DRESENTLY REFINED IN THIS RANDOM FILE	
0468 * THE GRANULE BEING SEARCHED FOR IS NOT PRESENTLY DEFINED IN THIS RANDOM FILE 0469 C34C E6 E4 LDB ,S GET OFFSET TO LAST GRANULE	
Ø47Ø C34E ØD D8 TST ÅD8 * CHECK GET/PUT FLAG	
0471 C350 26 14 BNE LC366 *AND BRANCH IF PUT	
0472     C352     C6     2E     LC352     LDB     #2*23     'INPUT PAST END OF FILE' ERROR       0473     C354     7E     AC     46     JMP     LAC46     JUMP TO ERROR HANDLER	
9473 C397 76 C4 9 Unit EAC40 SUFF LEAST (EGEON, U POINT X TO FCB DATA BUFFER	
0475	
0476 * READ/WRITE A SECTOR. ENTER WITH OP CODE IN ACCB, BUFFER PTR IN X 0477 C35A D7 EA LC35A STB DCOPC SAVE DSKCON OPERATION CODE VARIABLE	
9477 C396 D7 E C000 VANIABLE STX DCBPT SAVE DSKCON LOAD BUFFER VARIABLE	
Ø479 C35E 3Ø C4 LEAX ,U POINT X TO FCB	
0480 C360 BD C7 63 JSR LC763 CONVERT FCB TRACK AND SECTOR TO DSKCON	WARTARI 50

Ø481	C363 7E D6 F2		JMP	LD6F2	READ/WRITE A TRACK OR SECTOR
Ø482					
Ø483	0000 04 40			INTO A GRANULE NOT PRESENTLY INCL	
Ø484 Ø485	C366 34 12	LC366	PSHS		SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE
Ø486	C368 BD C7 BF C36B 1F 89		TFR	LC7BF	FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB
Ø487	C36D 35 42		PULS		PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK
Ø488	C36F E7 C4		STB		SAVE NEWLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE
Ø489	C371 4A		DECA	**	DECREMENT GRANULE COUNTER
0490	C372 26 F2			LC366	GET ANOTHER GRANULE IF NOT DONE
Ø491	C374 34 14		PSHS	Х,В	SAVE POINTER TO LAST GRANULE AND OFFSET
0492	C376 BD C7 1E			LC71E	WRITE FAT TO DISK
Ø493	C379 35 14		PULS	B,X	RESTORE POINTER AND OFFSET
Ø494		4 100EN		F ODANULE TO FOUND FIND THE DIGHT	T CECTOR
Ø495 Ø496	C27B 22 61	LC37B		Γ GRANULE IS FOUND, FIND THE RIGH \$01,S	
Ø497	C37B 32 61 C37D DE F1	LU3/B		FCBTMP	REMOVE GRAN NUMBER FROM STACK POINT U TO FCB
Ø498	C37F E7 43			FCBCGR,U	SAVE CURRENT GRANULE IN FCB
Ø499	C381 86 FF		LDA		*SET FCBSOF,U TO ILLEGAL SECTOR OFFSET WHICH WILL
0500	C383 A7 4D		STA	FCBSOF,U	*FORCE NEW SECTOR DATA TO BE READ IN
0501	C385 A6 84		LDA	,Х	GET CURRENT GRANULE
0502	C387 81 CØ		CMPA		IS IT THE LAST GRANULE?
0503	C389 25 27			LC3B2	NO
Ø5Ø4	C38B 84 3F		ANDA		MASK OFF LAST GRANULE FLAG BITS
Ø5Ø5 Ø5Ø6	C38D A1 44 C38F 24 21			FCBSEC,U LC3B2	* COMPARE CALCULATED SECTOR TO CURRENT SECTOR IN FCB  * AND BRANCH IF CALCULATED SECTOR IS > LAST SECTOR IN FILE
Ø5Ø7	C391 96 D8		LDA		= CHECK GET/PUT FLAG: IF 'GET' THEN 'INPUT
Ø5Ø8	C393 27 BD			LC352	= PAST END OF FILE' ERROR
Ø5Ø9	C395 A6 44			FCBSEC,U	* GET CURRENT SECTOR NUMBER FROM FCB,
0510	C397 8A CØ			#\$CØ	* OR IN THE LAST GRANULE FLAG BITS
Ø511	C399 A7 84			,Х	* AND SAVE IN FAT
Ø512	C39B BD C5 A9			LC5A9	WRITE FAT TO DISK IF NECESSARY
Ø513	C39E AE 49			FCBRLN,U	* GET RECORD LENGTH AND CHECK TO
Ø514	C3AØ 8C Ø1 ØØ			#SECLEN LC3AD	* SEE IF IT IS SECLEN (EXACTLY ONE SECTOR) BRANCH IF IT IS NOT EXACTLY ONE SECTOR
Ø515 Ø516	C3A3 26 Ø8 C3A5 AC C8 13			FCBLST, U	=BRANCH IF THE NUMBER OF BYTES IN THE LAST SECTOR
Ø510	C3A8 27 Ø8			LC3B2	=IS SET TO ONE SECTOR (SECLEN)
Ø518	C3AA 86 81			#\$81	*SET THE PRESAVED FLAG (BIT15) AND FORCE
Ø519	C3AC 21 4F	LC3AC		\$C3FD	*THE NUMBER OF BYTES IN LAST SECTOR TO 256
Ø52Ø	C3AD 4F	LC3AD	CLRA		SET THE NUMBER OF BYTES IN LAST SECTOR TO ZERO
Ø521	C3AE 5F		CLRB		CLEAR LS BYTE OF ACCD
Ø522	C3AF ED C8 13			FCBLST,U	SAVE THE NUMBER OF BYTES IN LAST SECTOR
Ø523	C3B2 C6 Ø2	LC3B2	LDB		DSKCON READ OP CODE
Ø524	C3B4 AE 49			FCBRLN,U #SECLEN	* GET RECORD LENGTH AND COMPARE
Ø525 Ø526	C3B6 8C Ø1 ØØ C3B9 26 ØD			LC3C8	* IT TO SECLEN - EXACTLY ONE SECTOR BRANCH IF NOT EXACTLY ONE SECTOR LONG
Ø527	C3BB 32 67			\$Ø7,S	CLEAN UP STACK
Ø528	C3BD AE 4B			FCBBUF,U	POINT X TO START OF RANDOM FILE BUFFER
Ø529	C3BF 96 D8			VD8	* CHECK GET/PUT FLAG AND
Ø53Ø	C3C1 27 Ø2			LC3C5	* BRANCH IF GET
Ø531	C3C3 C6 Ø3		LDB	#\$03	DSKCON WRITE OP CODE
	> C3C5 7E C3 5A	LC3C5		LC35A	READ/WRITE A SECTOR
	> C3C8 BD C3 57	LC3C8		LC357	READ A SECTOR INTO FCB DATA BUFFER
Ø534 Ø535	C3CB 35 14	*	PULS	в, х	* GET BACK THE BYTE OFFSET TO RECORD: X = NUMBER OF * SECTORS; ACCB = BYTE POINTER IN SECTOR
Ø536	C3CD AF 4D		STX	FCBSOF,U	SAVE SECTOR OFFSET IN FCB
Ø537	C3CF 34 Ø4	LC3CF	PSHS		SAVE BYTE OFFSET ON STACK
Ø538	C3D1 BD C7 55			LC755	POINT X TO FILE ALLOCATION TABLE
Ø539	C3D4 3Ø Ø6		LEAX	FATCON, X	MOVE X TO FAT DATA
Ø54Ø	C3D6 E6 43			FCBCGR,U	GET CURRENT GRANULE NUMBER
Ø541	C3D8 3A		ABX		POINT X TO PROPER GRANULE IN FAT
Ø542	C3D9 A6 84			, X	* GET CURRENT GRANULE AND CHECK TO
Ø543	C3DB 81 CØ			#\$CØ LC4ØA	* SEE IF IT IS LAST GRANULE
Ø544 Ø545	C3DD 25 2B C3DF 84 3F		ANDA		BRANCH IF THIS GRANULE IS < LAST GRANULE MASK OFF LAST GRANULE FLAG BITS
Ø546	C3E1 A1 44			FCBSEC,U	* COMPARE LAST SECTOR USED IN GRANULE TO
Ø547	C3E3 26 25			LC4ØA	* CALCULATED SECTOR; BRANCH IF NOT EQUAL
Ø548	C3E5 EC C8 13		LDD	FCBLST,U	GET NUMBER OF BYTES IN LAST SECTOR
Ø549	C3E8 84 7F		ANDA		MASK OFF PRESAVED FLAG (BIT 15)
Ø55Ø	C3EA 34 Ø6		PSHS	В,А	SAVE NUMBER OF BYTES IN LAST SECTOR ON STACK
Ø551	C3EC 4F		CLRA	\$Ø2,S	* LOAD ACCB WITH THE BYTE OFFSET TO CURRENT
Ø552 Ø553	C3ED E6 62 C3EF E3 63			\$02,5 \$03,S	* RECORD AND ADD THE REMAINING RECORD LENGTH * TO IT - ACCD = END OF RECORD OFFSET
Ø554	C3F1 10 A3 E1			,S++	=COMPARE THE END OF RECORD OFFSET TO THE NUMBER OF
Ø555	C3F4 23 14			LC4ØA	=BYTES USED IN THE LAST SECTOR
Ø556	C3F6 ØD D8		TST		* CHECK GET/PUT FLAG AND BRANCH IF 'GET'
Ø557				LC352	* TO 'INPUT PAST END OF FILE' ERROR
arro	C3F8 10 27 FF 56				
Ø558	C3F8 10 27 FF 56				
Ø559	C3F8 10 27 FF 56			O SECTOR, CALCULATE HOW MANY BYTE:	
Ø559 Ø56Ø			TA IS	BEING 'PUT' PASTH THE CURRENT END	OF FILE
Ø559 Ø56Ø Ø561	C3FC 10 83 01 00		TA IS I	BEING 'PUT' PASTH THE CURRENT END #SECLEN	OF FILE COMPARE TO ONE SECTOR'S LENGTH
Ø559 Ø56Ø Ø561 Ø562	C3FC 10 83 01 00 C400 23 03		TA IS I CMPD BLS	BEING 'PUT' PASTH THE CURRENT END #SECLEN LC405	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR
Ø559 Ø56Ø Ø561	C3FC 10 83 01 00		TA IS I CMPD BLS LDD	BEING 'PUT' PASTH THE CURRENT END #SECLEN	OF FILE COMPARE TO ONE SECTOR'S LENGTH
0559 0560 0561 0562 0563	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00	* IF DA	TA IS I CMPD BLS LDD	BEING 'PUT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH
0559 0560 0561 0562 0563 0564 0565 0566	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13	* IF DA LC405 *	TA IS I CMPD BLS LDD ORA STD	BEING 'PÙT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$80 FCBLST,U	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR
0559 0560 0561 0562 0563 0564 0565 0566	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13 C40A 35 04	* IF DA	TA IS I CMPD BLS LDD ORA STD PULS	BEING 'PÚT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$80 FCBLST,U B	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR PULL BYTE OFFSET OFF OF THE STACK
0559 0560 0561 0562 0563 0564 0565 0566 0567	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13 C40A 35 04 C40C 30 C8 19	* IF DA LC405 *	TA IS I CMPD BLS LDD ORA STD PULS LEAX	BEING 'PÙT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$80 FCBLST,U	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR PULL BYTE OFFSET OFF OF THE STACK POINT X TO FCB DATA BUFFER
0559 0560 0561 0562 0563 0564 0565 0566 0567 0568 0569	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13 C40A 35 04 C40C 30 C8 19 C40F 3A A	* IF DA LC405 *	CMPD BLS LDD ORA STD PULS LEAX ABX	BEING 'PÙT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$80 FCBLST,U B FCBCON,U	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR PULL BYTE OFFSET OFF OF THE STACK POINT X TO FCB DATA BUFFER MOVE X TO START OF RECORD
0559 0560 0561 0562 0563 0564 0565 0566 0567 0568 0569	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13 C40A 35 04 C40C 30 C8 19 C40F 3A C410 EE 62	* IF DA LC405 *	TA IS I CMPD BLS LDD ORA STD PULS LEAX ABX LDU	BEING 'PUT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$80  FCBLST,U B FCBCON,U  \$02,S	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH  * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE  * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR PULL BYTE OFFSET OFF OF THE STACK POINT X TO FCB DATA BUFFER MOVE X TO START OF RECORD POINT U TO CURRENT POSITION IN RANDOM FILE BUFFER
0559 0560 0561 0562 0563 0564 0565 0566 0567 0568 0569 0570 0571	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13 C40A 35 04 C40C 30 C8 19 C40F 3A C410 EE 62 C412 34 04	* IF DA LC405 *	TA IS I CMPD BLS LDD ORA STD PULS LEAX ABX LDU PSHS	BEING 'PÜT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$60  FCBLST,U B FCBCON,U  \$02,S B	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR PULL BYTE OFFSET OFF OF THE STACK POINT X TO FCB DATA BUFFER MOVE X TO START OF RECORD POINT U TO CURRENT POSITION IN RANDOM FILE BUFFER SAVE BYTE OFFSET ON STACK
0559 0560 0561 0562 0563 0564 0565 0566 0567 0568 0569 0570 0571	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13 C40A 35 04 C40C 30 C8 19 C40F 3A C410 EE 62	* IF DA LC405 *	TA IS I CMPD BLS LDD ORA STD PULS LEAX ABX LDU	BEING 'PÜT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$60  FCBLST,U B FCBCON,U  \$02,S B	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH  * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE  * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR PULL BYTE OFFSET OFF OF THE STACK POINT X TO FCB DATA BUFFER MOVE X TO START OF RECORD POINT U TO CURRENT POSITION IN RANDOM FILE BUFFER SAVE BYTE OFFSET ON STACK  * CONVERT ACCD INTO A NEGATIVE 2 BYTE NUMBER
0559 0560 0561 0562 0563 0564 0565 0566 0567 0568 0569 0570 0571	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13 C40A 35 04 C40C 30 C8 19 C40F 3A C410 EE 62 C412 34 04	* IF DA LC405 *	TA IS I CMPD BLS LDD ORA STD PULS LEAX ABX LDU PSHS LDA	BEING 'PÜT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$60  FCBLST,U B FCBCON,U  \$02,S B	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR PULL BYTE OFFSET OFF OF THE STACK POINT X TO FCB DATA BUFFER MOVE X TO START OF RECORD POINT U TO CURRENT POSITION IN RANDOM FILE BUFFER SAVE BYTE OFFSET ON STACK
0559 0560 0561 0562 0563 0564 0565 0566 0567 0568 0569 0570 0571 0572	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13 C40A 35 04 C40C 30 C8 19 C40F 3A C410 EE 62 C412 34 04 C414 86 FF	* IF DA LC405 *	TA IS I CMPD BLS LDD ORA STD PULS LEAX ABX LDU PSHS LDA	BEING 'PUT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$80  FCBLST,U B FCBCON,U  \$02,S B #-1	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH  * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE  * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR PULL BYTE OFFSET OFF OF THE STACK POINT X TO FCB DATA BUFFER MOVE X TO START OF RECORD POINT U TO CURRENT POSITION IN RANDOM FILE BUFFER SAVE BYTE OFFSET ON STACK  * CONVERT ACCD INTO A NEGATIVE 2 BYTE NUMBER  * REPRESENTING THE REMAINING UNUSED BYTES IN THE SECTOR
0559 0560 0561 0562 0563 0564 0565 0566 0566 0567 0570 0571 0572 0573	C3FC 10 83 01 00 C400 23 03 C402 CC 01 00 C405 8A 80 C407 ED C8 13 C40A 35 04 C40C 30 C8 19 C40F 3A C410 EE 62 C412 34 04 C414 86 FF	* IF DA LC405 *	TA IS I CMPD BLS LDD ORA STD PULS LEAX ABX LDU PSHS LDA ADDD	BEING 'PUT' PASTH THE CURRENT END #SECLEN LC405 #SECLEN #\$80  FCBLST,U B FCBCON,U  \$02,S B #-1	OF FILE COMPARE TO ONE SECTOR'S LENGTH BRANCH IF REMAINDER OF RECORD LENGTH WILL FIT IN THIS SECTOR FORCE NUMBER OF BYTES = ONE SECTOR LENGTH  * SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE  * WRITTEN TO DISK BEFORE LEAVING 'PUT' SAVE NUMBER OF BYTES USED IN LAST SECTOR PULL BYTE OFFSET OFF OF THE STACK POINT X TO FCB DATA BUFFER MOVE X TO START OF RECORD POINT U TO CURRENT POSITION IN RANDOM FILE BUFFER SAVE BYTE OFFSET ON STACK  * CONVERT ACCO INTO A NEGATIVE 2 BYTE NUMBER  * REPRESENTING THE REMAINING UNUSED BYTES IN THE SECTOR  * ADD TEMPORARY RECORD LENGTH COUNTER (SUBTRACT

Ø577 Ø578	C41A ED 61 C41C 35 Ø4		STD PULS	\$01,S B	SAVE NEW TEMPORARY RECORD LENGTH COUNTER RESTORE BYTE COUNTER
Ø579 Ø58Ø	C41E 50	*	NEGB		* NEGATE IT - ACCB = THE NUMBER OF BYTES * AVAILABLE TO A RECORD IN THIS SECTOR
Ø581 Ø582	C41F 2Ø Ø8		BRA	LC429	MOVE THE DATA
Ø583 Ø584				IF REMAINING RECORD LENGTH WILL FOR THE CURRENTLY SELECTED SECTOR	IT IN
Ø585	C421 E6 62	LC421	LDB	\$Ø2,S	GET REMAINING RECORD LENGTH
Ø586 Ø587	C423 6F 61 C425 6F 62			\$01,S \$02,S	* CLEAR THE TEMPORARY RECORD LENGTH * COUNTER ON THE STACK
Ø588	C427 32 61		LEAS	\$Ø1,S	PURGE BYTE OFFSET FROM STACK
	C429 96 D8 C42B 27 Ø2	LC429		VD8 LC42F	* CHECK GET/PUT FLAG AND * BRANCH IF GET
	C42D 1E 13 C42F BD A5 9A	LC42F		X,U LA59A	SWAP SOURCE AND DESTINATION POINTERS TRANSFER DATA FROM SOURCE TO DESTINATION BUFFERS
Ø593	C432 EF 62	20421		\$02,5	SAVE NEW TEMP RECORD POINTER ON THE STACK (GET)
Ø594 Ø595				ROM FCB DATA BUFFER TO THE RANDOM	
Ø596 Ø597	C434 DE F1	* OR FRO		OOM FILE BUFFER TO FCB DATA BUFFER FCBTMP	IF 'PUT' POINT U TO FCB
	C436 96 D8			VD8	* CHECK GET/PUT FLAG AND
	C438 27 Ø4 C43A A7 4F			LC43E FCBFLG,U	* BRANCH IF GET SAVE 'PUT' FLAG IN THE FCB
	C43C AF 62 C43E AE 4D	LC43E		\$02,S FCBSOF,U	SAVE NEW TEMPORARY RECORD POINTER ON STACK (PUT) * GET SECTOR OFFSET COUNTER AND
0603	C440 30 01	20.02	LEAX	\$Ø1,X	* ADD ONE TO IT
	C442 5F C443 EE E4		CLRB LDU	,\$	SET BYTE OFFSET = Ø * CHECK THE LENGTH OF THE TEMPORARY RECORD LENGTH
Ø6Ø6 Ø6Ø7	C445 10 26 FE BD C449 35 96				* COUNTER AND KEEP MOVING DATA IF ←> Ø * PULL TEMPORARY RECORD LENGTH AND
0608	0449 35 90	*	FULS	<b>А,</b> Б, <b>А,</b> F С	* BUFFER ADDRESS OFF STACK AND RETURN
Ø6Ø9 Ø61Ø		* OPEN R	АМ НО	nk	
Ø611	C44B 32 62	DVECØ		\$02,S	PULL RETURN ADDRESS OFF OF THE STACK
	C44D BD B1 56 C45Ø BD B6 A4			LB156 LB6A4	EVALUATE AN EXPRESSION *GET MODE(I,O,R) - FIRST BYTE OF STRING EXPRESSION
	C453 34 Ø4 C455 BD A5 A2		PSHS JSR	B LA5A2	*AND SAVE IT ON STACK GET DEVICE NUMBER
Ø616	C458 5D		TSTB		SET FLAGS
	C459 10 2F E1 A6 C45D 35 02		LBLE PULS	LA6Ø3 A	BRANCH IF NOT A DISK FILE GET MODE
Ø619	C45F 34 Ø6 C461 ØF 6F		PSHS	B,A DEVNUM	SAVE MODE AND DEVICE NUMBER (FILE NUMBER) SET DEVICE NUMBER TO SCREEN
	C463 BD B2 6D			SYNCOMMA	SYNTAX CHECK FOR COMMA
	C466 8E C2 AC C469 BD C9 38			#DATEXT LC938	POINT TO 'DAT' FOR EXTENSION GET FILENAME FROM BASIC
Ø624	C46C CC Ø1 FF		LDD	#\$01FF	DEFAULT DISK FILE TYPE AND ASCII FLAG
	C46F FD Ø9 57 C472 8E Ø1 ØØ			DFLTYP #SECLEN	SAVE DEFAULT VALUES: DATA, ASCII DEFAULT RECORD LENGTH - 1 PAGE
	C475 9D A5 C477 27 Ø8			GETCCH LC481	GET CHAR FROM BASIC BRANCH IF END OF LINE
Ø629	C479 BD B2 6D		JSR	SYNCOMMA	SYNTAX CHECK FOR COMMA
	C47C BD B3 E6 C47F 9E 52			LB3E6 FPAØ+2	EVALUATE EXPRESSION GET EVALUATED EXPRESSION
Ø632	C481 BF Ø9 7C	LC481	STX	DFFLEN LB44A	RECORD LENGTH
Ø634	C484 10 27 EF C2 C488 BD A5 C7			LA5C7	IF = Ø, THEN 'ILLEGAL FUNCTION CALL' ERROR IF ANY FURTHER CHARACTERS ON LINE
Ø635 Ø636	C48B 35 Ø6		PULS	A,B	GET MODE AND FILE NUMBER
Ø637	C40D 24 G0			ILE FOR READ OR WRITE	CAVE MODE ON CTACK
Ø638 Ø639	C48D 34 Ø2 C48F BD C7 49	LC48D	PSHS JSR	LC749	SAVE MODE ON STACK POINT X TO FCB FOR THIS FILE
Ø64Ø Ø641	C492 10 26 E1 86 C496 9F F1			LA61C FCBTMP	'FILE ALREADY OPEN' ERROR IF FILE OPEN SAVE FILE BUFFER POINTER
Ø642	C498 BD C7 9D		JSR	LC79D	MAKE SURE FILE ALLOC TABLE IS VALID
	C49B BD C6 8C C49E 35 Ø4		PULS	LC68C B	SCAN DIRECTORY FOR 'FILENAME.EXT' GET MODE
	C4AØ 86 1Ø C4A2 34 Ø2		LDA PSHS	#INPFIL	INPUT TYPE FILE SAVE FILE TYPE ON STACK
Ø647	C4A4 C1 49		CMPB	#'I'	INPUT MODE?
Ø648 Ø649	C4A6 26 1F		BNE	LC4C7	BRANCH IF NOT
Ø65Ø	CAAO DD C6 FF	* OPEN A		ENTIAL FILE FOR INPUT	CHECK TO SEE IE DIDECTORY MATCH IS FOUND
	C4A8 BD C6 E5 C4AB BD C8 Ø7			LC6E5 LC8Ø7	CHECK TO SEE IF DIRECTORY MATCH IS FOUND CHECK TO SEE IF FILE ALREADY OPEN
	C4AE BE Ø9 74 C4B1 EC ØB			V974 DIRTYP,X	GET RAM DIRECTORY BUFFER GET FILE TYPE AND ASCII FLAG
Ø655	C4B3 FD Ø9 57		STD	DFLTYP	SAVE IN RAM IMAGE
	C4B6 8D 75 C4B8 BD C6 27			LC52D LC627	INITIALIZE FILE BUFFER CONTROL BLOCK GO FILL DATA BUFFER
	C4BB BD C7 55 C4BE 6C ØØ	LC4BB		LC755 FATØ,X	POINT X TO PROPER FILE ALLOCATION TABLE ADD ONE TO FAT ACTIVE FILE COUNTER
0660	C4CØ 9E F1		LDX	FCBTMP	GET FILE BUFFER POINTER
	C4C2 35 Ø2 C4C4 A7 ØØ		PULS STA	A FCBTYP,X	GET FILE TYPE SAVE IT IN FCB
Ø663	C4C6 39	1.0407	RTS		
	C4C7 68 E4 C4C9 C1 4F	LC4C7	ASL CMPB	#'0'	SET FILE TYPE TO OUTPUT FILE MODE = OUTPUT?
Ø666 Ø667	C4CB 26 1B		BNE	LC4E8	BRANCH IF NOT
Ø668	0400 70 70 77	* OPEN A		ENTIAL FILE FOR OUTPUT	DOES EVER EVEST ON DESCRIPTION
	C4CD 7D Ø9 73 C4DØ 27 ØF		TST BEQ	V973 LC4E1	DOES FILE EXIST ON DIRECTORY? BRANCH IF NOT
Ø671	C4D2 BD C6 FC C4D5 B6 Ø9 73		JSR	LC6CF V973	KILL THE OLD FILE  * GET DIRECTORY SECTOR NUMBER OF OLD FILE AND
שט/ ב	כ/ לש טם טעדט		LUA	V3/3	AET DIVECTOR! SECTOR MOMBER OL OFD LIFE WAD

C4D8 B7 C4DB BE				V977 V974	* SAVE IT AS FIRST FREE DIRECTORY ENTRY =GET RAM DIRECTORY IMAGE OF OLD FILE AND
C4DE BF				V978	=SAVE IT AS FIRST FREE DIRECTORY ENTRY
C4E1 BD		LC4E1		LC567	SET UP NEW DIRECTORY ENTRY ON DISK
C4E4 8D C4E6 20				LC538 LC4BB	INITIALIZE FILE BUFFER FLAG AND MAP FCB AS BEING USED
C4E8 C1 C4EA 27		LC4E8		#'R'	FILE MODE = R (RANDOM)?
C4EA 27 C4EC C1				LC4F2 #'D'	BRANCH IF SO FILE MODE = D (DIRECT)?
C4EE 10	Ø 26 E1	24	LBNE	LA616	'BAD FILE MODE' ERROR IF NOT
		* OPEN	A RAND	OM/DIRECT FILE	
C4F2 68 C4F4 FC		LC4F2	ASL	,S RNBFAD	SET FILE TYPE TO DIRECT * GET ADDRESS OF RANDOM FILE BUFFER AREA
C4F7 34				B,A	* AND SAVE IT ON THE STACK
C4F9 F3 C4FC 25				DFFLEN LC5Ø4	ADD THE RECORD LENGTH 'OB' ERROR IF SUM > \$FFFF
C4FE 10	Ø B3 Ø9		CMPD	FCBADR	IS IT > THAN FCB DATA AREA?
C5Ø2 23 C5Ø4 C6		LC5Ø4		LC5Ø9 #2*29	BRANCH IF NOT 'OUT OF BUFFER SPACE' ERROR
C5Ø6 7E	E AC 46		JMP	LAC46	JUMP TO ERROR HANDLER
C5Ø9 34 C5ØB 7D		LC5Ø9		B,A V973	SAVE END OF RANDOM BUFFER ON STACK DID THIS FILE EXIST
C5ØE 26	5 Ø4		BNE	LC514	BRANCH IF SO
C51Ø 8D C512 2Ø				LC567 LC519	SET UP NEW FILE IN DIRECTORY INITIALIZE FCB
C514 86	5 FF	LC514	LDA	#\$FF	* SET FILE TYPE MATCH = \$FF (ILLEGAL VALUE) -
C516 BD	C8 Ø7	*	JSR	LC807	* THIS WILL FORCE ANY OPEN MATCHED FILE TO CAUSE  * A 'FILE ALREADY OPEN' ERROR
C519 8D		LC519			INITIALIZE FCB
C51B 63	3 ØD	*	COM	FCBSOF,X	* SET FCBSOF,X TO \$FF (ILLEGAL SECTOR OFFSET) WHICH N * FORCE NEW SECTOR DATA TO BE READ IN DURING GET/PUT
C51D 60				FCBREC+1,X	INITIALIZE RECORD NUMBER = 1
C51F 35 C521 FD					U = START OF RANDOM FILE BUFFER AREA, ACCD = END SAVE NEW START OF RANDOM FILE BUFFER AREA
C524 EF	ØB		STU	FCBBUF, X	SAVE BUFFER START IN FCB
C526 FE C529 EF				DFFLEN FCBRLN,X	* GET RANDOM FILE RECORD LENGTH * AND SAVE IT IN FCB
C52B 20				LC4BB	SET FAT FLAG, SAVE FILE TYPE IN FCB
		* INIT	TAL T7F	FCB DATA FOR INPUT	
C52D 8D		LC52D	BSR	LC538	INITIALIZE FCB
C52F FE C532 EE				V974 DIRLIST,U	GET RAM DIRECTORY IMAGE *GET NUMBER OF BYTES IN LAST SECTOR OF FILE
C534 EF	88 13		STU	FCBLST,X	*SAVE IT IN FCB
C537 39	9	* INIT	RTS TALTZE	FILE CONTROL BLOCK	
C538 9E			LDX	FCBTMP	GET CURRENT FILE BUFFER
C53A C6		LC53C	CLR	#FCBCON .X+	CLEAR FCB CONTROL BYTES CLEAR A BYTE
C53E 5A	Ą		DECB		DECREMENT COUNTER
C53F 26 C541 9E				LC53C FCBTMP	BRANCH IF NOT DONE GET CURRENT FILE BUFFER ADDRESS BACK
C543 96				DCDRV	*GET CURRENT DRIVE NUMBER AND
C545 A7 C547 B6				FCBDRV,X V976	*SAVE IT IN FCB =GET FIRST GRANULE -
C54A A7 C54C A7				FCBFGR, X	=SAVE IT AS THE STARTING GRANULE NUMBER AND =SAVE IT AS CURRENT GRANULE NUMBER
C54E F6				FCBCGR, X V973	GET DIRECTORY SECTOR NUMBER
C551 CØ C553 58			SUBB ASLB		SUBTRACT 3 - DIRECTORY SECTORS START AT 3 * MULTIPLY SECTORS
C554 58			ASLB		* BY 8 (8 DIRECTORY
C555 58			ASLB PSHS	R	* ENTRIES PER SECTOR) SAVE SECTOR OFFSET
C558 FC	09 74		LDD	V974	GET RAM DIRECTORY IMAGE
C55B 83				#DBUFØ #\$Ø8	SUBTRACT RAM OFFSET 8 DIRECTORY ENTRIES/SECTOR
C560 3D	)		MUL		NOW ACCA CONTAINS Ø-7
C561 AB C563 A7				,S+ FCBDIR,X	ACCA CONTAINS DIRECTORY ENTRY (0-71) SAVE DIRECTORY ENTRY NUMBER
C566 39			RTS	· -··•,··	
		* SET	UP DIRF	CTORY AND UPDATE FILE ALLOCATION	TABLE ENTRY IN FIRST UNUSED SECTOR
C567 C6		LC567	LDB	#28*2	'DISK FULL' ERROR
C569 B6	5 Ø9 77 Ø 27 E6	D6	LDA LBEQ	V977 LAC46	GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRIES
C57Ø B7	7 Ø9 73		STA	V973	SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY
C573 97 C575 C6				DSEC #\$Ø2	SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE
C577 D7				DCOPC	SAVE IN DSKCON REGISTER READ SECTOR
C579 BD C57C BE				LD6F2 V978	* GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY
C57F BF				V974	* ENTRY AND SAVE AS CURRENT USED RAM IMAGE
C582 33 C584 C6			LEAU LDB	,X #DIRLEN	(TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY)
	- 8Ø	LC586	CLR	,X+	CLEAR BYTE
C586 6F			DECB BNE	LC586	DECREMENT COUNTER CONTINUE IF NOT DONE
C588 5A C589 26			LDX	#DNAMBF	POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION
C588 5A C589 26 C58B 8E	Ø9 4C				
C588 5A C589 26	Ø9 4C ØB		LDB JSR	#11 LA59A	MOVE B BYTES FROM X TO U
C588 5A C589 26 C58B 8E C58E C6 C59Ø BD C593 FC	09 4C 0 0B 0 A5 9A 0 09 57		JSR LDD	LA59A DFLTYP	MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG
C588 5A C589 26 C58B 8E C58E C6 C59Ø BD	09 4C 0 0B 0 A5 9A 0 09 57 0 40		JSR LDD	LA59A DFLTYP \$00,U	MOVE B BYTES FROM X TO U

	C59D B7 Ø9 76		STA	V976	SAVE IN RAM
	C5AØ A7 42	10510	STA	\$Ø2,U	SAVE IN RAM IMAGE OF DIRECTORY TRACK
Ø771	C5A2 C6 Ø3		LDB	#\$Ø3	* GET WRITE OPERATION CODE AND SAVE
0//2 0773	C5A4 D7 EA		.1CD	1 D6F2	* IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY
0774	L5A9 34 56	LC5A9	PSHS	U.X.B.A	SAVE REGISTERS
Ø775	C5AB BD C7 55	200/13	JSR	LC755	POINT X TO FILE ALLOCATION TABLE
	C5AE 6C Ø1		INC	FAT1,X	INDICATE NEW DATA IN FILE ALLOC TABLE
	C5BØ A6 Ø1		LDA	FAT1,X	GET NEW DATA FLAG
	C5B2 B1 Ø9 7A		CMPA	WFATVL	* HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO
Ø779	C5B5 25 Ø3 C5B7 BD C7 1E C5BA 35 D6	*	DIO	LCEDA	* CAUSE THE FAT TO BE WRITTEN TO THE DISK
Ø78Ø Ø781	C5B5 25 W3		.1CD	LC3BA LC31F	RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK
Ø782	C5BA 35 D6	LC5BA	PULS	A.B.X.U.PC	RESTORE REGISTERS
Ø783	00011 00 00	2005/1	. 020	,,,,,,,,	RESTORE REGISTERS
Ø784				RAM VECTOR	
Ø785	C5BC 96 6F C5BE 10 2F C7 2F C5C2 32 62 C5C4 34 14 C5C6 0F 70 C5C8 8E 09 26	DVEC4	LDA	DEVNUM	GET DEVICE NUMBER
Ø786	C5BE 10 2F C7 2F		LBLE	XVEC4	BRANCH IF NOT DISK FILE
0/8/	C5C4 34 14	1.0504	LEA5	\$02,5	GET RID OF RETURN ADDRESS SAVE REGISTERS
Ø789	C5C4 34 14 C5C6 0F 70	LC3C4	CLB	CINREL	
Ø79Ø	C5C8 8E Ø9 26		LDX	#FCBV1-2	POINT TO FILE BUFFER VECTOR TABLE
Ø791	C5CB D6 6F		LDB	DEVNUM	GET ACTIVE DISK FILE NUMBER
Ø792	C5C8 8E Ø9 26 C5CB D6 6F C5CD 58		ASLB		CLEAR BUFFER NOT EMPTY FLAG POINT TO FILE BUFFER VECTOR TABLE GET ACTIVE DISK FILE NUMBER TIMES 2 - TWO BYTES PER FCB ADDRESS NOW Y POINTS TO ELIF BUFFER
Ø793	C5CE AE 85 C5DØ E6 84 C5D2 C1 4Ø		LDX	D, N	NOW A TOTAL TO THEE BOTTER
0/94	C5D0 E6 84		LDB	FCBIYPE,X	GET FILE TYPE
0795 0796	C5D2 CI 40		CWLR	#RANFIL LC5EC	IS THIS A RANDOM (DIRECT) FILE? BRANCH IF NOT
Ø796 Ø797	C5D4 26 16		DNE	LUSEU	DRANCH IF NUI
~7.00		* GET A	BYTE	FROM A RANDOM FILE - RETURN CHAR I	IN ACCA
0799	C5D6 EC 88 15		LDD	FROM A KANDOM FILE - REIURN CHAR I FCEGET,X FCBGET,X FCBGET,X FCBBUF,X D,X \$\$-1,X B,X,PC EDOM A SEQUENTIAL FILE	GET THE RECORD COUNTER
0800	C5D9 10 A3 09		CMPD	FCBRLN,X	*COMPARE TO RECORD LENGTH AND
0801	C5DC 24 2Ø		BHS	LC5FE	*BRANCH TO BUFFER EMPTY IF >= RECORD LENGTH
Ø8Ø2	C5DE C3 ØØ Ø1		ADDD	#\$0001 FCDOFF V	= ADD ONE TO RECORD POINTER AND
0803 0001	CEE4 VE WD		210	FUBLITY	* POINT X TO START OF RANDOM FILE BUFFER AND
Ø8Ø5	C5E4 AE 0B		LDA	n X	* ADD THE RECORD COUNTER TO IT
Ø8Ø6	C5E8 A6 1F		LDA	\$-1.X	GET A CHARACTER FROM THE BUFFER
Ø8Ø7	C5EA 35 94		PULS	B,X,PC	RESTORE REGISTERS AND RETURN
0808		* GET A	BYTE	B,X,PC FROM A SEQUENTIAL FILE FCBCFL,X LC5F9	
0809	C5EC E6 88 1Ø	LC5EC	LDB	FCBCFL,X	* TEST THE CACHE FLAG AND BRANCH IF AN
0810	C5EF 2/ Ø8		BEQ	LC5F9 FCBCDT,X FCBCFL,X R, Y, DC	* EXTRA CHARACTER HAS NOT BEEN READ FROM FILE
0811 0812	C5F1 A6 88 11 C5F4 6F 88 10		CLD	FCBCEL Y	GET THE CACHE CHARACTER CLEAR THE CACHE FLAG
Ø813	C5F7 35 94		PIIIS	B,X,PC	RESTORE REGISTERS AND RETURN
Ø814					RESTORE REGISTERS THIS RETORN
Ø815	C5F9 E6 88 17 C5FC 27 Ø4	LC5F9	LDB	FCBDFL,X	IS ANY DATA LEFT?
Ø816	C5FC 27 Ø4		BEQ	LC602	BRANCH IF SO
Ø817	C5FE Ø3 7Ø C6ØØ 35 94		COM	CINBFL	SET FLAG TO BUFFER EMPTY
				B,X,PC	RESTORE REGISTERS AND RETURN
Ø82Ø	C602 E6 05 C604 6C 05 C606 6A 88 18	LC602	LDB	FCBCPT.X	GET CHARACTER POINTER
Ø821	C6Ø4 6C Ø5		INC	FCBCPT, X	ADD ONE TO CHARACTER POINTER
Ø822	C6Ø6 6A 88 18 C6Ø9 27 Ø6		DEC	FCBLFT,X	DECREMENT NUMBER OF CHARACTERS LEFT IN FILE BUFFER
Ø823	C6Ø9 27 Ø6		BEQ	LC611	IF LAST CHARACTER, GO GET SOME MORE
Ø824	C6ØB 3A		ABX	FORCON V	ADD CHARACTER COUNTER TO X
Ø825	CEME 3E 04		DIII S	R Y DC	GET DATA CHARACTER (SKIP PAST 25 FCB CONTROL BYTES
Ø827	CODI 33 94	* GET A	CHARA	CTER FROM FCB DATA BUFFER - RETURN	CHAR IN ACCA
Ø828	C611 34 6Ø	10611	DCIIC	II V	SAVE REGISTERS
Ø829	C613 4F		РЭПЭ		
0830		20011	CLRA	0,1	*
0831	C614 33 8B	20011	CLRA	D, X	* * POINT U TO CORRECT CHARACTER
MU 27	C614 33 8B C616 A6 C8 19			CLC611  B,X,PC CTER FROM FCB DATA BUFFER - RETURN U,Y  D,X FCBCON,U	
NX33	C614 33 8B C616 A6 C8 19 C619 34 Ø2 C61B 6F Ø5		PSHS	A	=AND SAVE DATA CHARACTER ON STACK
Ø834	C619 34 Ø2 C61B 6F Ø5 C61D A6 Ø1		PSHS CLR	A FCBCPT,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER
Ø834	C619 34 Ø2 C61B 6F Ø5 C61D A6 Ø1		PSHS CLR	A FCBCPT,X	=AND SAVE DATA CHARACTER ON STACK
Ø834	C619 34 Ø2 C61B 6F Ø5 C61D A6 Ø1		PSHS CLR LDA STA BSR	A FCBCPT,X FCBDRV,X DCDRV LC627	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER
Ø834	C619 34 Ø2 C61B 6F Ø5 C61D A6 Ø1		PSHS CLR LDA STA BSR PULS	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER
Ø834 Ø835 Ø836 Ø837 Ø838	C619 34 02 C61B 6F 05 C61D A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94	+ 055111	PSHS CLR LDA STA BSR PULS	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER
Ø834 Ø835 Ø836 Ø837 Ø838	C619 34 02 C61B 6F 05 C61D A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94	+ 055111	PSHS CLR LDA STA BSR PULS PULS THE	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN (AL FILES
Ø834 Ø835 Ø836 Ø837 Ø838	C619 34 02 C61B 6F 05 C61D A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94	+ 055111	PSHS CLR LDA STA BSR PULS PULS THE	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER
Ø834 Ø835 Ø836 Ø837 Ø838 Ø839 Ø84Ø Ø841	C619 34 02 C61B 6F 05 C61D A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94 C627 A6 04 C629 4C C62A 34 02 C62C 81 09	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843	C619 34 Ø2 C618 6F Ø5 C610 A6 Ø1 C61F 97 E8 C621 8D Ø4 C623 35 62 C625 35 94  C627 A6 Ø4 C629 4C C62A 34 Ø2 C62C 81 Ø9 C62E 23 Ø1	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843	C619 34 Ø2 C618 6F Ø5 C610 A6 Ø1 C61F 97 E8 C621 8D Ø4 C623 35 62 C625 35 94  C627 A6 Ø4 C629 4C C62A 34 Ø2 C62C 81 Ø9 C62E 23 Ø1	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843	C619 34 Ø2 C618 6F Ø5 C610 A6 Ø1 C61F 97 E8 C621 8D Ø4 C623 35 62 C625 35 94  C627 A6 Ø4 C629 4C C62A 34 Ø2 C62C 81 Ø9 C62E 23 Ø1	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0847	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 8D 07 55	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0847	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 8D 07 55	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0847	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 8D 07 55	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0847	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 8D 07 55	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0847	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 8D 07 55	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0847	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 8D 07 55	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0847	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 8D 07 55	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0847	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 8D 07 55	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0836 0837 0838 0841 0842 0843 0844 0845 0846 0847 0848 0853 0853 0854 0857	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 BD C7 55 C638 A6 C638 E6 06 C630 30 C4 C63F C1 C0 C641 24 0A C643 35 02 C645 80 0A C647 26 15 C649 E7 03 C648 20 DC	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0837 0838 0837 0840 0841 0842 0843 0844 0845 0846 0851 0852 0853 0854 0855 0857 0858	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C626 81 09 C626 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 BD C7 55 C63A 3A C648 20 02 C641 24 0A C643 35 02 C645 80 0A C647 26 15 C649 E7 03 C648 20 DC C648 C648 20 DC C640 C4 3F	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0851 0851 0852 0853 0854 0857 0858	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C626 81 09 C626 23 81 09 C626 23 81 09 C626 23 81 09 C627 A6 04 C630 AF C630 AF C631 A7 04 C633 AF C633 BC 07 55 C63A 3A C64B 80 0A C641 24 0A C643 35 02 C645 80 0A C641 26 0A C647 26 15 C649 E7 03 C648 20 DC C640 C4 3F C64F C1 09	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0836 0837 0838 0839 0840 0841 0842 0843 0844 0845 0846 0851 0851 0852 0853 0854 0857 0858	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 62 C625 35 94  C627 A6 04 C629 4C C626 81 09 C626 23 81 09 C626 23 81 09 C626 23 81 09 C627 A6 04 C630 AF C630 AF C631 A7 04 C633 AF C633 BC 07 55 C63A 3A C64B 80 0A C641 24 0A C643 35 02 C645 80 0A C641 26 0A C647 26 15 C649 E7 03 C648 20 DC C640 C4 3F C64F C1 09	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT,X FCBDRV,X DCDRV LC627 A,Y,U B,X,PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC,X	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK
0834 0835 0837 0838 0837 0840 0841 0842 0843 0844 0845 0846 0851 0852 0853 0854 0855 0856 0857 0858 0857 0858 0859	C619 34 02 C618 6F 05 C610 A6 01 C61F 97 EB C621 8D 04 C623 35 52 C625 35 94  C627 A6 04 C629 4C C62A 34 02 C62C 81 09 C62E 23 01 C630 4F C631 A7 04 C633 E6 03 C635 33 84 C637 BD C7 55 C63A 3A C638 E6 06 C630 30 C4 C63F C1 C0 C641 24 0A C643 35 02 C645 80 0A C647 26 15 C649 E7 03 C648 20 DC C649 C647 C1 09	* REFILL LC627 LC629	PSHS CLR LDA STA BSR PULS THE LDA INCA	A FCBCPT, X FCCBDRV, X DCDRV LC627 A, Y, U B, X, PC FCB INPUT DATA BUFFER FOR SEQUENTI FCBSEC, X A #\$09 LC631 FCBSEC, X FCBCGR, X , X LC755 FATCON, X , U #\$C0 LC640 A #10 LC65E FCBCGR, X LC629 #\$37 #\$39 #\$39	=AND SAVE DATA CHARACTER ON STACK RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR - FILL THE BUFFER RESTORE REGISTERS AND DATA CHARACTER RESTORE REGISTERS AND RETURN IAL FILES GET CURRENT SECTOR NUMBER ADD ONE SAVE NEW SECTOR NUMBER ON THE STACK

Ø865	C658 EØ EØ	LC658	SUBB	. S+	SUBTRACT CURRENT SECTOR NUMBER AND PULS A
Ø866	C65A 25 21			LC67D	BRANCH IF PAST LAST SECTOR
Ø867	C65C 1F 98		TFR	B,A	SECTOR NUMBER TO ACCA
Ø868	C65E 34 Ø2	LC65E	PSHS		SAVE SECTOR NUMBER DIFFERENCE
Ø869	C660 8D 23			LC685	INCREMENT RECORD NUMBER
0870	C662 86 Ø2			#\$02	*GET READ OPERATION CODE
Ø871 Ø872	C664 97 EA			DCOPC	*AND SAVE IT IN DSKCON VARIABLE GET PROPER TRACK AND SECTOR TO DSKCON VARIABLES
Ø873	C666 BD C7 63 C669 33 88 19			LC763 FCBCON,X	* POINT U TO START OF FCB DATA BUFFER
Ø874	C66C DF EE			DCBPT	* AND SAVE IT IN DSKCON VARIABLE
Ø875	C66E BD D6 F2			LD6F2	GO READ A SECTOR INTO FCB BUFFER
Ø876	C671 6F 88 18			FCBLFT,X	NUMBER OF CHARS LEFT IN BUFFER = 256
Ø877	C674 E6 EØ		LDB	,S+	GET SECTOR NUMBER OFF STACK
Ø878	C676 26 ØC			LC684	RETURN IF DATA LEFT; FALL THRU IF LAST SECTOR
Ø879	C678 EC 88 13			FCBLST,X	GET NUMBER OF BYTES IN THE LAST SECTOR
Ø88Ø Ø881	C67B 26 Ø4 C67D 5F	LC67D	CLRB	LC681	BRANCH IF SOME BYTES IN LAST SECTOR SET NUMBER OF REMAINING BYTES = 256
Ø882	C67E 63 88 17	LOUYD		FCBDFL,X	SET DATA LEFT FLAG TO \$FF
Ø883	C681 E7 88 18	LC681		FCBLFT,X	SAVE THE NUMBER OF CHARS LEFT IN BUFFER
Ø884	C684 39	LC684	RTS		
Ø885					
Ø886	C685 EE Ø7	LC685		FCBREC,X	GET CURRENT RECORD NUMBER
Ø887 Ø888	C687 33 41 C689 EF Ø7			\$Ø1,U FCBREC,X	BUMP IT PUT IT BACK
Ø889	C68B 39		RTS	robrec, x	FUI II DACK
Ø89Ø	0002 03				
Ø891				DRY FOR FILENAME.EXT FOUND IN DNAM	
Ø892				SECTOR NUMBER IN V973, GRANULE IN	
Ø893				DIRECTORY DATA IN V974. IF DISK IS	
Ø894 Ø895	C68C 7F Ø9 73	* V9// =		HE FIRST UNUSED SECTOR RETURNED IN V973	
Ø895 Ø896	C68F 7F Ø9 77	20000		V973 V977	CLEAR SECTOR NUMBER CLEAR TEMP SECTOR COUNTER
Ø897	C692 CC 11 Ø2			#\$1102	TRACK 17 (DIRECTORY), READ OPERATION CODE
	C695 97 EC			DCTRK	SAVE TRACK NUMBER
Ø899	C697 D7 EA			DCOPC	SAVE OPERATION CODE (READ)
0900	C699 C6 Ø3			#\$Ø3	READ SECTOR 3 (FIRST DIRECTORY SECTOR)
Ø9Ø1	C69B D7 ED C69D CE Ø6 ØØ	LC69B		DSEC	SAVE SECTOR NUMBER IN DSKCON VARIABLE
Ø9Ø2 Ø9Ø3	C6AØ DF EE			#DBUFØ DCBPT	*BUFFER AREA NUMBER Ø AS DATA BUFFER - SAVE *IN DSKCON VARIABLE
	C6A2 BD D6 F2			LD6F2	GO READ A SECTOR
	C6A5 FF Ø9 74	LC6A5	STU		SAVE RAM DIRECTORY BUFFER ADDRESS
Ø9Ø6	C6A8 31 C4		LEAY	,U	POINT Y TO DIRECTORY BUFFER
0907	C6AA A6 C4			, U	GET A BYTE FROM BUFFER
Ø9Ø8 Ø9Ø9	C6AC 26 28 C6AE 8D 29			LC6D6 LC6D9	BRANCH IF NOT ZERO - FILE IS ACTIVE SET UNUSED FILE POINTERS IF ENTRY HAS BEEN KILLED
Ø91Ø		LC6BØ		#DNAMBF	POINT TO DISK FILE NAME BUFFER
Ø911	C6B3 A6 8Ø	LC6B3	LDA	,X+	*COMPARE THE FILENAME AND EXTENSION
Ø912	C6B5 A1 CØ		CMPA		*STORED IN RAM AT DNAMBF TO THE DIRECTORY
Ø913	C6B7 26 ØE			LC6C7	*ENTRY STORED AT ,U (BRANCH IF MISMATCH)
0914	C6B9 8C Ø9 57			#DNAMBF+11	AT END OF FILE NAME BUFFER?
Ø915 Ø916	C6BC 26 F5 C6BE F7 Ø9 73			LC6B3 V973	BRANCH IF NOT DONE CHECKING FILENAME SAVE SECTOR NUMBER IN DSKCON VARIABLE
Ø917	C6C1 A6 42			FCBFGR,U	*GET NUMBER OF FIRST GRANULE IN FILE
Ø918	C6C3 B7 Ø9 76			V976	*AND SAVE IT IN V976
Ø919	C6C6 39		RTS		
Ø92Ø					
Ø921 Ø922	C6C7 33 A8 20	LC6C7		DIRLEN,Y	GET NEXT DIRECTORY ENTRY (DIRLEN BYTES PER ENTRY)
Ø922	C6CA 11 83 07 00 C6CE 26 D5			#DBUFØ+SECLEN LC6A5	AT END OF BUFFER? CHECK NEXT ENTRY IF NOT AT END
Ø924	C6DØ 5C		INCB	2007.0	NEXT SECTOR
Ø925	C6D1 C1 ØB		CMPB	#11	11 SECTORS MAX IN DIRECTORY
Ø926	C6D3 23 C6			LC69B	BRANCH IF MORE SECTORS
Ø927	C6D5 39		RTS		
Ø928 Ø929	C6D6 43	LC6D6	COMA		COMPLEMENT FIRST BYTE IN DIRECTORY EMTRY
Ø929 Ø93Ø	C6D7 26 D7	20000		LC6BØ	BRANCH IF FILE IS ACTIVE - FALL THRU IF NOT USED
Ø931	555. 25 5.		5.1.2	20055	The state of the s
0932				FOR FIRST UNUSED DIRECTORY ENTRY	
Ø933	C6D9 B6 Ø9 77	LC6D9	LDA		UNUSED ENTRY ALREADY FOUND?
	C6DC 26 Ø6 C6DE F7 Ø9 77		BNE STB	DVEC12	RETURN IF UNUSED ENTRY ALREADY FOUND SECTOR CONTAINING THIS DIRECTORY ENTRY
	C6E1 FF Ø9 78		STU		POINTS TO RAM AREA WHERE DIRECTORY DATA IS STORED
	C6E4 39	DVEC12		V376	FOINTS TO RAM AREA WHERE DIRECTORY DATA 15 STORED
Ø938		J. L. J. L.	5		
0939	C6E5 C6 34	LC6E4		#2*26	'NE' ERROR
	C6E7 7D Ø9 73		TST		WAS A DIRECTORY MATCH FOUND?
0941	C6EA 26 F8			DVEC12	RETURN IF FOUND
Ø942 Ø943	C6EC 7E AC 46		JMP	LAC46	JUMP TO ERROR HANDLER IF NOT FOUND
Ø944		* KILL C	OMMAND		
Ø945	C6EF BD C9 35	KILL	JSR	LC935	GET FILENAME.EXT FROM BASIC
Ø946	C6F2 BD A5 C7		JSR	LA5C7	'SYNTAX' ERROR IF MORE CHARACTERS ON LINE
	C6F5 BD C7 9D		JSR	LC79D	GET VALID FAT DATA
	C6F8 8D 92 C6FA 8D E9		R2L R2K	LUBBU	TEST FOR FILE NAME MATCH IN DIRECTORY MAKE SURE THE FILE EXISTED
	C6FC 86 FF	LC6FC	LDA	#\$FF	* MATCH FILE TYPE = \$FF; THIS WILL CAUSE AN 'AO'
Ø951		*	LUM		* ERROR TO BE GENERATED IF ANY FILE TYPE IS OPEN
0952	C6FE BD C8 Ø7		JSR	LC807	CHECK TO MAKE SURE FILE IS NOT OPEN
	C701 BE 09 74		LDX	V974	*GET RAM IMAGE OF DIRECTORY
	C7Ø4 6F 84 C7Ø6 C6 Ø3		CLR	DIKNAM, X	*AND ZERO FIRST BYTE - KILL FILE
	C708 D7 EA		STB	DCOPC	=WRITE OPERATION CODE - SAVE =IT IN DSKCON VARIABLE
	C7ØA BD D6 F2		JSR	LD6F2	WRITE A SECTOR
Ø958	C7ØD E6 ØD		LDB	DIRGRN,X	GET NUMBER OF FIRST GRANULE IN FILE
	C70F 8D 44	LC7ØF	BSR	LC755	POINT X TO PROPER FILE ALLOCATION TABLE
Ø96Ø	C711 3Ø Ø6		LEAX	LCG95 LA5C7 LC79D LC68C LC6E5 #\$FF LC807 V974 DIRNAM,X #\$03 DCOPC LD6F2 DIRGRN,X LC755 FATCON,X	SKIP 6 CONTROL BYTES

```
C713 3A
C714 E6 84
                                                                                                           POINT TO CORRECT ENTRY GET NEXT GRANULE
Ø961
Ø962
                                                     LDB
Ø963
         C716 86 FF
C718 A7 84
                                                     LDA
STA
                                                            #$FF
                                                                                                           *GET FREE GRANULE FLAG AND
*MARK GRANULE AS FREE
Ø964
                                                                                                           WAS THIS THE LAST GRANULE?
* KEEP FREEING GRANULES IF NOT LAST ONE
Ø965
         C71A C1 CØ
                                                     CMPR
                                                            #$CØ
Ø966
         C71C 25 F1
                                                            LC7ØF
                                                     BLO
                                                                                                           * WRITE FILE ALLOCATION SECTOR TO DIRECTORY - DO NOT WRITE
* THE SIX CONTROL BYTES AT THE START OF THE FAT TO THE DISK
Ø967
Ø968
                                                                                                          POINT U TO DISK BUFFER Ø AND
=SAVE IT AS DSKCON VARIABLE

* WRITE DIRECTORY TRACK - SAVE

* TRACK AND WRITE OPERATION CODE IN
                                                            #DBUFØ
Ø969
          C71E CE Ø6 ØØ
                                         LC71E
                                                     LDU
Ø97Ø
         C721 DF FF
                                                     STII
                                                            DCRPT
          C723 CC 11 Ø3
Ø971
                                                             #$1103
0972
         C726 97 EC
                                                     STA
                                                             DCTRK
         C728 D7 EA
                                                     STB
                                                             DCOPC
                                                                                                           * DSKCON VARIABLES
Ø973
Ø974
         C72A C6 Ø2
C72C D7 ED
                                                     LDB
                                                             #$Ø2
                                                                                                          = GET FILE ALLOCATION SECTOR AND
= SAVE IN DSKCON VARIABLE
Ø975
                                                             DSEC
                                                     STB
         C72E 8D 25
C73Ø 6F Ø1
                                                             LC755
FAT1,X
                                                                                                           POINT X TO PROPER FILE ALLOCATION TABLE RESET FLAG INDICATING VALID FAT DATA HAS BEEN STORED ON DISK
Ø976
                                                     BSR
Ø977
                                                     CLR
                                                                                                           MOVE (X) TO START OF GRANULE DATA
68 BYTES IN FAT
Ø978
          C732 30 06
                                                     LEAX
                                                            FATCON,X
         C734 C6 44
                                                             #GRANMX
Ø979
                                                     LDB
Ø98Ø
                                                                                                           MOVE ACCB BYTES FROM FAT RAM IMAGE TO DBUFØ
Ø981
Ø982
                                         * ZERO OUT ALL OF THE BYTES IN THE FAT SECTOR WHICH DO NOT CONTAIN THE GRANULE DATA
Ø983
         C739 6F CØ
                                         LC739
                                                     CLR ,U+
CMPU #DBUFØ+SECLEN
                                                                                                          CLEAR A BYTE FINISHED THE WHOLE SECTOR?
         C73B 11 83 Ø7 ØØ
Ø984
                                                     BNE
JMP
Ø985
         C73F 26 F8
                                                            LC739
                                                                                                           WRITE A SECTOR
         C741 7E D6 F2
                                                            LD6F2
Ø986
Ø987
                                         * ENTER WITH ACCB CONTAINING FILE NUMBER (1-15); EXIT WITH X POINTING * TO CORRECT FILE BUFFER; FLAGS SET ACCORDING TO FILE TYPE.
0988
Ø989
0990
         C744 34 Ø4
C746 D6 6F
                                         LC744
                                                     PSHS B
                                                                                                          SAVE FILE NUMBER ON STACK
GET DEVICE NUMBER (FILE NUMBER)
Ø991
Ø992
                                                     LDB DEVNUM
          C748 8C
                                                     CMPX
Ø993
                                                            #$3404
                                                                                                           SAVE FILE NUMBER ON STACK
Ø994
         C749 34 Ø4
                                         I C719
                                                     PSHS R
         C74B 58
                                                                                                           X2: 2 BYTES PER POINTER
Ø995
                                                     ASLB
Ø996
         C74C 8E Ø9 26
                                                     LDX
                                                            #FCBV1-2
                                                                                                           POINT X TO START OF FCB POINTERS
         C74F AE 85
                                                                                                           POINT X TO PROPER FCB
Ø997
                                                     LDX
                                                            B.X
Ø998
         C751 E6 ØØ
                                                     LDB
                                                            FĆBTYP,X
                                                                                                           SET FLAGS ACCORDING TO FILE TYPE
                                                                                                           RESTORE FILE NUMBER
Ø999
         C753 35 84
                                                     PULS B.PC
1000
                                         * POINT X TO DRIVE ALLOCATION TABLE
1001
1002
         C755 34 Ø6
                                         LC755
                                                     PSHS B.A
                                                                                                           SAVE ACCD ON STACK
1003
         C757 96 EB
                                                             DCDRV
1004
                                                     LDA
                                                                                                          GET LENGTH OF FILE ALLOCATION TABLE
MULTIPLY BY DRIVE NUMBER TO GET OFFSET
START OF FILE ALLOCATION TABLE
POINT TO RIGHT TABLE
1005
         C759 C6 4A
                                                     LDB
                                                             #FATLEN
1006
         C75B 3D
                                                     MUL
         C75C 8E Ø8 ØØ
C75F 3Ø 8B
1007
                                                     LDX
                                                            #FATBLØ
                                                     LEAX D,X
1008
1009
         C761 35 86
                                                     PULS A,B,PC
                                                                                                           RESTORE ACCD
1010
                                         * CONVERT GRANULE NUMBER TO TRACK & SECTOR NUMBER - X MUST BE POINTING TO CORRECT
* FCB; THE TRACK AND SECTOR NUMBER WILL BE STORED IN DSKCON REGISTERS
LC763 LDB FCBCGR,X GET GRANULE NUMBER
LSRB DIVIDE BY 2 - 2 GRANULES / TRACK
1012
         C763 E6 Ø3
C765 54
1013
1014
1015
         C766 D7 EC
                                                     STB
                                                                                                           TRACK NUMBER
                                                                                                          TRACK 17 = DIRECTORY TRACK
BRANCH IF < DIRECTORY TRACK
INCR TRACK NUMBER IF > DIRECTORY TRACK
MULTIPLY TRACK NUMBER BY 2
1016
         C768 C1 11
                                                     CMPR #17
         C76A 25 Ø2
                                                            LC76E
1017
                                                     BLO
         C76C ØC EC
1018
                                                     TNC
                                                            DCTRK
         C76E 58
                                         LC76E
1019
                                                     ASLB
1020
         C76F 5Ø
                                                     NEGB
                                                                                                           NEGATE GRANULE NUMBER
B=Ø IF EVEN GRANULE; 1 IF ODD
         C77Ø EB Ø3
                                                     ADDB FCBCGR.X
1021
          C772 8D Ø5
                                                                                                           RETURN B=Ø FOR EVEN GRANULE NUMBER, B=9 FOR ODD GRANULE NUMBER
1022
1023
         C774 EB Ø4
                                                     ADDB FCBSEC.X
                                                                                                           ADD SECTOR NUMBER
1024
         C776 D7 ED
                                                     STB
                                                                                                           SAVE SECTOR NUMBER
1025
         C778 39
                                                     RTS
1026
                                          MULTIPLY ACCD BY 9
         C779 34 Ø6
                                                                                                           TEMP STORE ACCD ON STACK
1027
                                         LC778
                                                     PSHS B,A
1028
         C77B 58
                                                     ASLB
1029
         C77C 49
                                                     ROLA
                                                                                                           * MULTIPLY BY 2
         C77D 58
1030
                                                     ASLB
1031
          C77E 49
                                                     ROLA
                                                                                                           = MULTIPLY BY FOUR
1032
         C77F 58
                                                     ASLB
1033
          C78Ø 49
                                                                                                           * MULTIPLY BY EIGHT
                                                                                                           ADD ONE = MULTIPLY BY NINE
         C781 E3 E1
                                                            ,S++
1034
                                                     ADDD
1035
         C783 39
1036
1037
                                           CONVERT ACCD INTO A GRANULE NUMBER - RETURN RESULT IN ACCB;
                                           ENTER WITH ACCD CONTAINING A NUMBER OF SECTORS. RETURN IN ACCB THE NUMBER (\emptyset-67) CORRESPONDING TO THE NUMBER OF COMPLETE
1038
1039
1040
                                         * GRANULES CONTAINED IN THAT MANY SECTORS.
* DIVIDE BY 90, MULTIPLY BY 10 IS FASTER THAN DIVIDE BY 9
1041
1042
          C784 6F F2
                                         LC784
                                                     CLR
                                                           ,-S
,S
                                                                                                          CLEAR A TEMPORARY SLOT ON THE STACK
* DIVIDE ACCD BY 90 - SAVE THE
         C786 6C E4
                                         LC756
1043
1044
          C788 83 ØØ 5A
                                                     SUBD #9*10
                                                                                                           * QUOTIENT+1 ON THE STACK - REMAINDER
                                                            LC786
1045
         C78B 2A F9
                                                     BPL
                                                                                                           * IN ACCB
1046
          C78D A6 E4
                                                                                                           = PUT THE QUOTIENT+1 IN ACCA AND
                                                             ,S
                                                                                                           = SAVE REMAINDER ON STACK
1047
         C78F F7 F4
                                                     STR
                                                                                                           * MULTIPLY (QUOTIENT+1)
1048
         C791 C6 ØA
                                                     LDB
                                                            #10
                                                                                                           * BY 10
PUT THE REMAINDER IN ACCA
1049
         C793 3D
                                                     MIII
1050
         C794 35 Ø2
                                                     PULS A
                                                                                                          * DECREMENT THE GRANULE COUNT BY ONE FOR

* EVERY NINE SECTORS (1 GRANULE) IN THE

* REMAINDER - COMPENSATE FOR THE + 1 IN QUOTIENT+1

CLEAR MS BYTE OF ACCD
1051
         C796 5A
C797 8B Ø9
                                         LC796
                                                     DECR
                                                            #$Ø9
1052
                                                     ADDA
1053
         C799 2B FB
                                                     BMI
                                                             LC796
1054
         C79B 4F
                                                     CLRA
1055
         C79C 39
                                         LC79C
                                                     RTS
1056
```

1057	0700 00 00			AM FILE ALLOCATION TABLE DATA IS V	
1058 1059	C79D 8D B6 C79F 6D ØØ	LC79D		LC755 FATØ,X	POINT X TO FAT FOR THE CORRECT DRIVE NUMBER CHECK TO SEE IF ANY FILES ARE ACTIVE
1060	C7A1 26 F9		BNE	LC79C	RETURN IF ANY FILES ACTIVE IN THIS FAT
1061 1062	C7A3 6F Ø1 C7A5 33 Ø6			FAT1,X FATCON,X	RESET FAT DATA VALID FLAG LOAD U WITH START OF GRANULE DATA BUFFER
1063	C7A7 8E Ø6 ØØ		LDX	#DBUFØ	BUFFER FOR DISK TRANSFER
1064 1065	C7AA 9F EE C7AC CC 11 Ø2		STX LDD	DCBPT #\$1102	PUT IN DSKCON PARAMETER DIRECTORY TRACK, READ SECTOR
1066	C7AF 97 EC		STA	DCTRK	STORE IN DSKCON TRACK NUMBER
1067 1068	C7B1 D7 EA C7B3 C6 Ø2			DCOPC #\$Ø2	STORE IN DSKCON OP CODE GET SECTOR NUMBER 2 (FILE ALLOCATION TABLE)
1069	C7B5 D7 ED			DSEC	STORE IN DSKCON PARAMETER
1070 1071	C7B7 BD D6 F2 C7BA C6 44		JSR LDB	LD6F2 #GRANMX	GO READ SECTOR TRANSFER FILE ALLOCATION TABLE TO FILE ALLOC TABLE BUFFER
1071	C7BC 7E A5 9A			LA59A	MOVE B BYTES FROM (X) TO (U)
1073		+ FIND F	TIDET I	EDEE CDANUUE ENTED WITH ACCD CON-	TAINING
1074 1075				FREE GRANULE – ENTER WITH ACCB CON M WHICH TO START SEARCHING. THE FO	
1076				Y STORING A \$CØ IN THE GRANULE'S D	
1077 1078				THAT IT IS THE LAST GRANULE IN TH FIRST FREE GRANULE FOUND IN ACCA	t filt.
1079	C7BF 8D 94	LC7BF	BSR	LC755	POINT X TO FILE ALLOC TABLE
1080 1081	C7C1 3Ø Ø6 C7C3 4F		CLRA	FATCON, X	SKIP CONTROL BYTES USE ACCA AS GRANULE COUNTER
1082	C7C4 C4 FE		ANDB		MASK OFF BIT ZERO OF SEARCH GRANULE
1083 1084	C7C6 6F E2 C7C8 63 85	LC7C8		, - S B, X	INITIALIZE AND SAVE A BYTE ON STACK (DIRECTION FLAG) IS THIS GRANULE FREE? (\$FF=FREE)
1085	C7CA 27 31	20700	BEQ	LC7FD	BRANCH IF IT IS
1086 1087	C7CC 63 85 C7CE 4C		COM INCA	В,Х	RESTORE GRANULE DATA ADD ONE TO GRANULE COUNTER
1088	C7CF 81 44			#GRANMX	GRANMX GEANULES PER DISK
1089 1090	C7D1 24 25 C7D3 5C		BHS INCB	LC7F8	BRANCH IF ALL GRANULES CHECKED (DISK FULL) INCR TO NEXT GRANULE
1090	C7D4 C5 Ø1		BITB	#\$01	IS BIT Ø SET?
1092	C7D6 26 FØ			LC7C8	BRANCH IF ODD GRANULE NUMBER (SAME TRACK)
1093 1094	C7D8 34 Ø6 C7DA CØ Ø2		PSHS SUBB		SAVE GRANULE COUNTER AND CURRENT GRANULE NUMBER SUBTRACT ONE TRACK (2 GRANULES)
1095	C7DC 63 62		COM	\$Ø2,S	COMPLEMENT DIRECTION FLAG
1096 1097	C7DE 26 ØC C7EØ EØ EØ		BNE SUBB	LC7EC S+	BRANCH EVERY OTHER TIME SUBTRACT THE GRANULE COUNTER FROM THE CURRENT GRANULE NUMBER
1098	C7E2 2A Ø4			LC7E8	BRANCH IF LOWER BOUND NOT EXCEEDED
1099	C7E4 E6 E4	10756	LDB	,\$	RESTORE CURRENT GRANULE NUMBER IF LOWER BOUND EXCEEDED
1100 1101	C7E6 63 61	LC7E6 *	СОМ	\$01,S	* COMPLEMENT FLAG - IF GRANULE NUMBER HAS EXCEEDED  * BOUNDS ON EITHER THE HI OR LO SIDE, FORCE IT TO GO IN
1102		*			* THE DIRECTION OPPOSITE THE EXCEEDED BOUND
1103 1104	C7E8 32 61 C7EA 20 DC	LC7E8		\$01,S LC7C8	CLEAN UP STACK CHECK FOR ANOTHER FREE GRANULE
1105					
1106 1107	C7EC EB EØ C7EE C1 44	LC7EC	ADDB	,S+ #GRANMX	ADD THE GRANULE COUNTER TO THE CURRENT GRANULE NUMBER GRANMX GRANULES PER DISK
1107	C7FØ 25 F6			LC7E8	BRANCH IF UPPER BOUND NOT EXCEEDED
1109	C7F2 E6 E4		LDB SUBB	, S	* RESTORE CURRENT GRANULE COUNT AND GO TWICE
1110 1111	C7F4 CØ Ø4 C7F6 2Ø EE			LC7E6	* AS FAR AS USUAL IN OPPOSITE DIRECTION IF UPPER BOUND EXCEEDED KEEP SEARCHING
1112	C7F8 C6 38	LC7F8	LDB	#2*28	'DISK FULL' ERROR
1113 1114	C7FA 7E AC 46		JMP	LAC46	JUMP TO ERROR HANDLER
1115				FIRST FREE GRANULE POSITION IN THE	
1116 1117	C7FD 32 61	* TABLE LC7FD		ARK THE POSITION WITH A LAST GRANU \$01,S	LE IN FILE MARKER CLEAR UP STACK - REMOVE DIRECTION FLAG
1118	C7FF 1F 98	LOTTE	TFR	B,A	GRANULE NUMBER TO ACCA
1119	C8Ø1 3A C8Ø2 C6 CØ		ABX	неса	POINT X TO FIRST FOUND GRANULE
1120 1121	C8Ø4 E7 84		LDB STB	#\$CØ ,X	LAST GRANULE FLAG MARK THE FIRST FOUND GRANULE AS THE LAST GRANULE
1122	C8Ø6 39	LC8Ø6	RTS		
1123 1124		* CHECK	ALL A	CTIVE FILES TO MAKE SURE A FILE IS	NOT ALREADY OPEN - TO BE OPEN
1125		* A FILE	E BUFFE	ER MUST MATCH THE DRIVE NUMBER AND	FIRST GRANULE NUMBER
1126 1127				CTORY ENTRY AND THE FCB TYPE MUST OR WILL NOT BE GENERATED IF A FILE	
1128				DE THAT IT HAS ALREADY BEEN OPENED	
1129 1130	C807 34 02	LC8Ø7	PSHS	A	SAVE FILE TYPE ON STACK
1131	C8Ø9 F6 Ø9 5B	20007	LDB	FCBACT	NUMBER OF CURRENTLY OPEN FILES
1132 1133	C8ØC 5C C8ØD BD C7 49	LC8ØD	INCB	LC749	ADD ONE MORE TO FILE COUNTER POINT X TO FCB OF THIS FILE
1133	C810 27 17	FCOND		LC749 LC829	BRANCH IF BUFFER NOT BEING USED
1135	C812 96 EB		LDA	DCDRV	* GET DRIVE NUMBER AND CHECK TO SEE IF IT
1136 1137	C814 A1 Ø1 C816 26 11			FCBDRV,X LC829	* MATCHES THE DRIVE NUMBER FOR THIS BUFFER FILE EXISTS ON ANOTHER DRIVE
1138	C818 FE Ø9 74		LDU	V974	GET RAM DIRECTORY AREA
1139 1140	C81B A6 4D C81D A1 Ø2			DIRGRAN,U FCBFGR,X	GET FIRST GRANULE IN FILE DOES IT MATCH THIS FILE BUFFER?
1140	C81F 26 Ø8			LC829	NO
1142	C821 A6 ØØ			FCBTYP,X	GET FILE TYPE OF THIS BUFFER
1143 1144	C823 A1 E4 C825 10 26 DD F3		CMPA LBNE	,S LA61C	DOES IT MATCH THE ONE WE ARE LOOKING FOR? 'FILE ALREADY OPEN' ERROR IF NOT
1145	C829 5A	LC829	DECB		DECR FILE COUNTER
1146 1147	C82A 26 E1 C82C 35 82		BNE PULS	LC8ØD A PC	BRANCH IF HAVEN'T CHECKED ALL ACTIVE FILES RESTORE FILE TYPE AND RETURN
1148					
1149 1150	C82E BD A5 A5 C831 ØF 6F	LC82E		LA5A5 DEVNUM	EVALUATE AN EXPRESSION (DEVICE NUMBER) SET DEVICE NUMBER TO SCREEN
1151	C833 5D		TSTB		TEST NEW DEVICE NUMBER
1152	C834 10 2F EC 12		LBLE	LB44A	'FC' ERROR IF DEVICE NUMBER NOT A DISK FILE

1153 1154	C838 BD C7 49 C83B A6 ØØ		JSR LDA	LC749 FCBTYP,X LA3FB	POINT X TO FCB TEST IF BUFFER IS IN USE
1155	C83D 10 27 DB BA		LBEQ	LA3FB	'FILE NOT OPEN' ERROR
1156 1157	C841 81 40		CMPA	#RANFIL	DIRECT/RANDOM FILE?
1157	C843 27 C1 C845 7E A6 16	LC856		LC8Ø6 LA616	RETURN IF RANDOM BAD FILE MODE ERROR IF NOT RANDOM
1159					
1160	C848 86 10			E NUMBER CHECK RAM HOOK #INPFIL	INPUT FILE TYPE
1162	C84A 8C	LC84A		#8620	SKIP TWO BYTES
1163			DEUTO		
1164 1165	C84B 86 2Ø			E NUMBER CHECK RAM HOOK #OUTFIL	OUTPUT FILE TYPE
1166	C84D ØD 6F		TST	DEVNUM	* CHECK DEVICE NUMBER AND RETURN IF
1167 1168	C84F 2F B5 C851 AF E4		BLE STX	LC806	* NOT A DISK FILE = REPLACE SUBROUTINE RETURN ADDRESS WITH X REGISTER -
1169		*			= THIS IS THE SAME AS LEAS 2,S AND PSHS X
	C853 BD C7 44		JSR	LC744	POINT X TO FCB
	C856 34 Ø6 C858 A6 ØØ		LDA	B,A FCBTYP.X	SAVE ACCB AND FILE TYPE ON STACK GET FILE TYPE
1173	C85A 10 27 DB 9D		LBEQ	LA3FB	'FILE NOT OPEN' ERROR
	C85E 81 4Ø C86Ø 27 Ø6		CMPA	#RANFIL	RANDOM FILE? BRANCH IF RANDOM FILE
	C862 A1 E4		CMPA	,S	IS THIS FCB OF THE PROPER TYPE?
	C864 26 DF	10055	BNE	LC845	'FILE MODE' ERROR IF NOT
1178	C866 35 96	LC866	PULS	LC744 B,A FCBTYP,X LA3FB #RANFIL LC868 ,S LC845 A,B,X,PC	RESTORE ACCB,X,ACCA (FILE TYPE) AND RETURN
1180	C868 AE 64	LC868	LDX		* GET CALLING ADDRESS FROM THE STACK AND
	C86A 8C BØ ØC C86D 26 F7		CMPX	#LB00C	* RETURN UNLESS COMING FROM * BASIC'S 'INPUT' STATEMENT
1183	C86F BD B2 6D		JSR	SYNCOMMA	SYNTAX CHECK FOR A COMMA
1184	C872 81 22		CMPA	#'"'	CHECK FOR A DOUBLE QUOTE
1185	C874 26 ØB C876 BD B2 44		JSR	LB244	RETURN TO BASIC'S 'INPUT' COMMAND STRIP PROMPT STRING FROM BASIC AND PUT IT ON THE STRING STACK
1187	C876 BD B2 44 C879 BD B6 57	LC881	JSR	LB657	PURGE THE STRING PUT ON THE STRING STACK
1188	C87C C6 3B C87E BD B2 6F		LDB	#';'	SEMICOLON DO A SYNTAX CHECK FOR SEMICOLON
1190	C881 8E BØ 1E	LC881	LDX	#LBØ1E	GET MODIFIED REENTRY POINT INTO BASIC
1191	C004 AT 04		21X	\$04,5	AND PUT IT INTO THE RETURN ADDRESS ON THE STACK
1192 1193	C886 35 96		PULS	A,B,X,PC	RETURN TO BASIC
1194				ER VALIDITY CHECK RAM HOOK	
	C888 2F 25 C88A F1 Ø9 5B	DVEC1		LC8AF FCBACT	RETURN IF NOT A DISK FILE COMPARE DEVICE NUMBER TO HIGHEST POSSIBLE
1197	C88D 10 22 DD 8E			LA61F	'DEVICE NUMBER' ERROR IF TOO BIG
1198	C891 35 9Ø		PULS	X,PC	RETURN
1199 1200		* SET PE	RINT P	ARAMETERS RAM HOOK	
	C893 ØD 6F	DVEC2			*CHECK DEVICE NUMBER AND
	C895 2F 18 C897 32 62		BLE	LC8AF	*RETURN IF NOT DISK FILE PURGE RETURN ADDRESS OFF OF THE STACK
	C899 34 16		PSHS	\$02,S X,B,A	SAVE REGISTERS
1205 1206	C89B ØF 6E C89D BD C7 44			PRTDEV	SET PRINT DEVICE NUMBER TO NON-CASSETTE
	C8AØ E6 Ø6		LDB	LC744 FCBPOS,X	POINT X TO FCB GET PRINT POSITION
1208	C8A2 4F		CLRA		PRINT WIDTH (256)
	C8A3 8E 10 00 C8A6 7E A3 7C			#\$1000 LA37C	TAB FIELD WIDTH AND TAB ZONE SAVE THE PRINT PARAMETERS
1211					
1212	C8A9 ØD 6F	* BREAK DVEC11		RAM HOOK	* CHECK DEVICE NUMBER AND RETURN
1214	C8AB 2F Ø2	DVLCII		LC8AF	* IF NOT A DISK FILE
	C8AD 32 62	1.0045		\$Ø2,S	= PURGE RETURN ADDRESS OFF OF THE STACK - DON'T
1216 1217	C8AF 39	LC8AF	K13		= DO A BREAK CHECK IF DISK FILE
1218	COD# 20 CC			ERPRETATION RAM HOOK	DUDGE DETUDN ADDRESS OFF OF THE STACK
1219 1220	C8BØ 32 62 C8B2 1C AF	DVEC2Ø LC8B2			PURGE RETURN ADDRESS OFF OF THE STACK ENABLE IRQ & FIRQ
1221	C8B4 7F FF Ø2		CLR	PIAØ+2	STROBE ALL KEYS (COLUMN STROBE)
1222 1223	C8B7 B6 FF ØØ C8BA 43		LDA COMA		READ KEYBOARD ROWS INVERT KEYBOARD ROW DATA
	C8BB 84 7F			#\$7F	MASK OFF JOYSTICK INPUT BIT
	C8BD 27 Ø3			LC8C2	BRANCH IF NO KEY DOWN
	C8BF BD AD EB C8C2 9E A6	LC8C2		LADEB CHARAD	GO DO A BREAK CHECK IF A KEY IS DOWN GET INPUT POINTER INTO X
1228	C8C4 9F 2F		STX	TINPTR	TEMP SAVE IT
	C8C6 A6 8Ø C8C8 27 Ø7		LDA	,X+ LC8D1	SEARCH FOR THE END OF CURRENT LINE BRANCH IF END OF LINE
	C8CA 81 3A			#':'	CHECK FOR END OF SUB LINE, TOO
	C8CC 27 25			LC8F3	BRANCH IF END OF SUB LINE
	C8CE 7E B2 77 C8D1 A6 81	LC8D1		LB277 ,X++	'SYNTAX' ERROR IF NOT END OF LINE *GET MS BYTE OF ADDRESS OF NEXT BASIC LINE
1235	C8D3 97 ØØ		STA	ENDFLAG	*AND SAVE IT IN CURLIN
	C8D5 26 Ø3 C8D7 7E AE 15			LC8DA LAE15	BRANCH IF NOT END OF PROGRAM GO 'STOP' THE SYSTEM
1238	C8DA EC 8Ø	LC8DA	LDD	, X+	*GET LINE NUMBER OF THIS LINE AND
	C8DC DD 68			CURLIN	*SAVE IT IN CURLIN
	C8DE 9F A6 C8EØ 96 AF			CHARAD TRCFLG	RESET BASIC'S INPUT POINTER * CHECK THE TRACE FLAG AND
1242	C8E2 27 ØF		BEQ	LC8F3	* BRANCH IF TRACE OFF
	C8E4 86 5B C8E6 BD A2 82			#'<' LA282	< LEFT DELIMITER OF TRON SEND CHARACTER TO CONSOLE OUT
1245	C8E9 96 68		LDA	CURLIN	GET NUMBER OF CURRENT LINE NUMBER
	C8EB BD BD CC C8EE 86 5D			LBDCC #'>'	CONVERT ACCD TO DECIMAL & PRINT IT ON SCREEN > RIGHT DELIMITER OF TRON
				LA282	SEND A CHARACTER TO CONSOLE OUT

1249				
	C8F3 9D 9F	LC8F3 J:	SR GETNCH	GET NEXT CHARACTER FROM BASIC
1250	C8F5 1F A9		FR CC,B	SAVE STATUS REGISTER IN ACCB
1251	C8F7 81 98		MPA #\$98	CSAVE TOKEN?
			NE LC8FE	NO
1252	C8F9 26 Ø3			***
1253	C8FB 7E 83 16		MP L8316	GO CHECK FOR CSAVEM
1254	C8FE 81 97	LC8FE CI	MPA #\$97	CLOAD TOKEN?
1255	C900 26 03	BI	NE LC9Ø5	NO NO
1256	C9Ø2 7E 83 11	JI	MP L8311	JUMP TO EXBAS' CLOAD ROUTINE
1257	C9Ø5 1F 9A		FR B,CC	RESTORE STATUS REGISTER
				LOOP THROUGH BASIC'S MAIN INTERPRETATION LOOP
1258	C907 BD AD C6		SR LADC6	LUUP IHKUUGH BASIC'S MAIN INTERPRETATION LUUP
1259	C9ØA 2Ø A6	BI	RA LC8B2	
1260				
1261		* EOF RAM	H00K	
1262	C9ØC 32 62	DVEC14 LI	FAS \$02.S	PURGE RETURN ADDRESS OFF OF THE STACK
1263	C9ØE 96 6F		DA DEVNUM	* GET DEVICE NUMBER AND SAVE
1264			SHS A	* IT ON THE STACK
1265			SR LA5AE	STRIP DEVICE NUMBER OFF OF INPUT LINE
1266			SR LA3ED	VERIFY THAT THE FILE TYPE WAS 'INPUT'
1267	C918 ØD 6F	T:	ST DEVNUM	* CHECK DEVICE NUMBER AND
1268	C91A 10 2F DC BC	LI	BLE LA5DA	* BRANCH BACK TO BASIC'S EOF IF NOT DISK FILE
1269	C91E BD C7 44	J:	SR LC744	POINT X TO FCB
	C921 E6 ØØ	11	SLE LASDA SR LC744 DB FCBTYP,X MPB #RANFIL	GET FILE TYPE
	C923 C1 4Ø	CI	MDR #DANETI	RANDOM FILE?
			11 D #KAN11L	
	C925 10 27 DC ED	LI	DEU LAGIO	'FM' BAD FILE MODE ERROR IF RANDOM
1273	C929 5F	C	LKD	FILE NOT EMPTY FLAG - SET TO NOT EMPTY
1274	C92A A6 88 1Ø		DA FCBCFL,X	*CHECK THE CACHE FLAG - BRANCH IF
1275	C92D 26 Ø3	BI	NE LC932	*THERE IS A CHARACTER WHICH HAS BEEN CACHED
1276	C92F E6 88 17	LI	DB FCBDFL,X	GET SEQUENTIAL INPUT FILE STATUS
1277	C932 7E A5 E4		MP LA5E4	LINK BACK TO BASIC'S EOF STATEMENT
1278	0302 72 710 21	20302 0.	2.102.1	ETHIC BROKE TO BROTE O EST STRIETER
		+ OFT FILE	NAME/EXTENSION: DRIVE NUMBER FROM	I DACTO
1279				
1280	C935 8E C2 A9		DX #DEFEXT	POINT TO ' ' BLANK (DEFAULT) EXTENSION
1281	C938 6F E2	CI	LR ,-S	CLEAR A BYTE ON STACK FOR USE AS A DRIVES FLAG
1282	C93A B6 Ø9 5A	LI	DA DEFDRV	* GET DEFAULT DISK NUMBER
1283	C93D 97 EB	S.	TA DCDRV	* STORE IN DSKCON PARAMETER
1284	C93F CE Ø9 4C		DU #DNAMBF	DISK FILENAME BUFFER
1285			DD #\$2008	STORE 8 BLANKS IN RAM (DEFAULT FILE NAME)
	C945 A7 CØ		ΓA ,U+	STORE A BLANK IN FILE NAME
1287	C947 5A		ECB	DECREMENT COUNTER
1288	C948 26 FB	BI	NE LC945	BRANCH IF NOT DONE
1289	C94A C6 Ø3	LI	DB #\$Ø3	3 BYTES IN EXTENSION
1290	C94C BD A5 9A		SR LA59A	MOVE B BYTES FROM (X) TO (U)
			SR L8748	EVALUATE A STRING EXPRESSION
		LI	EAU ,X	POINT U TO START OF STRING
1293			MPB #\$Ø2	* CHECK LENGTH OF STRING AND
1294	C956 25 12	BI	LO LC96A	* BRANCH IF < 2
1295	C958 A6 41		DA \$01,U	= GET 2ND CHARACTER IN STRING AND
1296	C95A 81 3A	CI	MPA #':'	= CHECK FOR COLON
1297	C95C 26 ØC	BI	NE LC96A	BRANCH IF NO DRIVE NUMBER
1298	C95E A6 C4		DA ,U	* GET 1ST CHARACTER
1299	C96Ø 81 3Ø		MPA #'Ø'	* IN STRING AND
1300	C962 25 Ø6		LO LC96A	* CHECK TO SEE
1301	C964 81 33		MPA #'3'	* IF IT IS IN
1302	C966 22 Ø2	BI	HI LC96A	* THE RANGE Ø-3
13Ø3	C968 8D 33	B:	SR LC99D	GET DRIVE NUMBER
1304	COCA OF GO AC		DX #DNAMBF	POINT X TO FILE NAME BUFFER
	C96A 8E Ø9 4C	LC96A LI		
1305			NCB	COMPENSATE FOR DECB BELOW
1305	C96D 5C	II	NCB FCB	COMPENSATE FOR DECB BELOW DECREMENT STRING LENGTH
1306	C96D 5C C96E 5A	LC96E DI	ECB	DECREMENT STRING LENGTH
1306 1307	C96D 5C C96E 5A C96F 26 ØC	LC96E DI BI	ECB NE LC97D	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING
1306 1307 1308	C96D 5C C96E 5A C96F 26 ØC C971 32 61	LC96E DI BI LI	ECB NE LC97D EAS \$01,S	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK – REMOVE DRIVE FLAG
1306 1307 1308 1309	C96D 5C C96E 5A C96F 26 ØC C971 32 61 C973 8C Ø9 4C	LC96E DI BI LI LC973 CI	ECB NE LC97D EAS \$01,S MPX #DNAMBF	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER?
1306 1307 1308 1309 1310	C96D 5C C96E 5A C96F 26 ØC C971 32 61 C973 8C Ø9 4C C976 26 67	LC96E DI BI LC973 CI BI	ECB NE LC97D EAS \$01,S MPX #DNAMBF NE LC9DF	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT
1306 1307 1308 1309 1310 1311	C96D 5C C96E 5A C96F 26 ØC C971 32 61 C973 8C Ø9 4C C976 26 67 C978 C6 3E	LC96E DI BI LI LC973 CI BI LC978 LI	ECB NE LC97D EAS \$01,S MPX #DNAMBF NE LC99F DB #2*31	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME
1306 1307 1308 1309 1310 1311	C96D 5C C96E 5A C96F 26 ØC C971 32 61 C973 8C Ø9 4C C976 26 67	LC96E DI BI LI LC973 CI BI LC978 LI	ECB NE LC97D EAS \$01,S MPX #DNAMBF NE LC9DF	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT
1306 1307 1308 1309 1310 1311	C96D 5C C96E 5A C96F 26 ØC C971 32 61 C973 8C Ø9 4C C976 26 67 C978 C6 3E	LC96E DI BI LC973 CI BI LC978 LI JI	ECB  VE LC97D  EAS \$01,S  MPX #DNAMBF  VE LC99F  DB #2*31  MP LAC46	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME
1306 1307 1308 1309 1310 1311 1312 1313	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0	LC96E DI BI LI LC973 CI BI LC978 LI JI LC97D LI	ECB  NE LC97D  FAS \$01,S  MPX #DNAMBF  NE LC9DF  DB #2*31  LAC46  DA ,U+	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING
1306 1307 1308 1309 1310 1311 1312 1313	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0	LC96E DI BI LI LC973 CI BI LC978 LI JI LC97D LI	ECB  VE LC97D  EAS \$01,S  MPX #DNAMBF  VE LC99F  DB #2*31  MP LAC46	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD?
1306 1307 1308 1309 1310 1311 1312 1313	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0	LC96E DI BI LI LC973 CI BI LC978 LI JI LC97D LI	ECB  VE LC97D  EAS \$01,S  MPX #DNAMBF  VE LC90F  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC9B0	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES
1306 1307 1308 1309 1310 1311 1312 1313	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0	LC96E DI BI LI LC973 CI BI LC978 LI JI LC97D LI	ECB  VE LC97D  EAS \$01,S  VEX #DNAMBF  VE LC9DF  DB #2*31  VP LAC46  DA ,U+  VPA #'.'  EC9BØ  VPA #'/'	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH?
1306 1307 1308 1309 1310 1311 1312 1313	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0	LC96E DI BI LI LC973 CI BI LC978 LI JI LC97D LI	ECB  VE LC97D  LC97D  LC97D  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #'/'  EQ LC980	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES
1306 1307 1308 1309 1310 1311 1312 1313	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0	LC96E DI BI LI LC973 CI BI LC978 LI JI LC97D LI	ECB  VE LC97D  EAS \$01,S  MPX #DNAMBF  VE LC99F  DB #2**11  MP LAC46  DA ,U+  MPA #'.'  EQ LC9BØ  MPA #'/'  MPA #'/'  MPA #'/'  MPA #'.'	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON?
1306 1307 1308 1309 1310 1311 1312 1313	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0	LC96E DI BI LI LC973 CI BI LC978 LI JI LC97D LI	ECB  VE LC97D  EAS \$01,S  MPX #DNAMBF  NE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #'/'  EQ LC980  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES
1306 1307 1308 1309 1310 1311 1312 1313	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0	LC96E DI BI LI LC973 CI BI LC978 LI JI LC97D LI	ECB  VE LC97D  CAS \$01,S  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  A, U+  MPA #'.'  EQ LC980  MPA #':  EQ CC980  MPA #':  MPA #	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER
1306 1307 1308 1309 1310 1311 1312 1313	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0	LC96E DI BI LI LC973 CI BI LC978 LI JI LC97D LI	ECB  VE LC97D  EAS \$01,S  MPX #DNAMBF  NE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #'/'  EQ LC980  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321	C96D 5C C96E 5A C96F 26 ØC C971 32 61 C973 8C Ø9 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 CØ C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 Ø9 C988 8C Ø9 54	LC978 LI LC978 LI LC977 CI LC978 LI LC97D LI CG CI BI CI BI BI CI BI BI BI BI BI BI BI BI BI BI BI BI BI	ECB  VE LC97D  CAS \$01,S  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  A, U+  MPA #'.'  EQ LC980  MPA #':  EQ CC980  MPA #':  MPA #	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322	C96D 5C C96E 5A C96F 5A C971 32 61 C971 32 61 C973 8C 09 4C C976 66 7 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C98E 57 E8	LC96E DI LC973 CI LC978 LI LC97D LI CG97D CI BI CG1 BI CG1 BI BI CG1 BI	ECB  VE LC97D  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C978 C6 67 C978 C6 3E C97A 7E AC C97D A6 C0 C97F 81 2E C981 27 2D C981 27 2D C983 81 2F C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C988 8C 09 54 C988 BC 09 54 C988 BC 09 54 C998 DA	LC96E DI LC973 CI LC978 LI LC97D LI CG97D CI BI CG1 BI CG1 BI BI CG1 BI	ECB  VE LC97D  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324	C96D 5C C96E 5A C96F 5A C971 32 61 C973 8C 09 4C C971 32 66 C976 66 7 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C98E 7 E8 C990 8D 3E C992 20 DA C994 8D DD	LC96E DI	ECB  WE LC97D  EAS \$01,S  MPX #DNAMBF  WE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC9B0  MPA #'.'  EQ LC980  MPA #':  EQ LC980  MPA #':  EQ LC980  MPA #':  EQ LC980  MPA #':  EQ LC980  MPA #C900  MPA #	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 8C 09 4C C978 C6 67 C978 C6 3E C97A 7E AC 46 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C996 8D 3E C992 20 DA C994 8D DD C996 8D 05	LC96E DI LC978 LI LC978 LI LC970 LI C670 CI BI C670 BI C670 BI C7 C1 BI BI C7 C1 BI BI C7 C1 BI BI BI C7 C1 BI BI BI C7 C1 BI	ECB  VE LC97D  LC97B  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325	C96D 5C C96E 5A C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C98E 27 E8 C990 8D 3E C992 20 DA C994 8D DD C996 8D 05 C998 5D	LC96E DI BI	ECB  VE LC97D  CAS \$01,S  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #':'  EQ LC980  MPA #':'  EQ LC994  MPX #DEXTBF  EQ LC978  SR LC973  SR LC973  SR LC973  SR LC990  STB	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325	C96D 5C C96E 5A C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C98E 27 E8 C990 8D 3E C992 20 DA C994 8D DD C996 8D 05 C998 5D	LC96E DI BI	ECB  WE LC97D  EAS \$01,S  MPX #DNAMBF  WE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA H'.'  EQ LC9B0  MPA H'.'  EQ LC9B0  MPA H':  EQ LC9B0  MPA H':  EQ LC994  MPX #DEXTBF  EQ LC994  MPX #DEXTBF  EQ LC998  RE LC990  SR LC990  STB  WE LC978	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * "'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1319 1320 1321 1322 1323 1324 1325 1326 1327 1328	C96D 5C C96E 5A C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C976 26 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C98E 27 E8 C990 8D 3E C992 20 DA C994 8D DD C996 8D 05 C998 5D	LC96E DI BI	ECB  VE LC97D  CAS \$01,S  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #':'  EQ LC980  MPA #':'  EQ LC994  MPX #DEXTBF  EQ LC978  SR LC973  SR LC973  SR LC973  SR LC990  STB	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 8C 09 4C C978 C6 67 C978 C6 3E C97A 7E AC 46 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C996 8D 3E C992 20 DA C994 8D DD C996 8D 05	LC96E DI BI	ECB  WE LC97D  EAS \$01,S  MPX #DNAMBF  WE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA H'.'  EQ LC9B0  MPA H'.'  EQ LC9B0  MPA H':  EQ LC9B0  MPA H':  EQ LC994  MPX #DEXTBF  EQ LC994  MPX #DEXTBF  EQ LC998  RE LC990  SR LC990  STB  WE LC978	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * "'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB WE LC97D LC35 \$01,S MPX #DNAMBF WE LC9DF DB #2*31 MP LAC46 DA, U+ MPA #'.' EQ LC980 MPA #':' EQ LC980 MPA #':' EQ LC994 MPX #DEXTBF EQ LC978 SR LC990 RA LC96E SR LC973 SR LC99D STB WE LC978 STB WE LC978 JLS A,PC	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * "'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * ''BAD FILENAME' FROM STRING * ''BAD FILENAME' FROM STACK AND RETURN  TOGGLE DRIVE FLAG
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  CAS \$01,S  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980  MPA #':  EQ LC980  MPA #':  EQ LC994  MPX #DEXTBF  EQ LC978  SR LC973  SR LC973  SR LC990  SR LC973  SR LC990  VE NUMBER  DD \$02,S  EQ U\$986  M\$62,S  EQ U\$986  M\$62,S  EQ LC978	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 67 C978 C6 3E C97A 7E AC 46 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * ''BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 67 C978 C6 3E C97A 7E AC 46 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  LC97D  LC97D  LC97D  LC97D  LC98D  LC99A  LC99B  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * ''BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X)
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 67 C978 C6 3E C97A 7E AC 46 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  CAS \$01, S  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #':'  EQ LC980  MPA #':'  EQ LC994  MPA #':'  EQ LC994  MPA #C994  MPA #C994  MPA #C995  MPA #C995  MPA #C996  MPA #C996  MPA #C996  MPA #C996  MPA #C998  MPA LC96E  SR LC978  SR LC990  SR LC990  WE NUMBER  DM \$02, S  EQ LC978  DA ,U++  JBB #\$02  JBB #\$02	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 67 C978 C6 3E C97A 7E AC 46 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  LC97D  LC97D  LC97D  LC97D  LC98D  LC99A  LC99B  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * ''BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X)
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 67 C978 C6 3E C97A 7E AC 46 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  LC97D  LC95C  VEX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980  MPA #':'  EQ LC994  MPA #':'  EQ LC994  MPA #':'  EQ LC995  MPA #C996  MPA #WE NUMBER  MPA	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 67 C978 C6 3E C97A 7E AC 46 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  LC97D  LC95C  VEX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980  MPA #':'  EQ LC994  MPA #':'  EQ LC994  MPA #':'  EQ LC995  MPA #C996  MPA #WE NUMBER  MPA	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * ''BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS DRIVE NUMBER
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1328 1328	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 67 C978 C6 67 C978 C6 3E C97A 7E AC 46 C97F 81 2E C981 27 2D C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C998 8D 3E C999 20 DA C994 8D DD C996 8D 05 C998 5D C999 26 DD C999 35 82	LC978 LI LC978 LI LC978 LI LC970 LI CG970 CI BI CG970	ECB  VE LC97D  CAS \$01, S  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980  MPA #':  EQ LC980  MPA #':  EQ LC994  MPA #':  EQ LC994  MPA #C994  MPA #C994  MPA #C994  MPA #C994  MPA #C994  MPA #C996  MPA #C996  MPA #C996  MPA LC978  SER LC973  SER LC973  SER LC973  SER LC978  SER LC978  SER LC978  SER LC978  ME LC	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS DRIVE NUMBER TOO LOW - 'BAD FILENAME' ERROR MAX OF 4 DRIVES DRIVE NUMBER TOO HIGH - 'BAD FILENAME' ERROR
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1329 1321 1322 1323 1324 1325 1326 1327 1328 1329 1321 1333 1334 1331 1332 1333 1334 1336 1337 1338 1339	C96D 5C C96E 5A C96E 5A C96F 26 ØC C971 32 61 C973 8C 09 4C C971 8C 66 7 C978 C6 3E C97A 7E AC 46 C97F 81 2E C98B 27 2D C98B 81 2F C985 27 29 C98B 8C 09 54 C98P 27 09 C98B 8C 09 54 C99C 8D 3E C992 2Ø DA C992 2Ø DA C992 8D 05 C998 5D C998 5D C999 8D 05 C998 5D C999 36 62 C998 37 82 C999 63 62 C999 63 62 C998 70 70 C998 8C 09 C998 8D 05 C998 3C 00 C998	LC96E   DI	ECB  VE LC97D  LC97B	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * ''BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS DRIVE NUMBER TOO LOW - 'BAD FILENAME' ERROR MAX OF 4 DRIVES
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1331 1332 1331 1332 1333 1334 1335 1336 1337 1338 1337 1338	C96D 5C C96E 5A C96E 5A C96F 26 ØC C971 32 61 C973 8C 09 4C C971 8C 66 7 C978 C6 3E C97A 7E AC 46 C97F 81 2E C98B 27 2D C98B 81 2F C985 27 29 C98B 8C 09 54 C98P 27 09 C98B 8C 09 54 C99C 8D 3E C992 2Ø DA C992 2Ø DA C992 8D 05 C998 5D C998 5D C999 8D 05 C998 5D C999 36 62 C998 37 82 C999 63 62 C999 63 62 C998 70 70 C998 8C 09 C998 8D 05 C998 3C 00 C998	LC96E   DI	ECB  VE LC97D  CAS \$01, S  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980  MPA #':  EQ LC980  MPA #':  EQ LC994  MPA #':  EQ LC994  MPA #C994  MPA #C994  MPA #C994  MPA #C994  MPA #C994  MPA #C996  MPA #C996  MPA #C996  MPA LC978  SER LC973  SER LC973  SER LC973  SER LC978  SER LC978  SER LC978  SER LC978  ME LC	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS DRIVE NUMBER TOO LOW - 'BAD FILENAME' ERROR MAX OF 4 DRIVES DRIVE NUMBER TOO HIGH - 'BAD FILENAME' ERROR
1306 1307 1308 1309 1310 1311 1312 1313 1314 1315 1316 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1333 1334 1335 1336 1337	C96D 5C C96E 5A C96E 5A C96F 26 ØC C971 32 61 C973 8C 09 4C C971 8C 66 7 C978 C6 3E C97A 7E AC 46 C97F 81 2E C98B 27 2D C98B 81 2F C985 27 29 C98B 8C 09 54 C98P 27 09 C98B 8C 09 54 C99C 8D 3E C992 2Ø DA C992 2Ø DA C992 8D 05 C998 5D C998 5D C999 8D 05 C998 5D C999 36 62 C998 37 82 C999 63 62 C999 63 62 C998 70 70 C998 8C 09 C998 8D 05 C998 3C 00 C998	LC96E DI LC978 LI LC978 LI LC97D LI CG7D CI BB BC7D CI CG7D CI BB CG7D CI CG7D	ECB WE LC97D LC35 \$01, S MPX #DNAMBF WE LC9DF DB #2*31 MP LAC46 DA ,U+ MPA #'.' EQ LC980 MPA #'.' EQ LC980 MPA #':' EQ LC980 MPA #':' EQ LC980 MPA #':' EQ LC994 MPX #DEXTBF EQ LC978 SC LC978 S	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK. REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS DRIVE NUMBER TOO LOW - 'BAD FILENAME' ERROR MAX OF 4 DRIVES DRIVE NUMBER TOO HIGH - 'BAD FILENAME' ERROR
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1329 1321 1322 1323 1324 1325 1326 1327 1328 1329 1330 1331 1332 1334 1335 1336 1337 1338 1339 1340 1331 1339	C96D 5C C96E 5A C96E 5A C96F 26 ØC C971 32 61 C973 8C 09 4C C971 8C 66 7 C978 C6 3E C97A 7E AC 46 C97F 81 2E C98B 81 27 2D C98B 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C990 8D 3E C992 2Ø DA C992 2Ø DA C998 8D 05 C998 5D C998 8D 05 C998 8D C999 8D 05 C998 8D C999 8D C999 8D C998 8D C999 8D C999 8D C998 8D C999 8D C998 8D C999 8D C998 8	LC996 DI LC978 LI LC978 LI LC970 LI CG70 LI CG	ECB  VE LC97D  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * ''BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS DRIVE NUMBER TOO LOW - 'BAD FILENAME' ERROR MAX OF 4 DRIVES DRIVE NUMBER TOO LOB - 'BAD FILENAME' ERROR STORE IN DSKCON DRIVE NUMBER
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1331 1332 1333 1334 1335 1336 1337 1338 1337 1338 1337 1338 1337 1338 1337 1338 1337 1348 1349 1341 1341 1341 1341 1341 1341 1341	C96D 5C C96E 5A C96F 26 0C C971 32 61 C973 8C 09 4C C971 32 61 C973 8C 67 C978 C6 3E C97A 7E AC 46 C97D A6 C0 C97F 81 2E C981 27 2D C983 81 2F C983 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C989 8D 3E C990 8D 3E C990 8D 3E C990 8D 0D C998 5D C999 26 DD C998 5D C999 35 82 C990 63 62 C997 27 07 C941 A6 C1 C943 C0 02 C945 80 30 C947 25 CF C949 81 03 C948 22 CB C948 39 C948 80 C1	LC96E   DI	ECB  VE LC97D  LC97D  LC97D  LC95C  MPX #DNAMBF  VE LC9DF  DB #2*31  MP LAC46  DA ,U+  MPA #'.'  EQ LC980  MPA #'.'  EQ LC980  MPA #':'  EQ LC980  MPA #':'  EQ LC994  MPX #DEXTBF  EQ LC978  RR LC996  RR LC996  RR LC996  RR LC997  RR LC998  MPA #'.'  LC978  MPA LC96E  LC978  MPA LC96E  MPA LC978  MPA LC978  MPA LC978  MPA #\delta \text{MPA}  MPA LC978  MPA MPA	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * ''BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS DRIVE NUMBER TOO LOW - 'BAD FILENAME' ERROR MAX OF 4 DRIVES DRIVE NUMBER TOO HIGH - 'BAD FILENAME' ERROR STORE IN DSKCON DRIVE NUMBER
1306 1307 1308 1309 1311 1312 1313 1314 1315 1316 1317 1318 1320 1321 1322 1323 1324 1325 1326 1327 1328 1329 1331 1332 1333 1334 1335 1336 1337 1338 1337 1338 1337 1338 1337 1338 1337 1338 1337 1348 1349 1341 1341 1341 1341 1341 1341 1341	C96D 5C C96E 5A C96E 5A C96F 26 ØC C971 32 61 C973 8C 09 4C C971 8C 66 7 C978 C6 3E C97A 7E AC 46 C97F 81 2E C98B 81 27 2D C98B 81 2F C985 27 29 C987 81 3A C989 27 09 C988 8C 09 54 C989 27 09 C988 8C 09 54 C990 8D 3E C992 2Ø DA C992 2Ø DA C998 8D 05 C998 5D C998 8D 05 C998 8D C999 8D 05 C998 8D C999 8D C999 8D C998 8D C999 8D C999 8D C998 8D C999 8D C998 8D C999 8D C998 8	LC96E   DI	ECB  VE LC97D  L	DECREMENT STRING LENGTH BRANCH IF MORE CHARACTERS IN STRING CLEAN UP STACK - REMOVE DRIVE FLAG POINTER STILL AT START OF BUFFER? RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME ERROR HANDLER GET A CHARACTER FROM STRING LOOK FOR PERIOD? YES SLASH? YES COLON? YES COMPARE POINTER TO END OF FILENAME BUFFER 'BAD FILENAME' ERROR - FILENAME TOO LONG PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING 'BAD FILENAME' ERROR IF NO FILENAME YET GET DRIVE NUMBER * CHECK LENGTH OF STRING * ''BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN  TOGGLE DRIVE FLAG 'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE ASCII VALUE OF DRIVE NUMBER TO ACCA DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X) SUBTRACT ASCII BIAS DRIVE NUMBER TOO LOW - 'BAD FILENAME' ERROR MAX OF 4 DRIVES DRIVE NUMBER TOO LOB - 'BAD FILENAME' ERROR STORE IN DSKCON DRIVE NUMBER

1345	C9B5 86 20	LDA	#SPACE	BLANK
1346	C9B7 A7 82	LC9B7 STA		*
1347 1348	C9B9 8C Ø9 54 C9BC 26 F9		X #DEXTBF LC9B7	* FILL EXTENSION WITH * BLANKS (DEFAULT)
1349	C9BE 5A	LC9BE DEC		DECREMENT STRING COUNTER
1350	C9BF 27 DA		LC99B	RETURN IF ZERO
1351	C9C1 A6 CØ	LDA	. ,U+ A #':'	GET A CHARACTER FROM STRING
1352 1353	C9C3 81 3A C9C5 27 CD		LC994	*CHECK FOR DRIVE SEPARATOR *
1354	C9C7 8C Ø9 57		X #DFLTYP	=CHECK FOR END OF ESTENSION RAM BUFFER &
1355	C9CA 27 AC		LC978	= 'BAD FILENAME' ERROR IF EXTENSION TOO LONG
1356 1357	C9CC 8D Ø2 C9CE 2Ø EE	BSR BRA		PUT A CHARACTER IN EXTENSION BUFFER GET ANOTHER EXTENSION CHARACTER
1358	0302 20 22	Ditt	20382	der Another Extension diamondren
1359			ARACTER INTO FILENAME OR EXTENSION	
1360 1361	C9DØ A7 8Ø C9D2 27 A4	LC9DØ STA BEQ		STORE CHARACTER IN FILENAME BUFFER 'BAD FILENAME' ERROR; ZEROES ARE ILLEGAL
1362	C9D4 81 2E		A #'.'	PERIOD?
1363	C9D6 27 AØ	BEQ	LC978	'BAD FILENAME' ERROR IF PERIOD
1364	C9D8 81 2F		A #'/'	SLASH?
1365 1366	C9DA 27 9C C9DC 4C	INC	LC978 A	'BAD FILENAME' ERROR IF SLASH CHECK FOR \$FF
1367	C9DD 27 99	BEQ		'BAD FILENAME' ERROR IF \$FF
1368	C9DF 39	LC9DF RTS		
1369 1370		* SAVE COMMA	IND	
1371	C9EØ 81 4D		A #'M'	*
1372	C9E2 10 27 05 82		Q LCF68	*BRANCH IF SAVEM
1373 1374	C9E6 8D 4B C9E8 9E 8A	BSR LDX	LCA33 ZERO	GO GET FILENAME, ETC. FROM BASIC ZERO OUT X REG
1375	C9EA BF Ø9 57	STX		SET FILE TYPE AND ASCII FLAG TO ZERO
1376	C9ED 9D A5	JSR	GETCCH	GET CURRENT INPUT CHARACTER FROM BASIC
1377 1378	C9EF 27 21 C9F1 BD B2 6D	BEQ 19 D	LCA12 SYNCOMMA	BRANCH IF END OF LINE
1378	C9F4 C6 41		#'A'	SYNTAX CHECK FOR COMMA *ASCII FILE?
1380	C9F6 BD B2 6F	JSR	LB26F	*SYNTAX CHECK ON CONTENTS OF ACCB
1381	C9F9 26 E4		LC9DF	RETURN IF NO MORE CHARACTERS ON LINE
1382 1383	C9FB 73 Ø9 58 C9FE 8D Ø4	BSR	DASCFL LCAØ4	SET CRUNCHED/ASCII FLAG TO ASCII OPEN A SEQUENTIAL FILE FOR OUTPUT
1384	CAØØ 4F	CLR		SET ZERO FLAG - CAUSE ENTIRE FILE TO BE LISTED
1385	CAØ1 7E B7 64	JMP	LIST	'LIST' THE FILE TO CONSOLE OUT
1386 1387		* ODEN V CEU	QUENTIAL FILE FOR INPUT/OUTPUT - US	THE CYCTEM
1388			ED AT THE TOP OF FCBS	DE THE STOTEM
1389	CAØ4 86 4F		#'0'	OUTPUT FILE TYPE
139Ø 1391	CAØ6 8C		X #\$8649 . #'I'	SKIP TWO BYTES INPUT FILE TYPE
1391	CAØ7 86 49 CAØ9 F6 Ø9 5B	LCAØ7 LDA LDB		GET NUMBER OF RESERVED FILES CURRENTLY RESERVED
1393	CAØC 5C	INC		ADD ONE - USE ONE ABOVE HIGHEST RESERVED FCB
1394	CAØD D7 6F	STB		SAVE IT IN DEVICE NUMBER
1395 1396	CAØF 7E C4 8D	JMP * SAVE A CRI	LC48D UNCHED FILE - A PREAMBLE OF THREE B	OPEN A FILE & INITIALIZE FCB
1397			TE 1 = \$FF, 2,3 = LENGTH OF BASIC P	
1398	CA12 8D FØ		LCAØ4	OPEN A SEQUENTIAL FILE FOR OUTPUT
1399 1400	CA14 86 FF CA16 BD CC 24		#\$FF LCC24	BASIC FILE FLAG CONSOLE OUT
1401	CA19 DC 1B	LDD		LOAD ACCD WITH START OF VARIABLES
1402	CA1B 93 19		D TXTTAB	SUBTRACT START OF BASIC
1403 1404	CA1D BD CC 24 CA2Ø 1F 98		LCC24 B.A	CONSOLE OUT FILE LENGTH MS BYTE PULL LS BYTE INTO ACCA
1405	CA22 BD CC 24	JSR	,	CONSOLE OUT FILE LENGTH LS BYTE
1406	CA25 9E 19	LDX		POINT X TO START OF BASIC
1407 1408	CA27 A6 8Ø CA29 BD CC 24	LCA27 LDA JSR	•	GET BYTE FROM BASIC SEND TO CONSOLE OUT
1409	CA2C 9C 1B		X VARTAB	COMPARE TO END OF BASIC
1410	CA2E 26 F7	BNE	LCA27	KEEP GOING IF NOT AT END
1411 1412	CA3Ø 7E A4 2D CA33 8E C2 A6	JMP	LA42D #BASEXT	CLOSE FILE POINT TO 'BAS' EXTENSION (DEFAULT)
1412	CA36 7E C9 38		LC938	GET FILENAME.EXT FROM BASIC
1414				
1415 1416	CA39 4F	* MERGE COMM MERGE CLR		DIM FLAC (A - DON'T DIM)
1416	CA3A C6 FF		A #\$FF	RUN FLAG (Ø = DON'T RUN) MERGE FLAG (\$FF = MERGE)
1418	CA3C 2Ø 12		LCA5Ø	GO LOAD THE FILE
1419				
1420 1421	CA3E 81 22	* RUN RAM VE DVEC18 CMP		CHECK FOR FILENAME DELIMITER (DOUBLE QUOTE)
1422	CA4Ø 1Ø 26 B8 58		E XVEC18	NONE - JUMP TO EXBAS RUN RAM HOOK
1423	CA44 86 Ø2		#\$02	RUN FLAG - DON'T CLOSE ALL FILES BEFORE RUN
1424 1425	CA46 20 07	BKA	LCA4F	LOAD THE FILE
1426		* LOAD COMMA	AND	
1427	CA48 81 4D		A #'M'	*
1428 1429	CA4A 10 27 05 73 CA4E 4F	LBE CLR	Q LCFC1	*BRANCH IF LOADM RUN FLAG = ZERO (DON'T RUN)
	CA4E 4F CA4F 5F	LCA4F CLR		CLEAR MERGE FLAG
1431	CA5Ø B7 Ø9 59	LCA5Ø STA	DRUNFL	RUN FLAG (Ø = DON'T RUN, 2 = RUN)
1432	CA53 F7 Ø9 5E		DMRGFL	MERGE FLAG (Ø = NO MERGE, \$FF = MERGE)
	CA56 8D DB CA58 9D A5		LCA33 GETCCH	GO GET FILENAME, ETC. FROM BASIC GET CURRENT INPUT CHAR
1435	CA5A 27 10	BEQ	LCA6C	BRANCH IF END OF LINE
1436	CASC BD B2 6D		SYNCOMMA #'R'	SYNTAX CHECK FOR COMMA
1437 1438	CA5F C6 52 CA61 BD B2 6F		#'R' LB26F	* *IS NEXT CHAR 'R'? RUN AFTER LOAD
1439	CA64 BD A5 C7	JSR	LA5C7	SYNTAX ERROR IF ANY MORE CHARS ON LINE
1440	CA67 86 Ø3	LDA	#\$Ø3	*SET FLAGS TO RUN AND CLOSE ALL FILES

1441	CA69 B7 Ø9 59		STA	DRUNFL	*BEFORE THE FILE IS RUN
1442	CA6C 8D 99	LCA6C	BSR	LCAØ7	GRAB FCB FOR INPUT FILE
1443	CA6E B6 Ø9 58			DASCFL	*CHECK ASCII FLAG AND BRANCH
1444	CA71 27 ØB		BEQ	LCA7E	*IF CRUNCHED BASIC FILE
1445	CA73 7D Ø9 5E			DMRGFL	IS THIS A MERGE?
1446	CA76 26 Ø3			LCA7B	BRANCH IF MERGE
1447 1448	CA78 BD AD 19	I CA7D		LAD19	DO A 'NEW' - ERASE VARIABLES, RESET VARIABLES GO TO BASIC'S MAIN LOOP, IT WILL LOAD PROGRAM
1446	CA7B 7E AC 7C	LCA7B	UMP	LAC7C	GU TU BASIC S MAIN LOUP, IT WILL LUAD PROGRAM
1450		* LOAD	TN A C	RUNCHED BASIC FILE	
1451	CA7E B6 Ø9 57	LCA7E		DFLTYP	*CHECK FILE TYPE (MUST BE BASIC:0) & CHECK
1452	CA81 BA Ø9 5E			DMRGFL	*MERGE FLAG (MUST BE NO MERGE: Ø)
1453	CA84 10 26 DB 8E		LBNE	LA616	'BAD FILE MODE' ERROR IF MERGE OR NON-BASIC
1454	CA88 BD AD 19			LAD19	DO A 'NEW' - RESET POINTERS, ERASE VARIABLES
1455	CA8B 73 Ø9 5D		COM	DLODFL	* SET THE LOAD FLAG TO \$FF - THIS WILL CAUSE A NEW TO
1456	CASE BD CD BC	*	100	LCDDC	* OCCUR IF AN ERROR OCCURS WHILE THE PROGRAM IS BEING LOADED
1457 1458	CA91 BD CD BC			LCDBC LCDBC	GET CHAR FROM BUFFER - SHOULD BE \$FF GET ANOTHER - MS BYTE OF LENGTH
1459	CA94 34 Ø2		PSHS		SAVE MS BYTE ON STACK
1460	CA96 BD CD BC			LCDBC	LS BYTE OF LENGTH OF PROGRAM
1461	CA99 1F 89		TFR		PUT LS BYTE INTO ACCB
1462	CA9B 35 Ø2		PULS		NOW ACCD CONTAINS LENGTH OF PROGRAM
1463	CA9D D3 19			TXTTAB	ADD BEGINNING OF BASIC
1464	CA9F BD AC 37			LAC37	SEE OF ENOUGH ROOM IN RAM FOR THIS FILE
1465	CAA2 9E 19	1.0444		TXTTAB	GET START OF BASIC
1466	CAA4 BD C5 C4 CAA7 D6 7Ø	LCAA4		LC5C4 CINBFL	READ A CHAR FROM CONSOLE IN
1467 1468	CAA9 26 Ø4			LCAAF	BUFFER EMPTY? BRANCH IF SO
1469	CAAB A7 8Ø		STA	, X+	STORE CHAR
1470	CAAD 20 F5		BRA	LCAA4	GET ANOTHER CHARACTER
1471					
1472	CAAF 7F Ø9 5D	LCAAF		DLODFL	CLEAR LOAD FLAG - LOAD WAS ERROR FREE
1473	CAB2 9F 1B			VARTAB	SAVE NEW START OF VARIABLES
1474	0.01 05 00	* MAKE		AST THREE BYTES LOADED WERE ZERO	AUGON TURES RATEO
1475 1476	CAB4 C6 Ø3 CAB6 A6 82	LCAB6		#\$Ø3 ,-X	CHECK THREE BYTES CHECK A BYTE
1477	CAB8 26 Ø3	LUADO		LCABD	BRANCH IF NON-ZERO
1478	CABA 5A		DECB		DECREMENT COUNTER
1479	CABB 26 F9			LCAB6	KEEP CHECKING IF NOT DONE
1480	CABD 9E 1B	LCABD		VARTAB	GET START OF VARIABLES
1481	CABF 9F 1B	LCABF	STX	VARTAB	SAVE START OF VARIABLES
1482	CAC1 6F 8Ø		CLR		CLEAR A BYTE
1483	CAC3 5A		DECB		DECREMENT COUNTER
1484 1485	CAC4 2A F9 CAC6 BD A4 2D	LCAC6		LCABF LA42D	KEEP CLEARING BYTES IF NOT DONE CLOSE SELECTED FILE
1486	CACO BD A4 2D CACO BD AD 21	LUACO		LAD21	DO PART OF NEW - ERASE VARIABLES, RESET INPUT PTR
1487	CACC BD 82 9C			XVEC18	INITIALIZE EXBAS GRAPHICS VARIABLES
1488	CACF BD AC EF			LACEF	RELOCATE ALL THE BASIC NEXT LINE POINTERS
1489	CAD2 77 Ø9 59			DRUNFL	CHECK LSB OF RUN FLAG
1490	CAD5 25 Ø3		BLO	LCADA	BRANCH IF DON'T CLOSE ALL FILES
1491	CAD7 BD A4 26			LA426	CLOSE ALL FILES
1492	CADA 77 Ø9 59	LCADA		DRUNFL	TEST BIT 1 OF RUN FLAG
1493 1494	CADD 10 25 E2 BD			LAD9E	BRANCH TO COMM INTERPRETATION LOOP IF BIT 1 SET
1495	CAE1 7E AC 73		UNF	LAC73	RETURN TO DIRECT MODE
1496	CAE4 ØD 6F	DVEC13	TST	DEVNUM	* CHECK DEVICE NUMBER AND
1497	CAE6 2E DE		BGT	LCAC6	* TRY TO RUN FILE IF IT IS A DISK FILE
1498	CAE8 39		RTS		
1499					
1500				ILE BUFFERS RAM VECTOR	
1501	CAE9 F6 Ø9 5B	DVEC7		FCBACT	GET THE NUMBER OF RESERVED FILE BUFFERS
1502 1503	CAEC 5C CAED 34 Ø4	LCAED	INCB PSHS		ADD ONE SAVE IT
1503	CAEF D7 6F	FOVED		DEVNUM	STORE IT IN DEVICE NUMBER
1505	CAF1 8D ØE			LCBØ1	CLOSE FILE
1506	CAF3 35 Ø4		PULS		GET BACK NUMBER OF FILE BUFFERS
1507	CAF5 5A		DECB		DECREMENT FILE BUFFER COUNTER
1508	CAF6 26 F5	10450		LCAED	BRANCH IF ALL FILES NOT CLOSED
1509 1510	CAF8 39	LCAF8	RTS		
1510		* (100	FILE	RAM HOOK	
	CAF9 ØD 6F	DVEC8		DEVNUM	* CHECK DEVICE NUMBER AND RETURN
1513	CAFB 10 2F B7 87	00		XVEC8	* IF NOT A DISK FILE
1514	CAFF 32 62			\$Ø2,S	PURGE RETURN ADDRESS OFF OF THE STACK
1515	CBØ1 BD C7 44	LCBØ1		LC744	POINT X TO CORRECT FCB
1516	CBØ4 ØF 6F			DEVNUM	SET DEVICE NUMBER TO SCREEN
1517	CBØ6 9F F1	LCBØ6		FCBTMP	SAVE FILE BUFFER POINTER
1518 1519	CBØ8 A6 ØØ CBØA 27 EC			FCBTYP,X LCAF8	GET THE TYPE OF THIS FILE RETURN IF FILE NOT OPEN
1519	CBØC 34 Ø2		PSHS		SAVE FILE TYPE
1521	CBØE 6F ØØ			FCBTYP,X	CLOSE THE FILE - ZERO OUT THE FILE TYPE
	CB10 E6 01			FCBDRV,X	* GET DRIVE NUMBER AND
	CB12 D7 EB			DCDRV	* SAVE IT IN DSKCON VARIABLE
1524	CB14 81 20			#OUTFIL	= CHECK FOR OUTPUT TYPE AND
1525	CB16 26 19		BNE	LCB31	= BRANCH IF NOT OUTPUT TYPE FILE
1526 1527		* (100	- A CFA	HENTIAL CHITCHT ETLE	
	CB18 E6 88 18	CLUSE		UENTIAL OUTPUT FILE FCBLFT,X	GET THE NUMBER OF CHARACTERS IN BUFFER
1529	CB1B 86 8Ø			#\$80	* SET THE PRE-SAVED BIT TO INDICATE THAT THE DATA
1530		*			* HAS ALREADY BEEN SAVED ON DISK
	CB1D AA Ø5			FCBCPT,X	'OR' IN THE FULL SECTOR FLAG
			STD	FCBLST,X	SAVE THE NUMBER OF BYTES USED IN THE LAST SECTOR
1532	CB1F ED 88 13				
1533	CB22 6C Ø4			FCBSEC, X	INCREMENT THE SECTOR NUMBER
1533 1534	CB22 6C Ø4 CB24 E6 Ø3		LDB	FCBCGR, X	GET THE CURRENT GRANULE NUMBER
1533	CB22 6C Ø4		LDB JSR		

1537 1538	CB2B 3A CB2C 6C Ø6		ABX INC	FATCON, X	ADD GRANULE OFFSET TO FAT POINTER * INCREMENT GRANULE DATA (ADD ONE SECTOR TO LAST
1539 1540 1541	CB2E 7E CB C3 CB31 81 4Ø	* LCB2E LCB31		LCBC3 #RANFIL	* GRANULE) SKIP PAST THE SIX FAT CONTROL BYTES UPDATE FAT AND DIRECTORY RANDOM FILE?
1542 1543	CB33 26 F9		BNE	LCB2E	NO - UPDATE FAT AND DIRECTORY IF SEQUENTIAL INPUT FILE
1544 1545	CB35 EC Ø9	* CLOSE		DOM FILE FCBRLN,X	GET RECORD LENGTH
1546	CB37 AE ØB			FCBBUF,X	POINT X TO RANDOM FILE BUFFER
1547	CB39 31 8B		LEAY		POINT Y TO END OF RANDOM FILE BUFFER
1548 1549	CB3B 34 36 CB3D 31 E4		LEAY PSH2	Y,X,B,A .S	SAVE POINTERS ON STACK POINT Y CURRENT STACK POINTER
1550	CB3F DE 1B		LDU	VARTAB	GET START OF VARIABLES
1551 1552	CB41 11 93 1D CB44 27 ØE	LCB41		ARYTAB LCB54	COMPARE TO START OF ARRAYS BRANCH IF ALL VARIABLES CHECKED
1553	CB46 A6 41		LDA	\$01,U	GET 2ND BYTE OF VARIABLE NAME
1554 1555	CB48 33 42 CB4A 2A Ø2			\$Ø2,U LCB4E	MOVE POINTER TO START OF DESCRIPTOR BRANCH IF VARIABLE - NUMERIC
1556	CB4C 8D 28			LCB76	ADJUST STRING VARIABLE IF IN RANDOM FILE BUFFER
1557 1558	CB4E 33 45 CB50 20 EF	LCB4E		\$Ø5,U LCB41	MOVE POINTER TO NEXT VARIABLE PROCESS ANOTHER VARIABLE
1559	CB52 35 4Ø	LCB52	PULS		GET ADDRESS OF NEXT ARRAY TO U
1560	CB54 11 93 1F	LCB54		ARYEND	COMPARE TO END OF ARRAYS
1561 1562	CB57 27 3A CB59 1F 3Ø		TFR	LCB93 U.D	BRANCH IF END OF ARRAYS * SAVE ARRAY START IN ACCD, ADD OFFSET
1563	CB5B E3 42		ADDD	\$Ø2,U	* TO NEXT ARRAY AND SAVE ADDRESS OF
1564 1565	CB5D 34 Ø6 CB5F A6 41		PSHS LDA	B,A \$01,U	* NEXT ARRAY ON THE STACK GET 2ND LETTER OF VARIABLE NAME
1566	CB61 2A EF		BPL	LCB52	BRANCH IF NUMERIC
1567 1568	CB63 E6 44 CB65 58		LDB ASLB	\$Ø4,U	GET THE NUMBER OF DIMENSIONS X2:2 BYTES PER DIMENSION
1569	CB66 CB Ø5		ADDB	#\$05	5 BYTES CONSTANT PER ARRAY DESCRIPTOR
1570	CB68 4F		CLRA	D. II	CLEAR MSB OF OFFSET - (ONLY 125 DIMENSIONS ALLOWED) POINT U TO START OF THIS ARRAY'S VARIABLES
1571 1572	CB69 33 CB CB6B 11 A3 E4	LCB6B	LEAU CMPU		AT END OF THIS ARRAY?
1573	CB6E 27 E2		BEQ	LCB52	YES
1574 1575	CB7Ø 8D Ø4 CB72 33 45			LCB76 \$05,U	ADJUST STRING VARIABLE IF IN RANDOM FILE BUFFER MOVE POINTER TO NEXT DESCRIPTOR
1576	CB74 20 F5			LCB6B	CHECK NEXT VARIABLE
1577 1578		* CHECK	TO SE	E IF A STRING IS LOCATED IN THE R	ANDOM FILE RUFFER AREA IF IT IS
1579		* THE RA	ANDOM	FILE BUFFER IN QUESTION, IT WILL	BE DELETED. IF IT IS HIGHER IN THE RANDOM
158Ø 1581				SPACE THAN THE BUFFER IN QUESTION BE SUBTRACTED FROM THE ADDRESS OF	
1582				EING DELETED (CLOSED).	THE STRING BECAUSE THE CONKENT
1583 1584	CB76 AE 42 CB78 BC Ø9 48	LCB76		\$Ø2,U RNBFAD	POINT X TO START OF STRING COMPARE TO START OF FREE RANDOM FILE BUFFER AREA
1585	CB7B 24 ØE			LCB8B	RETURN IF > START OF FREE RANDOM FILE BUFFER AREA
1586	CB7D AC 22			\$Ø2,Y	COMPARE TO START OF THIS FILE'S RANDOM BUFFER
1587 1588	CB7F 25 ØA CB81 AC 24			LCB8B \$04,Y	RETURN IF < START OF THIS FILE'S RANDOM BUFFER COMPARE TO END OF THIS FILE'S RANDOM BUFFER
1589	CB83 25 Ø7		BLO	LCB8C	RETURN IF < END OF THIS FILE'S RANDOM BUFFER
159Ø 1591	CB85 1F 1Ø CB87 A3 A4		TFR SUBD		SAVE POINTER IN ACCD SUBTRACT RECORD LENGTH FROM START OF STRING ADDRESS
1592	CB89 ED 42		STD	\$Ø2,U	SAVE NEW START OF STRING ADDRESS
1593 1594	CB8B 39 CB8C 6F C4	LCB8B LCB8C	RTS CLR	,U	CLEAR THE LENGTH OF THE STRING
1595	CB8E 6F 42	LUBOU	CLR	,0 \$Ø2,U	* CLEAR THE ADDRESS
1596	CB9Ø 6F 43		CLR	\$Ø3,U	* OF THE STRING
1597 1598	CB92 39	* REMOVE	RTS E RESE	RVED SPACE IN RANDOM FILE BUFFER	FOR A 'CLOSED' RANDOM FILE
1599	0000 56 85 55	* ADJUST	THE:	START OF RANDOM FILE BUFFER POINT	ER IN ALL RANDOM FCBS
1600 1601	CB93 F6 Ø9 5B CB96 5C	LCB93	LDB INCB	FCBACT	GET THE NUMBER OF ACTIVE FILES ADD ONE
1602	CB97 34 Ø4	LCB97	PSHS		SAVE FILES COUNT ON THE STACK
1603 1604	CB99 BD C7 49 CB9C A6 ØØ			LC749 FCBTYP,X	POINT X TO FCB GET FILE TYPE
1605	CB9E 81 40		CMPA	#RANFIL	IS IT A RANDOM FILE?
	CBAØ 26 ØB CBA2 EC ØB			LCBAD FCBBUF.X	BRANCH IF NOT GET START OF THIS FILE'S RANDOM FILE BUFFER
1608	CBA4 10 A3 24			\$04,Y	* COMPARE TO END OF RANDOM FILE BUFFER AREA AND
1609	CBA7 25 Ø4		BLO	LCBAD	* BRANCH IF < END OF RANDOM FILE BUFFER AREA
	CBA9 A3 A4 CBAB ED ØB		SUBD STD	,Y FCBBUF,X	= SUBTRACT RECORD LENGTH OF SELECTED FILE = SAVE NEW START OF RANDOM FILE BUFFER
1612	CBAD 35 Ø4	LCBAD	PULS	В	GET THE FILES COUNTER
1613 1614	CBAF 5A CBBØ 26 E5		DECB BNE	LCB97	DECREMENT FILES COUNTER BRANCH IF ALL FILES NOT DONE
1615	CBB2 35 56			A,B,X,U	* U = END OF RANDOM FILE BUFFER, X = START OF RANDOM
1616 1617					* FILE BUFFER, ACCD = RECORD LENGTH
1618				PROBABLY BE THE MOST CONVENIENT	
1619 1620		** CAUSE	ES THE	SYSTEM TO HANG IF AN ERROR IS EN	ICOUNTERED DURING 'COPY'
1621		*	CMPU	FCBADR	* IS THE END OF THIS FCB'S BUFFER ABOVE THE END
1622 1623		*	RIO	LCBB4	* OF THE START OF THE FCB AREA NO - FREE UP THE SPACE USED BY THIS FILE IN RANDOM BUFFER
1623		*	LDX	#DFLBUF	YES - DOING A 'COPY'; RESET START OF RANDOM BUFFER
1625		* DANDON		LCBCØ	
1626 1627				BUFFER AREA RVED SPACE FOR CLOSED FILE FROM R	ANDOM FILE BUFFER SPACE
1628 1629	CBB4 11 B3 Ø9 48	LCBB4	CMPII	RNBFAD	AT THE BOTTOM OF FREE RANDOM BUFFER AREA?
1630	CBB8 27 Ø6	LODDA	BEQ	LCBCØ	BRANCH IF THERE
	CBBA A6 CØ CBBC A7 8Ø		LDA STA		= GRAB A SOURCE BYTE AND = MOVE IT TO DESTINATION
1032	מס עו סממי		JIA	, n ·	- HOVE IT TO DESITINATION

1633	CBBE 20 F4			LCBB4	KEEP MOVING BYTES
1634 1635	CBCØ BF Ø9 48 CBC3 BD C7 55	LCBCØ LCBC3		RNBFAD LC755	SAVE NEW START OF FREE RANDOM BUFFER AREA POINT X TO PROPER FILE ALLOCATION TABLE
1636	CBC6 6A ØØ	LUBUS		FATØ,X	REMOVE ONE ACTIVE FILE
1637	CBC8 6D Ø1			FAT1,X	NEW DATA IN FAT RAM IMAGE?
1638 1639	CBCA 27 Ø3 CBCC BD C7 1E			LCBCF LC71E	NO WRITE OUT FILE ALLOCATION TABLE TO DISK
1640	CBCF 9E F1	LCBCF		FCBTMP	GET FILE BUFFER POINTER
1641	CBD1 35 Ø2		PULS		GET FILE TYPE
1642	CBD3 81 20			#OUTFIL	IS IT A SEQUENTIAL OUTPUT FILE?
1643 1644	CBD5 27 Ø8 CBD7 81 40			LCBDF #RANFIL	YES IS IT A RANDOM FILE?
	CBD9 26 BØ			LCB8B	RETURN IF NOT A RANDOM FILE (SEQUENTIAL INPUT)
	CBDB A6 ØF			FCBFLG,X	* TEST THE GET/PUT FLAG AND
1647 1648	CBDD 27 ØA		BEQ	LCBE9	* BRANCH IF 'GET'
1649		* WRITE	CONTEN	ITS OF FILE BUFFER TO DISK	
		LCBDF	JSR		GET PROPER TRACK & SECTOR NUMBERS
1651 1652	CBE2 33 88 19 CBE5 DF EE			FCBCON,X DCBPT	POINT U TO START OF FCB DATA SET UP FILE BUFFER POINTER FOR DSKCON
	CBE7 8D 2C		BSR		GO WRITE A SECTOR
		LCBE9		FCBLST,X	CHECK THE PRE-SAVED FLAG
	CBEC 2A 9D CBEE E6 88 12			LCB8B FCBDIR,X	RETURN IF RECORD HAS ALREADY BEEN SAVED ON DISK GET DIRECTORY NUMBER OF THIS FILE
1657	CBF1 C4 Ø7		ANDB		8 ENTRIES PER SECTOR
	CBF3 86 2Ø			#DIRLEN	DIRLEN BYTES PER DIRECTORY ENTRY
1659 1660	CBF5 3D CBF6 CE Ø6 ØØ		MUL	#DBUFØ	GET SECTOR OFFSET FOR THIS ENTRY  * GET READ/WRITE BUFFER Ø AND
	CBF9 DF EE			DCBPT	* SAVE IT IN DSKCON REGISTER
	CBFB 31 CB		LEAY		Y POINTS TO CORRECT DIRECTORY ENTRY
	CBFD E6 88 12 CCØØ 54		LDB	FCBDIR,X	GET DIRECTORY ENTRY NUMBER *
	CC00 54 CC01 54		LSRB		*
	CCØ2 54		LSRB		* DIVIDE BY 8; EIGHT DIRECTORY ENTRIES PER SECTOR
1667 1668	CCØ3 CB Ø3 CCØ5 D7 ED		ADDB STB		ADD BIAS; FIRST 3 SECTORS NOT DIRECTORY STORE SECTOR NUMBER
	CCØ7 CC 11 Ø2			#\$1102	DIRECTORY TRACK - READ OP CODE
	CCØA 97 EC			DCTRK	STORE TRACK NUMBER
	CCØC 8D Ø9 CCØE EC 88 13		BSR	FCBLST,X	GO READ DIRECTORY GET NUMBER OF BYTES IN THE LAST SECTOR
	CC11 84 7F		ANDA		MASK OFF THE PRE-SAVED FLAG
	CC13 ED 2E	LCC15		DIRLIST, Y	SAVE NUMBER OF BYTES IN LAST SECTOR OF FILE IN DIRECTORY
1675 1676	CC15 C6 Ø3 CC17 D7 EA	LCC15 LCC17	LDB STB		WRITE OP CODE SAVE DSKCON OP CODE VARIABLE
1677	CC19 7E D6 F2	20017	JMP		GO READ/WRITE SECTOR
1678			- AU-	244 11221	
1679 1680	CC1C ØD 6F	TONSUL DVEC3		RAM HOOK DEVNUM	CHECK DEVICE NUMBER
1681	CC1E 10 2F B6 51	51200		XVEC3	BRANCH TO EX BASIC IF NOT A DISK FILE
1682	CC22 32 62		LEAS		POP RETURN OFF STACK
1683 1684				CTER IN ACCA TO A DISK FILE. A CA ON AND CONTROL CODES WILL NOT INC	
	CC24 34 16	LCC24	PSHS	X,B,A	SAVE REGISTERS
	CC26 8E Ø9 26			#FCBV1-2	POINT X TO TABLE OF FILE NUMBER VECTORS
1687 1688	CC29 D6 6F CC2B 58		ASLB	DEVNUM	GET CURRENT FILE NUMBER 2 BYTES PER FCB ADDRESS
1689	CC2C AE 85		LDX	В,Х	POINT X TO PROPER FCB
	CC2E E6 84			FCBTYP,X #INPFIL	GET FILE TYPE IS IT AN INPUT FILE?
	CC3Ø C1 1Ø CC32 27 36		BEQ		RETURN IF SO
1693	CC34 81 ØD		CMPA		CARRIAGE RETURN (ENTER)
	CC36 26 Ø2		BNE		NO CLEAR PRINT POSITION IF CARRIAGE RETURN
1695 1696	CC38 6F Ø6 CC3A 81 2Ø	LCC3A		FCBPOS,X #SPACE	*
1697	CC3C 25 Ø2		BLO		*BRANCH IF CONTROL CHAR
1698 1699	CC3E 6C Ø6 CC4Ø C1 4Ø	LCC4Ø		FCBPOS,X #RANFIL	INCREMENT PRINT POSITION IS IT RANDOM FILE?
1700	CC42 26 1A		BNE	LCC5E	BRANCH IF NOT RANDOM
1701	0044 50 00 47	* PUT A		NTO A RANDOM FILE	
1702 1703	CC44 EC 88 17 CC47 C3 ØØ Ø1			FCBPUT, X #\$0001	GET 'PUT' BYTE COUNTER ADD ONE
1704	CC4A 10 A3 09		CMPD	FCBRLN,X	COMPARE TO RECORD LENGTH
1705	CC4D 10 22 01 7A		LBHI		'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH
1706 1707	CC51 ED 88 17 CC54 AE ØB			FCBPUT,X FCBBUF,X	SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER
1708	CC56 3Ø 8B		LEAX	D,X	POINT TO ONE PAST END OF CURRENT RECORD DATA
1709	CC58 35 Ø2		PULS		PULL DATA FROM STACK
1710 1711	CC5A A7 1F CC5C 35 94		STA PULS	-1,X B,X,PC	STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN
1712					
1713 1714	CC5E 6C 88 18	* WRITE LCC5E		TO SEQUENTIAL OUTPUT FILE FCBLFT,X	INCREMENT CHARACTER COUNT
1714	CC61 E6 88 18	LUUJE		FCBLFT,X	* GET CHARACTER COUNT AND BRANCH
1716	CC64 27 Ø6		BEQ	LCC6C	* IF THE BUFFER IS FULL
1717 1718	CC66 3A CC67 A7 88 18		ABX STA	FCBCON-1,X	ADD CHARACTER COUNT TO FCB ADDRESS STORE NEW CHARACTER (SKIP PAST 25 CONTROL BYTES AT FCB START)
1718	CC6A 35 96	LCC6A		A,B,X,PC	STORE HEM CHARACTER (SKIF FAST 25 CONTROL BITES AT FUB STAKT)
1720					
1721 1722	CC6C 34 6Ø	* WRITE LCC6C	OUT A PSHS	FULL BUFFER AND RESET BUFFER	SAVE REGISTERS
1723	CC6E A7 89 Ø1 18	20000		SECLEN+FCBCON-1,X	STORE LAST CHARACTER IN BUFFER
1724	CC72 E6 Ø1		LDB	FCBDRV,X	* GET DRIVE NUMBER AND SAVE
1725 1726	CC74 D7 EB CC76 6C Ø4		STB	DCDRV FCBSEC,X	* IT IN DSKCON CONTROL TABLE INCREMENT SECTOR NUMBER
1727	CC78 BD CB DF		JSR	LCBDF	WRITE THE FILE BUFFER TO DISK
1728	CC7B 31 84		LEAY	,Х	SAVE FCB POINTER IN Y

1729	CC7D E6 Ø3		LDB FCBCGR,X	GET GRANULE NUMBER
1730	CC7F BD C7 55		JSR LC755	POINT X TO PROPER ALLOCATION TABLE
1731	CC82 3A		ABX	ADD THE GRANULE NUMBER TO FAT POINTER
1732	CC83 33 Ø6		LEAU FATCON,X	POINT U TO THE CORRECT GRANULE IN FAT - SKIP PAST
1733				THE SIX FAT CONTROL BYTES
1734	CC85 A6 24		LDA FCBSEC,Y	GET CURRENT SECTOR FOR THIS GRANULE
1735	CC87 81 Ø9		CMPA #\$Ø9	MAX SECTOR NUMBER (9 SECTORS/GRANULE)
1736 1737	CC89 25 ØE CC8B 6A 24		BLO LCC99 DEC FCBSEC,Y	BRANCH IF NOT AT END OF GRANULE *DECREMENT SECTOR NUMBER AND INCREMENT ERROR FLAG IN
1737	CC8D 6C 25		INC FCBCPT,Y	*CASE ERROR FOUND WHILE LOOKING FOR NEXT GRANULE
1739	0000 00 23	*	110 100011,1	THE ERROR FLAG IS USED TO INDICATE THAT ANOTHER SECTOR
1740		*		MUST BE ADDED TO THE LENGTH OF FILE FOLLOWING ERROR PROCESSING.
1741	CC8F BD C7 BF		JSR LC7BF	GET NEXT FREE GRANULE
1742	CC92 6F 24		CLR FCBSEC,Y	*CLEAR SECTOR NUMBER AND
1743	CC94 6F 25		CLR FCBCPT,Y	*ERROR FLAG - DISK WAS NOT FULL
1744	CC96 A7 23		STA FCBCGR, Y	SAVE NEW GRANULE IN FCB
1745	CC98 8C 8A CØ	1.0000	CMPX #\$8ACØ	SKIP TWO BYTES NO DATA STORED IN NEW SECTOR YET
1746 1747	CC99 8A CØ CC9B A7 C4	LCC99	ORA #\$CØ	FORCE GRANULE NUMBER TO BE FINAL GRANULE IN FILE STORE IN MAP
1748	CC9D 3Ø A4		STA ,U LEAX ,Y	POINT X TO FCB
1749	CC9F BD C6 85		JSR LC685	INCREMENT RECORD NUMBER
1750	CCA2 BD C5 A9		JSR LC5A9	UPDATE FILE ALLOCATION TABLE
1751	CCA5 35 6Ø		PULS Y,U	RESTORE REGISTERS
1752	CCA7 35 96		PULS A,B,X,PC	RESTORE REGISTERS AND RETURN
1753				
1754	CC40 DD DO 45	* DIR C		CCAN DRIVE NUMBER FROM INDUT LINE
1755 1756	CCA9 BD D2 4F CCAC BD C7 9D	DIR	JSR LD24F JSR LC79D	SCAN DRIVE NUMBER FROM INPUT LINE GET FAT FOR THIS DRIVE
1757	CCAC BD C7 9D		JSR LB958	PRINT CARRIAGE RETURN TO CONSOLE OUT
1758	CCB2 CC 11 Ø2		LDD #\$1102	* GET TRACK 17 AND
1759	CCB5 97 EC		STA DCTRK	* READ OP CODE AND
1760	CCB7 D7 EA		STB DCOPC	* SAVE IN DSKCON VARIABLES
1761	CCB9 C6 Ø3		LDB #\$Ø3	START WITH SECTOR 3 (FIRST DIRECTORY SECTOR)
1762				
1763	CCDD D7 FC		A DIRECTORY SECTOR INTO THE	
1764	CCBB D7 ED	LCCBB	STB DSEC	SAVE SECTOR NUMBER IN DSKCON VARIABLE
1765 1766	CCBD 8E 06 00 CCC0 9F EE		LDX #DBUFØ STX DCBPT	* USE I/O BUFFER Ø FOR DATA TRANSFER  * SAVE IN DSKCON VARIABLE
1767	CCC2 BD D6 F2		JSR LD6F2	READ A SECTOR
1768	CCCZ DD DO 12		OSK EDOI E	KEAD A SECION
1769		* SEND	DIRECTORY INFORMATION TO CONS	OLE OUT
1770	CCC5 35 4Ø	LCCC5	PULS U	SAVE TOP OF STACK
1771	CCC7 BD A5 49		JSR LA549	GO DO A BREAK CHECK
1772	CCCA 34 4Ø		PSHS U	RESTORE STACK
1773	CCCC A6 84		LDA DIRNAM,X	TEST FILE NAME FIRST BYTE
1774 1775	CCCE 27 38		BEQ LCDØ8	BRANCH IF KILLED
1776	CCDØ 43 CCD1 27 44		COMA BEQ LCD17	FF = END OF DIRECTORY RETURN IF END OF DIRECTORY
1777	CCD3 34 10		PSHS X	SAVE DIRECTORY POINTER ON STACK
1778	CCD5 C6 Ø8		LDB #\$Ø8	NUMBER CHARACTERS TO PRINT
1779	CCD7 BD B9 A2		JSR LB9A2	SEND FILENAME TO CONSOLE OUT
1780	CCDA 8D 3F		BSR LCD1B	SEND BLANK TO CONSOLE OUT
1781	CCDC C6 Ø3		LDB #\$Ø3	NUMBER CHARACTERS TO PRINT
1782	CCDE BD B9 A2		JSR LB9A2	SEND EXTENSION TO CONSOLE OUT
1783	CCE1 8D 38		BSR LCD1B	SEND BLANK TO CONSOLE OUT
1784 1785	CCE3 E6 ØØ		LDB FCBTYP,X CMPB #10	GET FILE TYPE
1786	CCE5 C1 ØA CCE7 24 Ø2		BHS LCCEB	* CHECK THE NUMBER OF DECIMAL DIGITS IN * ACCB: IF THERE IS ONLY ONE DIGIT,
1787	CCE9 8D 3Ø		BSR LCD1B	* SEND BLANK TO CONSOLE OUT
1788	CCEB 4F	LCCEB	CLRA	CLEAR MS BYTE OF ACCO
1789	CCEC BD BD CC		JSR LBDCC	PRINT ACCD IN DECIMAL TO CONSOLE OUT
1790	CCEF 8D 2A		BSR LCD1B	SEND BLANK TO CONSOLE OUT
1791	CCF1 AE E4		LDX ,S	X NOW POINTS TO DIRECTORY ENTRY
1792	CCF3 86 42		LDA #'A'+1	ASCII BIAS
1793 1794	CCF5 AB ØC		ADDA DIRASC,X	ADD TO ASCII FLAG PRINT CHARACTER AND BLANK TO CONSOLE OUT
1794	CCF7 8D 1F CCF9 E6 ØD		BSR LCD18 LDB DIRGRN,X	GET FIRST GRANULE IN FILE
1796	CCFB 8D 21		BSR LCD1E	COUNT GRANULES
1797	CCFD 1F 89		TFR A,B	SAVE COUNT IN ACCB
1798	CCFF 4F		CLRA	CLEAR MS BYTE OF ACCD
1799	CDØØ BD BD CC		JSR LBDCC	PRINT ACCO IN DECIMAL TO CONSOLE OUT
1800	CDØ3 BD B9 58		JSR LB958	SEND CARRIAGE RETURN TO CONSOLE OUT
1801	CDØ6 35 1Ø CDØ8 3Ø 88 2Ø	Lengo	PULS X	PULL DIRECTORY POINTER OFF OF THE STACK
1802 1803	CD08 30 88 20 CD0B 8C 07 00	LCDØ8	LEAX DIRLEN,X CMPX #DBUFØ+SECLEN	MOVE X TO NEXT DIRECTORY ENTRY END OF I/O BUFFER?
1804	CDØE 25 B5		BLO LCCC5	BRANCH IF MORE DIRECTORY ENTRIES IN BUFFER
1805	CD10 D6 ED		LDB DSEC	GET CURRENT SECTOR
1806	CD12 5C		INCB	BUMP COUNT
1807	CD13 C1 12		CMPB #SECMAX	SECMAX SECTORS IN DIRECTORY TRACK
1808	CD15 23 A4		BLS LCCBB	GET NEXT SECTOR
1809	CD17 39	LCD17	RTS	FINISHED
1810	CD18 BD A2 82	LCD18 LCD1B	JSR LA282 JMP LB9AC	SEND CHARACTER TO CONSOLE OUT SEND BLANK TO CONSOLE OUT
1811 1812	CD1B 7E B9 AC	FCDIR	OHF EDDAL	SEMP DEWAY IN COMPONE AMI
1813		* ENTER	WITH ACCB POINTING TO FIRST	GRANULE IN A FILE; RETURN THE NUMBER OF
1814				GRANULE DATA FOR THE LAST SECTOR IN ACCB
	CD1E BD C7 55	LCD1E	JSR LC755	POINT X TO FILE ALLOCATION BUFFER
1816			LEAU FATCON,X	POINT U TO START OF GRANULE DATA
1817	CD23 4F		CLRA	RESET GRANULE COUNTER
	CD24 4C	LCD24	INCA	INCREMENT GRANULE COUNTER
1819			CMPA #GRANMX	CHECKED ALL 68 GRANULES?
1820 1821	CD27 10 22 F9 28 CD2B 30 C4		LBHI LC653 LEAX ,U	YES – 'BAD FILE STRUCTURE' ERROR POINT U TO START OF GRANULE DATA
1822	CD2D 3A		ABX	ADD POINTER TO FIRST GRANULE
1823	CD2E E6 84		LDB ,X	GET THIS GRANULE'S CONTROL BYTE
1824	CD3Ø C1 CØ		CMPB #\$CØ	IS THIS THE LAST GRANULE IN FILE?

1005	0000 05 50			10004	NO. 1/550 003NO
1825 1826	CD32 25 FØ CD34 39		BLO RTS	LCD24	NO - KEEP GOING
1827 1828 1829 1830 1831	CD35 ØD 6F CD37 2F 5E CD39 8E BØ 69	* INPUT DVEC10	RAM H TST BLE LDX	OOK DEVNUM LCD97 #LBØ69	* CHECK DEVICE NUMBER AND RETURN * IF NOT A DISK FILE = CHANGE THE RETURN ADDRESS ON THE STACK TO RE-ENTER BASIC'S INPUT
1832 1833 1834 1835	CD3C AF E4 CD3E 8E Ø2 DD CD41 C6 2C CD43 D7 Ø1		STX LDX	#LINBUF+1 #',' CHARAC	= ROUTINE AT A DIFFERENT PLACE THAN THE CALLING ROUTINE POINT X TO THE LINE INPUT BUFFER = COMMA IS READ ITEM SEPARATOR (TEMPORARY STRING SEARCH FLAG)
1836 1837 1838 1839	CD45 96 Ø6 CD47 26 Ø2 CD49 C6 2Ø CD4B 8D 6F	LCD4B	LDA BNE LDB BSR	VALTYP LCD4B #SPACE LCDBC	* GET VARIABLE TYPE AND BRANCH IF * IT IS A STRING SPACE = NUMERIC SEARCH DELIMITER GET AN INPUT CHARACTER
1840 1841 1842 1843	CD4D 81 2Ø CD4F 27 FA CD51 81 22 CD53 26 ØA	20040	CMPA BEQ CMPA	#SPACE LCD4B #'"' LCD5F	SPACE? YES - GET ANOTHER CHARACTER QUOTE? NO
1844 1845 1846 1847	CD55 C1 2C CD57 26 Ø6 CD59 1F 89 CD5B D7 Ø1		CMPB	#',' LCD5F	SEARCH CHARACTER = COMMA? NO - NUMERIC SEARCH * SAVE DOUBLE QUOTE AS * THE SEARCH FLAG
1848 1849	CD5D 20 22		BRA	LCD81	SAVE DOUBLE QUOTES AS FIRST ITEM IN BUFFER
1850 1851 1852 1853	CD5F C1 22 CD61 27 11 CD63 81 ØD CD65 26 ØD	LCD5F		#'"' LCD74 #CR LCD74	* *BRANCH IF INPUTTING A STRING VARIABLE IS THE INPUT CHARACTER A CARRIAGE RETURN NO
1854 1855 1856 1857 1858	CD67 8C Ø2 DD CD6A 27 44 CD6C A6 1F CD6E 81 ØA CD7Ø 26 3E		BEQ LDA CMPA		*IF AT THE START OF IMPUTBUFFER, CHECK FOR A *FOLLOWING LINE FEED AND EXIT ROUTINE =IF THE INPUT CHARACTER PRECEEDING THE CR WAS A LINE FEED, =THEN INSERT THE CR IN THE INPUT STRING, OTHERWISE
1859 1860 1861 1862	CD72 86 ØD CD74 4D CD75 27 17 CD77 91 Ø1	LCD74	LDA TSTA BEQ	LCDBØ #CR LCD8E CHARAC	=CHECK FOR A FOLLOWING LINE FEED AND EXIT THE ROUTINE RESTORE CARRIAGE RETURN AS THE INPUT CHARACTER *CHECK FOR A NULL (ZERO) INPUT CHARACTER AND *IGNORE IT IF IT IS A NULL =
1863 1864 1865 1866	CD79 27 1D CD7B 34 Ø4 CD7D A1 EØ CD7F 27 17			LCD98	=CHECK TO SEE IF THE INPUT CHARACTER MATCHES =EITHER ACCB OR CHARAC AND IF IT DOES, THEN =BRANCH TO CHECK FOR ITEM SEPARATOR OR =TERMINATOR SEQUENCE AND EXIT ROUTINE
1867 1868 1869 1870	CD81 A7 80 CD83 8C 03 D6 CD86 26 06 CD88 8D 46	LCD81	STA CMPX BNE BSR	,X+ #LINBUF+LBUFMX LCD8E LCDDØ	STORE NEW CHARACTER IN BUFFER END OF INPUT BUFFER NO GET A CHARACTER FROM CONSOLE IN
1871 1872 1873 1874	CD8A 26 Ø6 CD8C 2Ø 1E CD8E 8D 4Ø	LCD8E	BNE BRA BSR	LCD92 LCDAC	EXIT ROUTINE IF BUFFER EMPTY CHECK FOR CR OR CR/LF AND EXIT ROUTINE  GET A CHARACTER FROM CONSOLE IN
1875 1876 1877	CD9Ø 27 CD CD92 6F 84 CD94 8E Ø2 DC	LCD92	BEQ CLR LDX	LCD5F ,X #LINBUF	BRANCH IF BUFFER NOT EMPTY PUT A ZERO AT END OF BUFFER WHEN DONE POINT (X) TO LINBUF - RESET POINTER
1878 1879 1880	CD97 39	LCD97 * CHECK	RTS FOR I	TEM SEPARATOR OR TERMINATOR AND E	XIT THE INPUT ROUTINE
1881 1882	CD98 81 22 CD9A 27 Ø4	LCD98	CMPA BEQ	#'"' LCDAØ	QUOTE? YES
1883 1884 1885 1886 1887	CD9C 81 20 CD9E 26 F2 CDAØ 8D 2E CDA2 26 EE CDA4 81 20		BNE BSR BNE	#SPACE LCD92 LCDD0 LCD92 #SPACE	SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE?
1888 1889 1890 1891	CDA6 27 F8 CDA8 81 2C CDAA 27 E6 CDAC 81 ØD	LCDAC	BEQ CMPA BEQ	#',' LCD92 #CR	VES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN?
1892 1893 1894 1895	CDAE 26 Ø8 CDBØ 8D 1E CDB2 26 DE CDB4 81 ØA		BSR BNE CMPA	LCDB8 LCDDØ LCD92 #LF	NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR,LF AS A CR
1896 1897 1898 1899	CDB6 27 DA CDB8 8D 1C CDBA 20 D6	LCDB8	BSR	LCD92 LCDD6 LCD92	YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE
1900 1901 1902 1903	CDBC 8D 12 CDBE 27 15 CDCØ BD C7 44 CDC3 E6 ØØ	LCDBC	JSR LDB	LCDD5 LC744 FCBTYP,X	GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE
1904 1905 1906 1907 1908	CDC5 C1 40 CDC7 10 26 F5 87 CDCB C6 4A CDCD 7E AC 46	LCDCB	LBNE LDB	#RANFIL LC352 #2*37 LAC46	IS IT RANDOM FILE TYPE?  'INPUT PAST END OF FILE ERROR IF NOT RANDOM  'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM  JUMP TO THE ERROR HANDLER
1908 1909 1910 1911 1912	CDDØ BD A1 76 CDD3 ØD 7Ø CDD5 39	LCDDØ		LA176 CINBFL	GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG
1913 1914 1915 1916 1917 1918 1919	CDD6 34 14 CDD8 BD C7 44 CDD8 E6 00 CDDD C1 40 CDDF 26 0B CDEI EC 88 15 CDE4 83 00 01	* MOVE T	PSHS JSR LDB CMPB BNE LDD	PUT POINTER BACK ONE (DISK FILE) X,B LC744 FCBTYP,X #RANFIL LCDEC FCBGET,X #\$0001	SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE *GRAB THE RANDOM FILE 'GET' POINTER, *MOVE IT BACK ONE AND RESTORE IT

1921 1922	CDE7 ED 88 15 CDEA 35 94	LCDEC	STD PULS	FCBGET,X B,X,PC FCBCDT,X FCBCFL,X B,X,PC	* RESTORE REGISTERS AND RETURN
1923 1924	CDEC A7 88 11 CDEF 63 88 10	LCDEC	STA	FCBCDT, X	SAVE THE CHARACTER IN THE CACHE
	CDF2 35 94		PULS	B,X,PC	SET THE CACHE FLAG TO \$FF - DATA IN CACHE RESTORE REGISTERS AND RETURN
1926 1927		* CVN CC	MMAND		
		CVN		LB654	GET LENGTH AND ADDRESS OF STRING
	CDF7 C1 Ø5 CDF9 1Ø 25 E6 4D		CMPB		FIVE BYTES IN A FLOATING POINT NUMBER 'FC' ERROR IF ⇔ 5 BYTES
1931	CDFD ØF Ø6		CLR	VALTYP	SET VARIABLE TYPE TO NUMERIC
1932 1933	CDFF 7E BC 14		JMP	LBC14	COPY A PACKED FP NUMBER FROM (X) TO FPAØ
1934		* MKN\$ C			
	CEØ2 BD B1 43 CEØ5 C6 Ø5	MKN	JSR I DR		'TM' ERROR IF VALTYP=STRING FIVE BYTES IN A FLOATING POINT NUMBER
1937	CEØ7 BD B5 ØF		JSR	LB5ØF	RESERVE FIVE BYTES IN STRING SPACE
	CEØA BD BC 35 CEØD 7E B6 9B		JSR JMP		PACK FPAØ AND STORE IT IN STRING SPACE SAVE STRING DESCRIPTOR ON STRING STACK
1940					
1941 1942	CE10 8D 07	* LOC CO LOC LCE14	BSR	LCE19	POINT X TO FILE BUFFER
	CE12 EC Ø7		LDD	FCBREC, X	GET RECORD NUMBER (RANDOM FILE) OR SECTOR CTR (SEQUENTIAL)
	CE14 DD 52 CE16 7E 88 ØE	LUE14	JMP	L88ØE	*SAVE ACCD IN BOTTOM 2 BYTES OF FPAØ AND *CONVERT TO FLOATING POINT NUMBER
1946 1947		* CTDID	A DEVI	CE NUMBER FROM A BASIC STATEMENT,	CET DDINT
1948		* PARAME	TERS A	ACCORDING TO IT - ERROR IF FILE NO	T
1949 1950	CF19 96 6F			WITH (X) POINTING TO THAT FILE'S DEVNUM	FCB * GET CURRENT DEVICE NUMBER AND
1951					
1952 1953	CE1D BD B1 43 CE20 BD A5 AE		JSR JSR	LB143 LA5AF	'TM' ERROR IF VALTYP=STRING CHECK FOR VALID DEVICE NUMBER/SET PRINT PARAMETERS
1954	CE23 ØD 6F		TST	DEVNUM	* CHECK DEVICE NUMBER
1955 1956	CE25 10 2F E6 21 CE29 BD C7 44		JSR		* BRANCH IF NOT DISK FILE 'ILLEGAL FUNCTION CALL' POINT (X) TO FILE BUFFER
1957	CE2C 35 Ø2		PULS	Α	* GET OLD DEVICE NUMBER OFF OF THE STACK AND
1958	CE2E 97 6F CE3Ø 6D ØØ		TST	FCBTYP,X	* SAVE IT AS DEVICE NUMBER IS FILE OPEN?
1960	CE32 10 27 D5 C5		LBEQ	LA3FB	'FILE NOT OPEN' ERROR IF NOT OPEN
1962	CE36 39				
1963		* LOF LOF	pen	LCE19 FCBDRV,X DCDRV FCBFGR,X X LCD1E #\$3F B A,B LC779	POINT X TO FILE BUFFER
1965	CE39 A6 Ø1	LUF	LDA	FCBDRV, X	* GET DRIVE NUMBER AND SAVE IT
	CE3B 97 EB		STA	DCDRV ECRECP Y	* IN DSKCON VARIABLE GET FIRST GRANULE OF FILE
1968	CE3D E6 Ø2 CE3F 34 1Ø		PSHS	X	SAVE FCB POINTER ON STACK
1969 1970	CE41 BD CD 1E CE44 4A		JSR	LCD1E	FIND TOTAL NUMBER OF GRANULES IN THIS FILE SUBTRACT THE LAST GRANULE IN THE FILE
1971	CE45 C4 3F		ANDB	#\$3F	GET NUMBER OF SECTORS USED IN LAST GRANULE
1972	CE47 34 Ø4 CE49 1E 89		PSHS	B A R	SAVE NUMBER OF SECTORS IN LAST GRANULE ON STACK * CONVERT ACCA TO POSITIVE
1974	CE49 1F 89 CE4B 4F		CLRA		* 2 BYTE VALUE IN ACCD
15/3	CE4C BD C7 79 CE4F EB EØ		JSR ADDB	LC779 .S+	MULT NUMBER OF FULL GRANULES BY 9 ADD NUMBER SECTORS IN LAST TRACK
1977	CE51 89 ØØ		ADCA	#\$00	PROPAGATE CARRY TO MS BYTE OF ACCD
1978 1979	CE53 35 10 CE55 34 02				GET FCB POINTER BACK SAVE ACCA ON STACK
1980	CE57 A6 ØØ		LDA	FCBTYP,X	* GET FILE TYPE OF THIS FCB AND
1981	CE53 35 10 CE55 34 02 CE57 A6 00 CE59 81 40 CE58 35 02		PULS	#RANFIL A	* CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA
1983 1984	CE5D 26 B5	*	BNE		IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE IS THE LENGTH OF THE FILE
1985					
1986 1987				OF FOR A RANDOM FILE - THE LENGTH ( ECORDS IN THE FILE.	OF A RANDOM FILE IS THE
1988	CE5F 34 10		PSHS	X	SAVE FCB POINTER ON STACK
	CE61 93 8A CE63 27 Ø3		SUBD BEQ		SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS
1991 1992	CE65 83 ØØ Ø1 CE68 8D AA	LCE68			SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE IOOZ USED PUT ACCD INTO FPAØ
	CE6A D6 4F	LUEUO	LDB	FPØEXP	GET EXPONENT OF FPAØ
	CE6C 27 Ø4 CE6E CB Ø8		BEQ ADDB		BRANCH IF FPAØ = Ø * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY
1996	CE7Ø D7 4F		STB	FPØEXP	* 256 BYTES/SECTOR) AND SAVE NEW EXPONENT
	CE72 BD BC 5F CE75 AE E4	LCE72	JSR LDX		SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB
1999	CE77 EC 88 13		LDD	FCBLST,X	GET NUMBER OF BYTES IN LAST SECTOR
	CE7A 84 7F CE7C 8D 96		ANDA BSR		MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ
2002	CE7E ØF 62		CLR	RESSGN	FORCE SUM SIGN = POSITIVE
	CE8Ø 96 5C CE82 D6 4F				* GET EXPONENTS OF FPAØ AND * FPA1 PRIOR TO ADDITION
2005	CE84 BD B9 C5			LB9C5	ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF
2006 2007	CE87 BD BC 5F		JSR		BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1
2008	CE8A 35 10		PULS	X	POINT X TO FCB
	CE8C EC Ø9 CE8E 8D 84				* GET RECORD LENGTH * PUT IT INTO FPAØ
2011	CE9Ø ØF 62		CLR	RESSGN	FORCE QUOTIENT SIGN = POSITIVE
	CE92 96 5C CE94 D6 4F				* GET EXPONENTS OF FPAØ AND * FPA1 PRIOR TO DIVISION
2014 2015	CE96 BD BB 91 CE99 7E BC EE			LBB91	DIVIDE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER
2016	OLJS /L DU EE		UNF	101	CONTENT TIME TO AN INTEGER

2017		* FREE COM		
	CE9C BD B1 43 CE9F BD B7 ØE		SR LB143 SR LB7ØE	* NUMBER TYPE CHECK  *EVALUATE NUMERIC EXPRESSION AND RETURN VALUE IN ACCB
	CEA2 C1 Ø3		MPB #\$03	ONLY 4 LEGAL DRIVES
	CEA4 10 22 D7 77	1.0	DUT IAG1E	'DEVICE NUMBER' ERROR IF DRIVE NUMBER IS > 3
	CEA8 D7 EB	ST	SR LC79D SR LC755 SR LC755 LC755 LR ,-S	SAVE IN DRIVE NUMBER
	CEAA BD C7 9D CEAD BD C7 55	J.	SR LC/9D SD LC755	GET FILE ALLOCATION TABLE AND STORE IN BUFFER POINT X TO START OF FILE ALLOCATION TABLE BUFFER
	CEBØ 3Ø Ø6	LI	EAX FATCON.X	MOVE TO FIRST GRANULE DATA BYTE
	CEB2 6F E2	CI	LR ,-S	SPACE FOR FREE GRANULE COUNTER
	CEB4 C6 44	LI	DB #GRANMX	GET MAXIMUM NUMBER OF GRANULES
	CEB6 A6 8Ø CEB8 43	LCEB6 LI	DA ,X+ DMA	GET GRANULE DATA *FREE GRANULES \$FF
	CEB9 26 Ø2		NE LCEBD	*BRANCH IF NOT FREE
	CEBB 6C E4		NC ,S	INCREMENT FREE GRANULE COUNTER
	CEBD 5A		ECB	DECREMENT GRANULE COUNTER
	CEBE 26 F6 CECØ 35 Ø4		NE LCEB6	BRANCH IF NOT DONE GET FREE GRANULE COUNTER TO ACCB
2034	CEC2 7E B4 F3		ULS B MP LB4F3	LOAD ACCB INTO FPAØ
2036				
2037	0505 00 07 00	* DRIVE CO		EVALUATE EVAN DETUNN VALUE IN ACCO
	CEC5 BD B7 ØB CEC8 C1 Ø3		SR EVALEXPB MPB #\$Ø3	EVALUATE EXPR; RETURN VALUE IN ACCB MAX DRIVE NUMBER = 3
	CECA 10 22 D7 51		BHI LA61F	'DEVICE #' ERROR IF DRIVE NUMBER > 3
	CECE F7 Ø9 5A		TB DEFDRV	SAVE DEFAULT DRIVE NUMBER
	CED1 39	R	TS	
2043 2044		* EVALUATE	EXPRESSION RAM VECTOR	
	CED2 A6 64	DVEC15 LI		= CHECK STACKED PRECEDENCE FLAG AND IF IT IS NOT AN END
	CED4 26 13		NE LCEÉ9	= OF OPERATION, BRANCH TO EXTENDED BASIC'S EXPRESSION
2047	0506 45 55		DV 495 0	= EVALUATION ROUTINE
	CED6 AE 65 CED8 8C AF 9A		DX \$05,S MPX #LAF9A	* *
	CEDB 26 ØC		NE LCEE9	* CHECK TWO RETURN ADDRESSES BACK ON THE STACK
	CEDD AE 62		DX \$02,S	* TO SEE IF THE CALL TO EVALUATE EXPRESSION IS
2052	CEDF 8C B1 66	CI	MPX #LB166	* COMING FROM THE 'LET' COMMAND - BRANCH OUT IF
	CEE2 26 Ø5		NE LCEE9	* NOT COMING FROM 'LET'
	CEE4 8E CE EC CEE7 AF 65		DX #LCEEC TX \$05,S	= IF COMING FROM 'LET', REPLACE THE RETURN ADDR = WITH THE DISK BASIC 'LET' MODIFIER ADDRESS
	CEE9 7E 88 46	LCEE9 JN		EXTENDED BASIC EXPRESSION EVALUATION
2057				
2Ø58 2Ø59	CEEC 35 Ø2	* LET MODI	FIER ULS A	PULL VARIABLE TYPE OFF OF THE STACK
2060	CEEE 46		ORA	SET CARRY IF SIRING, CLEAR CARRY IF NUMERIC
2061	CEEF BD B1 48	J:	SR LB148	DO A 'TM' CHECK
2062	CEF2 10 27 ED 3D	LE	BEQ LBC33 DX FPAØ+2 DD \$02,X MPD #DFLBUF	IF NUMERIC VARIABLE, PACK FPAØ INTO VARDES
	CEF6 9E 52	LI	DX FPAØ+2	POINT X TO STRING DESCRIPTOR
	CEF8 EC Ø2 CEFA 1Ø 83 Ø9 89	CI	MPD #DFIRIF	GET ADDRESS OF SIRING * COMPARE TO START OF RANDOM FILE BUFFERS
	CEFE 25 Ø7	BI	LO LCFØ7	* AND BRANCH IF LOWER
2067	CFØØ B3 Ø9 4A		UBD FCBADR	SUBTRACT OUT THE END OF RANDOM FILE BUFFERS
2068 2069	CFØ3 1Ø 25 EØ AA	LI	BCS LAFB1	BRANCH IF STRING STORED IN RANDOM FILE BUFFER - MOVE IT INTO THE STRING SPACE
	CFØ7 7E AF A4	LCFØ7 JN	MP LAFA4	BRANCH BACK TO BASIC S 'LET' COMMAND
2071				
2072 2073	CFØA 81 CA	*MODIFIER DXCVEC CN	FOR EXBAS COMMAND INTERPRETATION HAND	DLER TOKEN FOR DLOAD?
	CFØC 27 1C		EO LCF2A	YES
	CFØE 81 C8		MPA #\$C8	TOKEN FOR PMODE?
	CF10 10 26 B2 28		BNE L813C	NO
2077	CE14 OD OE		IC MODIFIER FOR PMODE - ALLOWS FOR TH	
	CF14 9D 9F CF16 81 2C		SR GETNCH MPA #','	GET NEXT CHARACTER FROM BASIC CHECK FOR COMMA
2080	CF18 10 27 C7 34		BEQ L965Ø	BRANCH IF COMMA
2081	CF1C BD B7 ØB	J	SR EVALEXPB	EVALUATE EXPRESSION; RETURN VALUE IN ACCB
	CF1F C1 Ø4		MPB #\$Ø4	CHECK FOR PMODE 4 'FC' ERROR IF PMODE > 4
	CF21 10 22 E5 25 CF25 96 BC		BHI LB44A DA GRPRAM	NUMBER BLOCKS BEFORE GRAPHICS PAGES
2085	CF27 7E 96 2E		MP L962E	JUMP TO EXEAS' PMODE COMMAND
2086				
2087	CF2A BD A4 29	* DISK BAS LCF2A JS	IC DLOAD MODIFIER	CLOSE FILES
	CF2A BD A4 29 CF2D 9D 9F		SR GETNCH	GET NEXT CHARACTER FROM BASIC
	CF2F 7E 8C 1B		MP L8C1B	JUMP TO EXEAS' DLOAD
2091				
2092	CF32 C1 34	DXIVEC C	MPB #(\$9A-\$8Ø)*2	MODIFIED TOKEN FOR POS
2033	CF34 1Ø 26 B2 3Ø CF38 BD B2 62	LI	BNE L8168 SR LB262	IF NOT POS, GO TO EXBAS SECONDARY COMM HANDLER SYNTAX CHECK FOR '(' AND EVALUATE EXPRESSION
	CF3B 96 6F		DA DEVNUM	* GET DEVICE NUMBER AND
2096	CF3D 34 Ø2		SHS A	* SAVE IT ON STACK
2097	CF3F BD A5 AE CF42 BD A4 Ø6	JS	SR LASAE	EVALUATE DEVICE NUMBER
20199	CF45 ØD 6F	J:	SR LA406 ST DEVNUM LE LCF5C SR LC744	TEST DEVICE NUMBER * CHECK DEVICE NUMBER AND BRANCH
2100	CF47 2F 13	BI	LE LCF5C	* IF NOT A DISK FILE
2101	CF49 BD C7 44	JS	SR LC744	POINT X TO FCB
2102	CF4C E6 ØØ	LI	DB FCBTYP,X	GET FILE TYPE
2103	CF50 26 04	CI Di	MF LCESC	DIRECT/RANDOM FILE? BRANCH IF NOT A RANDOM FILE
2105	CF52 35 Ø2	Pl	ULS A	* RESTORE DEVICE NUMBER
2106	CF47 2F 13 CF49 BD C7 44 CF4C E6 00 CF4E C1 40 CF50 26 0A CF52 35 02 CF54 97 6F	ST	TA DEVNUM	*
2107	0130 EC 00 17	LI	DD 160101, A	=GRAB THE 'PUT' DATA ITEM COUNTER AND CONVERT
	CF59 7E B4 F4 CF5C BD A3 5F	LCF5C JS	MP GIVABF SR LA35F	=IT TO A FLOATING POINT NUMBER SET PRINT PARAMETERS
2110	CF5F 35 Ø2		ULS A	* RESTORE DEVICE NUMBER
2111	CF61 97 6F	ST	TA DEVNUM	*
2112	CF63 D6 6C	L[	DB DEVPOS	=GET PRINT POSITION AND

2113	CF65 7E B4 F3	JMP LB4F3	=CONVERT IT TO FLOATING POINT NUMBER IN FPAØ
2114 2115		* SAVEM COMMAND	
2116	CF68 9D 9F	LCF568 JSR GETNCH	GET NEXT INPUT CHARACTER
2117 2118	CF6A 8D 4F CF6C BD 83 6C	BSR LCFBB JSR L836C	GET FILENAME, ETC. EVALUATE EXPRESSION, PUT II (2 BYTES) ON STACK
2119	CF6F BD 83 6C	JSR L836C	DITTO
2120	CF72 AC 62	CMPX \$02,S	COMPARE END ADDRESS TO START ADDRESS
2121 2122	CF74 10 25 E4 D2 CF78 BD 83 6C	LBCS LB44A JSR L836C	IF START > END, THEN 'ILLEGAL FUNCTION CALL' EVAL EXPRESSION (TRANSFER ADDRESS), PUT ON STACK
2123	CF7B BD A5 C7	JSR LA5C7	SYNTAX ERROR IF ANY MORE CHARS ON THIS LINE
2124 2125	CF7E CC 02 00 CF81 FD 09 57	LDD #\$0200 STD DFLTYP	* FILE TYPE=2, ASCII FLAG = CRUNCHED (Ø) *
	CF84 BD CA Ø4	JSR LCAØ4	GET NEXT UNOPEN FILE AND INITIALIZE FCB
	CF87 4F	CLRA BSB LCEBE	*ZERO FLAG - FIRST BYTE OF PREAMBLE
2128 2129	CF88 8D 2B CF8A EC 62	BSR LCFB5 LDD \$02,S	*WRITE A BYTE TO BUFFER GET END ADDRESS
2130	CF8C A3 64	SUBD \$04,S	SUBTRACT THE START ADDRESS
2131 2132	CF8E C3 00 01 CF91 1F 02	ADDD #\$0001 TFR D,Y	THE SAVED DATA BLOCK WILL INCLUDE BOTH THE FIRST AND LAST BYTES SAVE LENGTH IN Y
2133	CF93 8D 1E	BSR LCFB3	WRITE FILE LENGTH TO BUFFER - FIRST ARGUMENT OF PREAMBLE
2134 2135	CF95 EC 64 CF97 8D 1A	LDD \$04,S BSR LCFB3	GET THE START ADDRESS WRITE OUT THE START ADDRESS - SECOND PREAMBLE ARGUMENT
	CF99 AE 64	LDX \$04,S	GET START ADDRESS
2137	CF9B A6 8Ø	LCF9B LDA ,X+	GRAB A BYTE
2138 2139	CF9D BD CC 24 CFAØ 31 3F	JSR LCC24 LEAY -1,Y	WRITE IT OUT DECREMENT BYTE COUNTER
2140	CFA2 26 F7	BNE LCF9B	BRANCH IF ALL BYTES NOT DONE
2141 2142	CFA4 86 FF CFA6 8D ØD	LDA #\$FF BSR LCFB5	FIRST BYTE OF POSTAMBLE WRITE IT OUT - EOF RECORD
2143	CFA8 4F	CLRA	* FIRST ARGUMENT OF POSTAMBLE IS
2144 2145	CFA9 5F CFAA 8D Ø7	CLRB BSR LCFB3	* A DUMMY - ZERO VALUE WRITE OUT POSTAMBLE FIRST ARGUMENT
2146	CFAC 35 36	PULS A,B,X,Y	GET CONTROL ADDRESSES FROM THE STACK
2147	CFAE 8D Ø3	BSR LCFB3	WRITE OUT THE TRANSFER ADDRESS - 2ND ARGUMENT
2148 2149	CFBØ 7E A4 2D	JMP LA42D	GO CLOSE ALL FILES
2150		* WRITE ACCD TO THE BUFFER	
	CFB3 8D 00 CFB5 BD CC 24	LCFB3 BSR LCFB5 LCFB5 JSR LCC24	WRITE ACCA TO BUFFER, THEN SWAP ACCA,ACCB WRITE ACCA TO BUFFER
2153	CFB8 1E 89	EXG A,B	SWAP ACCA, ACCB
2154 2155	CFBA 39 CFBB 8E C2 AF	RTS LCFBB LDX #BINEXT	POINT TO .BIN EXTENSION
2156	CFBE 7E C9 38	JMP LC938	GET FILENAME, ETC.
2157 2158		* LOADM COMMAND	
2159	CFC1 9D 9F	LCFC1 JSR GETNCH	GET NEXT INPUT CHARACTER
2160	CFC3 8D F6	BSR LCFBB	GET FILENAME, ETC.
2161 2162	CFC5 BD CA Ø7 CFC8 FC Ø9 57	JSR LCAØ7 LDD DFLTYP	OPEN NEXT AVAILABLE FILE FOR INPUT GET FILE TYPE AND ASCII FLAG
2163	CFCB 83 Ø2 ØØ	SUBD #\$0200	FOR LOADM FILE: TYPE=2, ASCII FLAG=0
2164 2165	CFCE 10 26 D6 44 CFD2 9E 8A	LBNE LA616 LDX ZERO	'BAD FILE MODE' ERROR ZERO OUT X REG - DEFAULT VALUE OF OFFSET
2166	CFD4 9D A5	JSR GETCCH	GET CURRENT CHARACTER FROM BASIC
2167 2168	CFD6 27 Ø6 CFD8 BD B2 6D	BEQ LCFDE JSR SYNCOMMA	BRANCH IF END OF LINE - NO OFFSET SYNTAX CHECK FOR COMMA
2169	CFDB BD B7 3D	JSR LB73D	EVALUATE EXPRESSION
217Ø 2171	CFDE 9F D3 CFEØ BD A5 C7	LCFDE STX VD3 JSR LA5C7	STORE OFFSET IN VD3 SYNTAX ERROR IF OTHER CHARACTERS ON LINE
2172	CFEW BD AS C7	USK EASC/	SINIAN ERROR IF DIHER CHARACTERS ON LINE
2173 2174	CFE3 BD CD BC	* GET PREAMBLE/POSTAMBLE LCFE3 JSR LCDBC	GET FIRST BYTE
2175	CFE6 34 Ø2	PSHS A	SAVE IT ON THE STACK
2176 2177	CFE8 8D 29 CFEA 1F Ø2	BSR LDØ13 TFR D,Y	GET FIRST ARGUMENT SAVE IT IN Y
2178	CFEC 8D 25	BSR LDØ13	GET THE SECOND ARGUMENT
2179	CFEE D3 D3 CFFØ DD 9D	ADDD VD3	ADD IT TO THE OFFSET
218Ø 2181	CFF2 1F Ø1	STD EXECJP TFR D,X	STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X
2182	CFF4 A6 EØ	LDA ,S+	GET THE FIRST BYTE OFF OF THE STACK
2183 2184	CFF6 10 26 D4 33	LBNE LA42D	CLOSE FILE IF POSTAMBLE (EOF)
2185		* GET RECORD BYTE(S)	
2186 2187	CFFA BD C5 C4 CFFD D6 70	LCFFA JSR LC5C4 LDB CINBFL	GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER
2188	CFFF 27 Ø3	BEQ LDØØ4	BRANCH IF BUFFER NOT EMPTY
2189 2190	DØØ1 7E C3 52 DØØ4 A7 84	JMP LC352 LD004 STA ,X	'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY
2190	D004 A7 84 D006 A1 80	LD004 STA ,X CMPA ,X+	*TEST TO SEE IF IT STORED PROPERLY AND
2192	DØØ8 27 Ø3	BEQ LDØØD	*BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM)
2193 2194	DØØA 7E D7 Ø9 DØØD 31 3F	JMP LD709 LD00D LEAY -1,Y	'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT
2195	DØØF 26 E9	BNE LCFFA	GET NEXT BYTE IF NOT DONE
2196 2197	DØ11 20 DØ	BRA LCFE3 * READ TWO BYTES FROM BUFFER - RETURN THEM I	READ ANOTHER PRE/POST AMBLE N ACCD
2198	DØ13 8D ØØ	LDØ13 BSR LDØ15	READ A BYTE, SAVE IT IN ACCB
2199 2200	DØ15 BD CD BC DØ18 1E 89	LDØ15 JSR LCDBC EXG A,B	GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA.ACCB
2201	DØ1A 39	RTS	omn noon, noon
22Ø2 22Ø3		* RENAME COMMAND	
2204	DØ1B 9E A6	RENAME LDX CHARAD	* SAVE CURRENT INPUT POINTER
22Ø5 22Ø6	DØ1D 34 1Ø DØ1F 8D 35	PSHS X BSR LDØ56	* ON THE STACK GET FILENAME OF SOURCE FILE
2207	DØ21 96 EB	LDA DCDRV	* SAVE DRIVE NUMBER
2208	DØ23 34 Ø2	PSHS A	* ON THE STACK

2209	DØ25 8D 2A		BSR	I DØE 1	SYNTAX CHECK FOR 'TO' AND GET NEW FILENAME
2210	DØ27 35 Ø2		PULS		GET SOURCE DRIVE NUMBER
2211	DØ29 91 EB			DCDRV	COMPARE TO NEW FILE DRIVE NUMBER
2212	DØ2B 1Ø 26 E4 1B			LB44A	'FC' ERROR IF FILES ON DIFFERENT DRIVES
2213	DØ2F 8D 28		BSR	LDØ59	VERIFY THAT NEW FILE DOES NOT ALREADY EXIST
2214	DØ31 35 1Ø		PULS	X	* RESTORE INPUT POINTER
2215	DØ33 9F A6			CHARAD	*
2216	DØ35 8D 1F		BSR	LDØ56	GET SOURCE FILENAME AGAIN
2217	DØ37 BD C6 8C		JSR	LC68C	SCAN DIRECTORY FOR SOURCE FILENAME
2218	DØ3A BD C6 E5		JSR	LC6E5	'NE' ERROR IF NOT FOUND
2219 2220	DØ3D 8D 12		R2K	#DNAMBF	SYNTAX CHECK FOR 'TO' AND GET NEW FILENAME POINT X TO FILENAME
2220	DØ3F 8E Ø9 4C DØ42 FE Ø9 74		LDII	#DNAMDF VQ7/I	POINT U TO DIRECTORY ENTRY OF SOURCE FILE
2222	DØ45 C6 ØB		LDB	#\$ØB	11 CHARACTERS IN FILENAME AND EXTENSION
2223	DØ47 BD A5 9A		JSR	LA59A	COPY NEW FILENAME TO SOURCE FILE DIRECTORY RAM IMAGE
2224	DØ4A C6 Ø3		LDB	#\$Ø3	* GET WRITE OP CODE AND
2225	DØ4C D7 EA			DCOPC	* SAVE IN DSKCON VARIABLE
2226	DØ4E 7E D6 F2		JMP	LD6F2	WRITE NEW DIRECTORY SECTOR
2227 2228		+ DO A C	VAITAV	CHECK FOR 'TO AND STRIP A FILENA	ME EDOM DACIC
2228	DØ51 C6 A5	- DU A S			'TO' TOKEN
2230	DØ53 BD B2 6F	LDDJI		LB26F	SYNTAX CHECK FOR 'TO'
2231		LDØ56	JMP		GET FILENAME FROM BASIC
2232	DØ59 BD C6 8C	LDØ56 LDØ59		LC68C	SCAN DIRECTORY FOR FILENAME
2233	DØ5C C6 42		LDB	#33*2	'FILE ALREADY EXISTS' ERROR
2234	DØ5E 7D Ø9 73		TST		CHECK FOR A MATCH
2235	DØ61 10 26 DB E1			LAC46	'AE' ERROR IF FILE IN DIRECTORY
2236 2237	DØ65 39		RTS		
2238		* WRITE	COMMAN	ın	
2239	DØ66 1Ø 27 E8 EE	WRITE			PRINT CARRIAGE RETURN TO CONSOLE OUT IF END OF LINE
2240	DØ6A 8D Ø3			LDØ6F	GO WRITE AN ITEM LIST
2241	DØ6C ØF 6F			DEVNUM	SET DEVICE NUMBER TO SCREEN
2242		LDØ6E	RTS		OUTON TOD DENTOS NUMBER ELAO
2243		LDØ6F	CMPA		CHECK FOR DEVICE NUMBER FLAG
2244 2245	DØ71 26 ØF			LDØ82	DEFAULT TO CURRENT DEVICE NUMBER IF NONE GIVEN
2245	DØ73 BD A5 A5 DØ76 BD A4 Ø6			LA5A5 LA4Ø6	SET DEVICE NUMBER; CHECK VALIDITY MAKE SURE SELECTED FILE IS AN OUTPUT FILE
2247	DØ79 9D A5			GETCCH	GET CURRENT INPUT CHARACTER
2248	DØ7B 1Ø 27 E8 D9			LB958	PRINT CR TO CONSOLE OUT IF END OF LINE
2249		LDØ7F		SYNCOMMA	SYNTAX CHECK FOR COMMA
2250	DØ82 BD B1 56	LDØ82		LB156	EVALUATE EXPRESSION
2251	DØ85 96 Ø6			VALTYP	GET VARIABLE TYPE
2252	DØ87 26 1E			LDØA7	BRANCH IF STRING
2253 2254	DØ89 BD BD D9 DØ8C BD B5 16			LBDD9 LB516	CONVERT FP NUMBER TO ASCII STRING PUT ON TEMPORARY STRING STACK
2255	DØ8F BD B9 9F			LB99F	PRINT STRING TO CONSOLE OUT
2256	5501 55 55 51		OSIK	20331	TRINI STRING TO CONSOLE COT
2257		* PRINT	ITEM S	EPARATOR TO CONSOLE OUT	
2258	DØ92 9D A5	LDØ92		GETCCH	GET CURRENT CHARACTER
	DØ92 9D A5 DØ94 1Ø 27 E8 CØ		JSR LBEQ	GETCCH LB958	GET CURRENT CHARACTER PUT CR TO CONSOLE OUT IF END OF LINE
2258 2259 2260	DØ94 10 27 E8 CØ DØ98 86 2C		JSR LBEQ LDA	GETCCH LB958 #','	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR
2258 2259 2260 2261	DØ94 10 27 E8 CØ DØ98 86 2C DØ9A BD A3 5F		JSR LBEQ LDA JSR	GETCCH LB958 #',' LA35F	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS
2258 2259 2260 2261 2262	D094 10 27 E8 C0 D098 86 2C D09A BD A3 5F D09D 0D 6E		JSR LBEQ LDA JSR TST	GETCCH LB958 #',' LA35F PRTDEV	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPRATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND
2258 2259 2260 2261 2262 2263	DØ94 10 27 E8 CØ DØ98 86 2C DØ9A BD A3 5F DØ9D ØD 6E DØ9F 27 Ø2		JSR LBEQ LDA JSR TST BEQ	GETCCH LB958 #',' LA35F PRTDEV LDØA3	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE
2258 2259 2260 2261 2262 2263 2264	D094 10 27 E8 C0 D098 86 2C D09A BD A3 5F D09D 0D 6E D09F 27 02 D0A1 86 0D	LDØ92	JSR LBEQ LDA JSR TST BEQ LDA	GETCCH LB958 #',' LA35F PRTDEV LDØA3 #CR	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR
2258 2259 2260 2261 2262 2263	DØ94 10 27 E8 CØ DØ98 86 2C DØ9A BD A3 5F DØ9D ØD 6E DØ9F 27 Ø2		JSR LBEQ LDA JSR TST BEQ LDA BSR	GETCCH LB958 #',' LA35F PRTDEV LDØA3	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267	D094 10 27 E8 C0 D098 86 2C D09A BD A3 5F D09D 0D 6E D09F 27 02 D0A1 86 0D D0A3 8D 14	LDØ92	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA	GETCCH LB958 #',' LA35F PRTDEV LDØA3 #CR LDØB9 LDØ7F	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPRATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268	D094 10 27 E8 C0 D098 B6 2C D09A BD A3 5F D09D 0D 6E D09F 27 02 D041 86 0D D0A3 8D 14 D0A5 20 D8	LDØ92 LDØA3 * PRINT	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F NG TO CONSOLE OUT	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269	D094 10 27 E8 C0 D098 86 2C D099 8D A3 5F D09D 0D 6E D09F 27 02 D0A1 86 0D D0A3 8D 14 D0A3 8D 08	LDØ92	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (")
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270	D094 10 27 E8 C0 D098 86 2C D09A BD A3 5F D09D 0D 6E D09F 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F	LDØ92 LDØA3 * PRINT	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F NG TO CONSOLE OUT LD0B0 LB09F	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269	D094 10 27 E8 C0 D098 86 2C D099 8D A3 5F D09D 0D 6E D09F 27 02 D0A1 86 0D D0A3 8D 14 D0A3 8D 08	LDØ92 LDØA3 * PRINT	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR BSR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (")
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2270 2271	DØ94 1Ø 27 E8 CØ DØ98 B6 2C DØ9A BD A3 5F DØ9D ØD 6E DØ9F 27 Ø2 DØA1 86 ØD DØA3 8D 14 DØA5 2Ø D8	LDØ92 LDØA3 * PRINT	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR BSR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F NG TO CONSOLE OUT LD0B0 LD080 LD09F LD080	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2270 2271 2272 2273 2274	D094 10 27 E8 C0 D098 86 2C D099 8D A3 5F D09D 0D 6E D09F 27 02 D041 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 8D 89 9F D0AC 8D 02 D0AE 20 E2	LDØ92 LDØA3 * PRINT LDØA7 * PRINT	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR BSR BSR BSR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F NG TO CONSOLE OUT LD0B0 LD080 LD09F LD080	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2269 2271 2272 2273 2274 2275	D094 10 27 E8 C0 D098 86 2C D099 8D A3 5F D09D 0D 6E D09F 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0B0 BD A3 5F	LDØ92 LDØA3 * PRINT LDØA7	JSR LBEQ LDA JSR TST BEQ LDA BSR BSR BSR JSR BSR JSR BSR JSR BSR JSR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LB99F LD0B0 LD0B0 LD0B0 LB99F LD0B0	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARTOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT GO PRINT SEPARATOR  SET PRINT PARAMETERS
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2270 2271 2272 2273 2274 2275 2276	D094 10 27 E8 C0 D098 86 2C D09A BD A3 5F D09D 0D 6E D09F 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0BB BD A3 5F D0BB 0D 6E	LDØ92 LDØA3 * PRINT LDØA7 * PRINT	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR BSR BSR BSR BSR BSR TST	GETCCH LB958 #',' LA35F PRTDEV LD08A3 #CR LD08B9 LD07F NG TO CONSOLE OUT LD08B0 LB99F LD08B0 LD092 i DELIMITER (") TO CONSOLE OUT LA35F PRTDEV	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2270 2271 2272 2273 2274 2275 2276 2276	D094 10 27 E8 C0 D098 86 2C D099 8D A3 5F D090 0D 6E D09F 27 02 D041 86 00 D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 8D 89 9F D0AC 8D 02 D0AE 20 E2  D0BB 8D A3 5F D0BB 0D 6E D0B5 26 B7	LDØ92 LDØA3 * PRINT LDØA7 * PRINT	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR BSR BSR BSR BSR BSR BSR BSR BSR BSR B	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD066	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2270 2271 2272 2273 2274 2275 2276 2277 2278	D094 10 27 E8 C0 D098 B6 2C D099 BD A3 5F D090 0D 6E D09F 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0B0 BD A3 5F D0B3 0D 6E D0B5 26 B7 D0B7 86 22	LD092 LD0A3 * PRINT LD0A7  * PRINT LD0B0	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR BSR BSR BSR BSR BSR BSR BSR BSR BSR B	GETCCH LB958 #',' LA35F PRTDEV LD00A3 #CR LD00B9 LD007F  NG TO CONSOLE OUT LD00B0 LB99F LD00B0 LD09B0 LD09B0 LD09B0 LD09B0 LD09B0 LD09B0 LD06B0 #""'	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE QUOTE: NON-CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2270 2271 2272 2273 2274 2275 2276 2277 2277 2278	D094 10 27 E8 C0 D098 86 2C D099 8D A3 5F D090 0D 6E D09F 27 02 D041 86 00 D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 8D 89 9F D0AC 8D 02 D0AE 20 E2  D0BB 8D A3 5F D0BB 0D 6E D0B5 26 B7	LDØ92 LDØA3 * PRINT LDØA7 * PRINT	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR BSR BSR BSR BSR BSR BSR BSR BSR BSR B	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD066	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2270 2271 2272 2273 2274 2275 2276 2277 2278	D094 10 27 E8 C0 D098 B6 2C D099 BD A3 5F D090 0D 6E D09F 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0B0 BD A3 5F D0B3 0D 6E D0B5 26 B7 D0B7 86 22	LD092 LD0A3 * PRINT LD0A7  * PRINT LD0B0	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR BSR BSR BSR BSR BSR BSR BSR BSR BSR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD066E #'"' LA282	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE QUOTE: NON-CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2271 2272 2273 2274 2275 2276 2277 2278 2277 2278	D094 10 27 E8 C0 D098 B6 2C D099 BD A3 5F D090 0D 6E D09F 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0B0 BD A3 5F D0B3 0D 6E D0B5 26 B7 D0B7 86 22	LD092 LD0A3 * PRINT LD0A7 * PRINT LD0B0	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR BSR BSR BSR BSR BSR USR BSR USR BSR DJSR BSR DJSR BSR BSR BSR BSR BSR BSR BSR BSR BSR B	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD066E #'"' LA282	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE QUOTE: NON-CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER
2258 2259 2260 2261 2262 2263 2264 2265 2266 2277 2272 2273 2274 2275 2276 2277 2278 2277 2278 2279 2280 2280 2281 2281	D094 10 27 E8 C0 D098 86 2C D099 8D A3 5F D090 0D 6E D09F 27 02 D041 86 0D D043 8D 14 D045 20 D8  D047 8D 07 D049 BD 89 9F D046 8D 02 D048 20 E2  D088 BD A3 5F D083 0D 6E D085 26 B7 D087 86 22 D086 BD C8 2E  D08C BD C8 2E	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR JSR BSR BSR BSR BSR BSR BSR BSR BSR DSR DSR DSR DSR DSR DSR DSR DSR DSR D	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD099  ID DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD06E #'"' LA282	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN *
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284	D094 10 27 E8 C0 D098 B6 2C D099 B0 A3 5F D090 0D 6E D097 27 02 D0A1 B6 0D D0A3 B0 14 D0A5 20 D8  D0A7 BD 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0B0 BD A3 5F D0B3 0D 6E D0B5 26 B7 D0B7 B6 22 D0B9 7E A2 82  D0BC BD C8 2E D0BF 4F D0C0 5F	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRIN BSR BSR BSR BSR BSR BSR BSR BSR LDA JSR LDA JSR COMMAN JSR CCLRA CLRB	GETCCH LB958 #',' LA35F PRTDEV LD00A3 #CR LD00B9 LD007F  NG TO CONSOLE OUT LD00B0 LB99F LD00B0 LD09B0 LD09B0 LD09B0 LD09B0 LD00B0 LD00B0 LD00B0 LD00B0 LD00B0 LD00B0 LD00B0 LD00B0 LD00B0 LD0B0B0 LD00B0B0 LD00B0B0B0B0B0B0B0B0B0B0B0B0B0B0B0B0B0B0	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * CLEAR TOTAL FIELD LENGTH COUNTER
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2283 2284	D094 10 27 E8 C0 D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 00 D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 8D 89 9F D0AC 8D 02 D0AE 20 E2  D0B8 BD A3 5F D0B3 00 6E D0B5 26 B7 D0B7 86 22 D0B9 7E A2 82  D0BC BD C8 2E D0BF 4F D0C0 5F D0C1 34 16	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD FIELD	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRIN BSR JSR BSR JSR TST BNE LDA JSR TST BNE DSR UJSR BSR UJSR UJSR UJSR UJSR UJSR UJSR UJSR UJ	GETCCH LB958 #',' LA35F PRTDEV LD08A3 #CR LD08B9 LD07F NG TO CONSOLE OUT LD08B0 LB99F LD08B0 LD092 S DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD06E #'"" LA282 LD LC82E X,B,A	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO
2258 2259 2260 2261 2262 2263 2264 2266 2267 2270 2271 2272 2273 2274 2275 2277 2278 2277 2278 2279 2280 2281 2282 2283 2284 2285 2285	D094 10 27 E8 C0 D098 86 2C D099 8D A3 5F D090 0D 6E D097 27 02 D041 86 0D D043 8D 14 D045 20 D8  D047 8D 07 D049 8D 89 9F D046 8D 02 D048 20 E2  D088 BD A3 5F D083 0D 6E D085 26 B7 D087 86 22 D089 7E A2 82  D08C BD C8 2E D08F 4F D0C1 34 16 D0C3 9D A5	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD	JSR LBEQ LDA JSR TST BEQ LDA BSR BRA A STRI BSR BSR BSR BSR BSR BSR BSR BSR BSR CLRA CURB LDA JSR CLRA CLRB	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD089 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  GDELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD066E #'"' LA282 LD LC82E  X,B,A GETCCH	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT IMPUT CHARACTER
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2283 2284	D094 10 27 E8 C0 D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 00 D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 8D 89 9F D0AC 8D 02 D0AE 20 E2  D0B8 BD A3 5F D0B3 00 6E D0B5 26 B7 D0B7 86 22 D0B9 7E A2 82  D0BC BD C8 2E D0BF 4F D0C0 5F D0C1 34 16	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD FIELD	JSR LBEQ LDA BSR A STRING BSR A STRING BSR A STRING BSR A STRING COMMAN JSR CLRA JSR CLRA JSR SSR BSR BSR BSR BSR BSR BSR BSR BSR B	GETCCH LB958 #',' LA35F PRTDEV LD08A3 #CR LD08B9 LD07F NG TO CONSOLE OUT LD08B0 LB99F LD08B0 LD092 S DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD06E #'"" LA282 LD LC82E X,B,A	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN  * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE
2258 2259 2260 2261 2262 2263 2264 2265 2266 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2281 2282 2283 2284 2285 2286	D094 10 27 E8 C0 D098 B6 2C D099 B A3 5F D090 0D 6E D097 27 02 D0A1 B6 0D D0A3 BD 14 D0A5 20 DB  D0A9 BD 89 9F D0AC BD 02 D0A6 BD A3 5F D0B3 0D 6E D0B5 26 B7 D0B7 B6 22 D0B8 BD A3 5F D0B7 B6 22 D0B8 C8 B7 D0B7 B6 22 D0B9 7E A2 82  D0BC BC BC BC BC D0C0 5F D0C1 34 16 D0C3 9D A5 D0C5 26 62	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD FIELD	JSR LBEQ LDA JSR BSR BSR BSR BSR BSR BSR BSR BSR COMMAN JSR CLRA CLRB BNE LDA JSR CLRA BNE BNE BNE BNE LDA JSR CLRA BNE BNE BNE BNE LDA JSR CLRA CLRB BNE BNE BNE LDA JSR CLRA CLRB BNE BNE BNE BNE BNE BNE BNE BNE BNE BN	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD087 NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LB99F LD0B0 LB99F LD0B0 LB99F LD0B0 LB99F LD0B0 LB99C  BELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD06E #'"' LA282	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT IMPUT CHARACTER
2258 2259 2260 2261 2262 2263 2264 2265 2266 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2281 2282 2283 2284 2285 2286 2287 2288 2289 2289 2290	D094 10 27 E8 C0 D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 00 D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 8D 89 9F D0AC 8D 02 D0AE 20 E2  D0B8 BD A3 5F D0B3 00 6E D0B5 26 B7 D0B7 86 22 D0B9 7E A2 82  D0BC BD C8 2E D0BF 4F D0C0 5F D0C1 34 16 D0C3 90 A5 D0C5 26 02 D0C7 35 96	LD092 LD0A3 * PRINT LD0A7  * PRINT LD0B0  LD0B9 * FIELD FIELD LD0C3	JSR LBEQ LDA JSR BSR BSR BSR BSR BSR BSR BSR BSR COMMAN JSR CLRA CLRB BNE LDA JSR CLRA BNE BNE BNE BNE LDA JSR CLRA BNE BNE BNE BNE LDA JSR CLRA CLRB BNE BNE BNE LDA JSR CLRA CLRB BNE BNE BNE BNE BNE BNE BNE BNE BNE BN	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD089 LD087F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092 GELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD066E #'"' LA282 LD LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS  * GET CONSOLE PRINT DEVICE AND  * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT SEPARATOR  SET PRINT PARAMETERS  * GET CONSOLE PRINT DEVICE AND  * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN  * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2285 2286 2289 2289 2289 2289 2289 2289 2289	D094 10 27 E8 C0 D098 B6 2C D099 B A3 5 F D090 00 6E D097 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0B0 BD A3 5 F D0B3 00 6E D0B5 26 B7 D0B7 86 22 D0B9 7E A2 82  D0BC BD C8 2E D0BF 4F D0C3 90 A5 D0C3 90 A5 D0C3 90 A5 D0C5 26 02 D0C7 35 96 D0C9 BD B7 38	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD FIELD LD0C3 LD0C9	JSR LBEQ LDA JSR BSR BSR BSR BSR BSR BSR BSR BSR BSR B	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  S DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD06E #'"' LA282  DD LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738 X,B	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2282 2284 2285 2286 2287 2288 2289 2290 2291	D094 10 27 E8 C0 D098 B6 2C D099 B A3 5 F D090 00 6E D097 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0B0 BD A3 5 F D0B3 00 6E D0B5 26 B7 D0B7 86 22 D0B9 7E A2 82  D0BC BD C8 2E D0BF 4F D0C3 90 A5 D0C3 90 A5 D0C3 90 A5 D0C5 26 02 D0C7 35 96 D0C9 BD B7 38	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD FIELD  LD0C3  LD0C9  * AT THI	JSR TST BSR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  G DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 ASSERVAN GETCCH LD0B0 ASSERVAN	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NG INFORMATION ON IT:
2258 2259 2260 2261 2262 2263 2264 2265 2266 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2291 2291 2292 2293	D094 10 27 E8 C0 D098 B6 2C D099 B A3 5 F D090 00 6E D097 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0B0 BD A3 5 F D0B3 00 6E D0B5 26 B7 D0B7 86 22 D0B9 7E A2 82  D0BC BD C8 2E D0BF 4F D0C3 90 A5 D0C3 90 A5 D0C3 90 A5 D0C5 26 02 D0C7 35 96 D0C9 BD B7 38	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD  LD0C3  LD0C9  * AT THI * ,S = F	JSR LBEQ LDA JSR TST BEQ LDA BSR BSR JSR BSR JSR BSR JSR BSR JSR BSR JSR STRING LDA JSR TST BNE LDA JMP COMMAN JSR CLRA JSR PSHS JSR SS POIN	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD087 NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LB99F LD0B0 LB99F LD0B0 LB99F LD0B0 LA35F PRTDEV LD06E #'"' LA282 DD LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738 X,B IT THE STACK WILL HAVE THE FOLLOWI EMGTH 1 2,S = RANDOM FILE BUFFE	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS  * GET CONSOLE PRINT DEVICE AND  * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS  * GET CONSOLE PRINT DEVICE AND  * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN  * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NG INFORMATION ON IT: R ADDRESS
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2280 2281 2282 2283 2284 2285 2286 2289 2290 2291 2292 2293 2291 2292	D094 10 27 E8 C0 D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 00 D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 8D 89 9F D0AC 8D 02 D0AE 20 E2  D0B8 BD A3 5F D0B3 00 6E D0B5 26 B7 D0B7 86 22 D0B9 7E A2 82  D0BC BD C8 2E D0B9 4F D0C0 5F D0C1 34 16 D0C3 9D A5 D0C7 35 96 D0C9 BD 87 38 D0CC 34 14	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD  LD0C3  LD0C9  * AT THI * ,S = F	JSR LBEQ LDA STRING BSR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  G DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 ASSERVAN GETCCH LD0B0 ASSERVAN	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NG INFORMATION ON IT: R ADDRESS ER
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2270 2271 2272 2273 2274 2275 2276 2280 2281 2282 2282 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294	D094 10 27 E8 C0 D098 B6 2C D099 BD A3 5F D090 D0 6E D097 27 Q2 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 BD 89 9F D0AC 8D 02 D0AE 20 E2  D0B0 BD A3 5F D0B3 0D 6E D0B5 26 B7 D0B7 86 22 D0B7 86 22 D0B7 46 22 D0B7 47 A2 82	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD  LD0C3  LD0C9  * AT THI * ,S = F	JSR TST BBRA A STRI BBRA BSR BBRA STRING STR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B8 LB99F LD0B0 LD092  DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD0B0 LD080 LD092  DELIMITER (") TO CONSOLE OUT LA25F PRTDEV LD06E #'" LA282  DL LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738 X,B  IT THE STACK WILL HAVE THE FOLLOWI LENGTH 1 2,S = RANDOM FILE BUFFE LL FIELD LENGTH 5 6,S = FCD POINT	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NG INFORMATION ON IT: R ADDRESS ER CLEAR MS BYTE
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2280 2281 2282 2283 2284 2285 2286 2289 2290 2291 2292 2293 2291 2292	D094 10 27 E8 C0 D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 00 D0A3 8D 14 D0A5 20 D8  D0A7 8D 07 D0A9 8D 89 9F D0AC 8D 02 D0AE 20 E2  D0B8 BD A3 5F D0B3 00 6E D0B5 26 B7 D0B7 86 22 D0B9 7E A2 82  D0BC BD C8 2E D0B9 4F D0C0 5F D0C1 34 16 D0C3 9D A5 D0C7 35 96 D0C9 BD 87 38 D0CC 34 14	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD  LD0C3  LD0C9  * AT THI * ,S = F	JSR LBEQ LDA JSR TST BEQ LDA BSR BSR JSR BSR JSR BSR JSR BSR JSR BSR JSR STRING CURA JSR CLRA JSR STRING SSR SSR SSR SSR SSR SSR SSR SSR SSR SS	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD087 NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LB99F LD0B0 LB99F LD0B0 LB99F LD0B0 LA35F PRTDEV LD06E #'"' LA282 DD LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738 X,B IT THE STACK WILL HAVE THE FOLLOWI EMGTH 1 2,S = RANDOM FILE BUFFE	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NG INFORMATION ON IT: R ADDRESS ER
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2270 2271 2272 2273 2274 2275 2276 2277 2278 2279 2281 2282 2283 2284 2285 2286 2287 2288 2289 2291 2292 2293 2294	D094 10 27 E8 C0 D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A6 8D 02 D0A6 8D 02 D0B8 8D A3 5F D0B3 0D 6E D0B5 26 87 D0B7 86 22 D0B9 7E A2 82  D0BC 8D C8 2E D0C1 34 16 D0C3 9D A5 D0CC 34 14  D0CE 4F D0CF E3 63	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD  LD0C3  LD0C9  * AT THI * ,S = F	JSR LBEQ LDA JSR BSR BSR BSR BSR BSR BSR BSR BSR BSR B	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD089 LD087 NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LB99F LD0B0 LD092  DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD06E #'"' LA282  D LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738 X,B  IT THE STACK WILL HAVE THE FOLLOWI ENGTH 1 2,S = RANDOM FILE BUFFE LL FIELD LENGTH 5 6,S = FCD POINT \$03,S	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NOE INFORMATION ON IT: R ADDRESS ER CLEAR MS BYTE ADD FIELD LENGTH TO TOTAL FIELD LENGTH COUNTER 'FO' ERROR IF SUM > \$FFFF POINT X TO FCB
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2271 2272 2273 2274 2275 2276 2277 2278 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2291 2292 2293 2294 2293 2294 2295	D094 10 27 E8 CØ D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A6 8D 02 D0AC 8D 02 D0BB 8D A3 5F D0B3 0D 6E D0B5 26 87 D0B7 86 22 D0B7 87 A2 82  D0BC BD C8 2E D0B7 4F D0C0 5F D0C1 34 16 D0C3 9D A5 D0C2 34 14  D0CE 4F D0C7 35 96 D0C9 BD 87 38 D0CC 34 14  D0CE 4F D0C7 E3 63 D0D1 25 67 D0D3 AE 65 D0D3 A9 99	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD  LD0C3  LD0C9  * AT THI * ,S = F	JSR LBEQ LDA JSR BSR BSR BSR BSR BSR BSR BSR BSR BSR B	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  B DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD06E #'"' LA282  LB0 LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738 X,B  IT THE STACK WILL HAVE THE FOLLOWI LENGTH 1 2,S = RANDOM FILE BUFFE LL FIELD LENGTH 5 6,S = FCD POINT \$03,S LD0DA \$05,S FCBRLN,X	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NG INFORMATION ON IT: R ADDRESS ER CLEAR MS BYTE ADD FIELD LENGTH TO TOTAL FIELD LENGTH COUNTER 'FO' ERROR IF SUM > \$FFFF POINT X TO FCB * COMPARE TO RECORD LENGTH & BRANCH IF
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2268 2271 2272 2273 2274 2275 2278 2279 2280 2281 2282 2283 2284 2285 2286 2287 2288 2289 2290 2291 2292 2293 2294 2295 2298	D094 10 27 E8 C0 D098 B6 2C D097 27 02 D041 B6 6D D047 B6 6D D043 B1 14 D045 20 D8  D047 BD 89 9F D04C B0 6E D098 BD A3 5F D088 BD A3 5F D083 00 6E D085 26 B7 D087 B6 22 D088 BD C8 2E D089 7E A2 82  D08C BD C8 2E D08C 5F D0C1 34 16 D0C3 9D A5 D0C5 26 02 D0C7 35 96 D0C9 BD B7 38 D0CC 34 14	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD  LD0C3  LD0C9  * AT THI * ,S = F * 3 4,S	JSR LBEQ LDA JSR BRA A STRING BSR BSR BSR BSR BSR BSR BSR BSR BSR STRING JSR TST BSR LDA JMP LDA JSR PSHS JSR BNE LDA JMP LCCMMAN JCRA BNE LDA JMP LCCLRB BNE LDA JMP LCCLRB BNE LDA JMP LCCLRB BNE LDA LDX CMBD BNE LDA LDX CMBD BNE LDA LDX CMBD BNE LDA LDX CMBD BNE LDA LDX	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD066E #'"' LA282  DL LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738 X,B  IT THE STACK WILL HAVE THE FOLLOWI ENGTH 1 2,S = RANDOM FILE BUFFE LFIELD LENGTH 5 6,S = FCD POINT \$03,S LD0DA \$05,S FCCBRLN,X LD0DF	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NG INFORMATION ON IT: R ADDRESS ER CLEAR MS BYTE ADD FIELD LENGTH TO TOTAL FIELD LENGTH COUNTER 'FO' ERROR IF SUM > SFFFF POINT X TO FCB * COMPARE TO RECORD LENGTH & BRANCH IF *TOTAL FIELD LENGTH < RECORD LENGTH & STANCH IF *TOTAL FIELD LENGTH < RECORD LENGTH & STANCH IF *TOTAL FIELD LENGTH < RECORD LENGTH & STANCH IF *TOTAL FIELD LENGTH < RECORD LENGTH & STANCH IF *TOTAL FIELD LENGTH < RECORD LENGTH & STANCH IF *TOTAL FIELD LENGTH < RECORD LENGTH & STANCH IF *TOTAL FIELD LENGTH < RECORD LENGTH # STOTAL FIELD LENGTH < RECORD LENGTH **  **TOTAL FIELD LENGTH < RECORD
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2270 2271 2272 2273 2274 2275 2276 2277 2282 2282 2284 2285 2284 2285 2289 2290 2291 2292 2293 2294 2295 2296 2297	D094 10 27 E8 C0 D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A6 8D 02 D0AC 8D 02 D0A6 8D A3 5F D0B3 00 6E D0B5 26 B7 D0B7 8C 22 D0B8 BD C8 2E D0B8 30 6E D085 26 87 D087 86 22 D087 86 22 D087 34 14  D0CC 34 14  D0CC 4F D0CG 34 16 D0CG 34 14  D0CC 4F D0CG 34 14	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD  LD0C3  LD0C9  * AT THI * ,S = F	JSR TST BBRA A STRI BBRA BSR BBRA STRING STR	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0  #'"' LA282  DL LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738 X,B  IT THE STACK WILL HAVE THE FOLLOWI LB73B X,B  IT THE STACK WILL HAVE THE FOLLOWI LB73B LG FIELD LENGTH 5 6,S = FCD POINT \$03,S LD0D0A \$05,S FCBRIN,X LD0D0F #34*2	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NOE INFORMATION ON IT: R ADDRESS ER CLEAR MS BYTE ADD FIELD LENGTH TO TOTAL FIELD LENGTH COUNTER 'FO' ERROR IF SUM > \$FFFF POINT X TO FCB * COMPARE TO RECORD LENGTH & BRANCH IF **TOTAL FIELD LENGTH < RECORD LENGTH 'FIELD OVERFLOW' ERROR
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2271 2272 2273 2274 2275 2276 2277 2278 2279 2280 2281 2282 2283 2284 2289 2290 2291 2292 2293 2294 2295 2297 2299 2300 2301 2302	D094 10 27 E8 CØ D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A6 8D 02 D0AC 8D 02 D0B8 8D A3 5F D0B3 0D 6E D0B5 26 B7 D0B7 86 22 D0B9 7E A2 82  D0BC BD C8 2E D0B7 4F D0C7 35 96 D0C9 BD B7 38 D0CC 34 14  D0CE 4F D0C7 E3 63 D0D1 25 67 D0D3 AE 65 D0D3 30 65 D0D3 AE 65 D0C9 37 AE 65 D0D3 AE 65 D0D3 AE 65 D0D3 AB 65 D0D3 AE 65 D0D3 AB 65 D0B3 AB 65 D	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD FIELD  LD0C3  LD0C9  * AT THI * , S = F * 3 4, S	JSR LBEQ LDA JSR BSR BSR BSR BSR BSR BSR BSR BSR BSR B	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD08B0 LB99F LD08B0 LB99F LD08B0 LB99F LD08B0 LB99F LA28E  LA282  LB28E  X,B,A GETCCH LB738 X,B  IT THE STACK WILL HAVE THE FOLLOWI LBNGTH 1 2,S = RANDOM FILE BUFFE LL FIELD LENGTH 5 6,S = FCD POINT \$03,S LD0DA \$05,S FCBRLN,X LD0DF #344*2 LAC46	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NOS INFORMATION ON IT: R ADDRESS ER CLEAR MS BYTE ADD FIELD LENGTH TO TOTAL FIELD LENGTH COUNTER 'FO' ERROR IF SUM > \$FFFF POINT X TO FCB * COMPARE TO RECORD LENGTH & BRANCH IF *TOTAL FIELD LENGTH < RECORD LENGTH 'FIELD OVERFLOW' ERROR JUMP TO ERROR DRIVER
2258 2259 2260 2261 2262 2263 2264 2265 2266 2267 2270 2271 2272 2273 2274 2275 2276 2277 2282 2282 2284 2285 2284 2285 2289 2290 2291 2292 2293 2294 2295 2296 2297	D094 10 27 E8 C0 D098 86 2C D099 80 A3 5F D090 00 6E D097 27 02 D0A1 86 0D D0A3 8D 14 D0A5 20 D8  D0A6 8D 02 D0AC 8D 02 D0A6 8D A3 5F D0B3 00 6E D0B5 26 B7 D0B7 8C 22 D0B8 BD C8 2E D0B8 30 6E D085 26 87 D087 86 22 D087 86 22 D087 34 14  D0CC 34 14  D0CC 4F D0CG 34 16 D0CG 34 14  D0CC 4F D0CG 34 14	LD092  LD0A3  * PRINT LD0A7  * PRINT LD0B0  LD0B9  * FIELD  LD0C3  LD0C9  * AT THI * ,S = F * 3 4,S	JSR LBEQ LDA JSR BRA A STRING BSR BRA STRING JSR TST BSR BRA LDA JMP LDA JSR STRING COMMAN JSR STRING CLRB BNE LDA JMP LCLRA ADDD BNE LDA LDX CMPD LDX CMPD LDX LDB LDB LDB LDB LDB LDB LDU LDU LDX LDB LDB LDB LDB LDU LDU LDX LDD LDD LDD LDD LDD LDD LDD LDD LDD	GETCCH LB958 #',' LA35F PRTDEV LD0A3 #CR LD0B9 LD07F  NG TO CONSOLE OUT LD0B0 LB99F LD0B0 LD092  DELIMITER (") TO CONSOLE OUT LA35F PRTDEV LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0 LD0B0  #'"' LA282  DL LC82E  X,B,A GETCCH LD0C9 A,B,X,PC LB738 X,B  IT THE STACK WILL HAVE THE FOLLOWI LB73B X,B  IT THE STACK WILL HAVE THE FOLLOWI LB73B LG FIELD LENGTH 5 6,S = FCD POINT \$03,S LD0D0A \$05,S FCBRIN,X LD0D0F #34*2	PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM  PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR  SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * RETURN IF CASSETTE QUOTE: NON-CASSETTE SEND TO CONSOLE OUT  EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO GET CURRENT INPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW NOE INFORMATION ON IT: R ADDRESS ER CLEAR MS BYTE ADD FIELD LENGTH TO TOTAL FIELD LENGTH COUNTER 'FO' ERROR IF SUM > \$FFFF POINT X TO FCB * COMPARE TO RECORD LENGTH & BRANCH IF **TOTAL FIELD LENGTH < RECORD LENGTH 'FIELD OVERFLOW' ERROR

2305 2306 2307 2308 2309 2310 2311 2312	DØE3 EC ØB DØE5 33 CB DØE7 EF 61 DØE9 C6 FF DØEB BD B2 6F DØEE C6 A7 DØFØ BD B2 6F DØF3 BD B3 57	LDD FCBBUF,X  LEAU D,U  STU \$01,S  LDB #SFF  JSR LB26F  LDB #\$A7  JSR LB26F  JSR LB357	POINT ACCD TO START OF RANDOM FILE BUFFER *POINT U TO THIS FIELD'S SLOT IN THE RANDOM *FILE BUFFER AND SAVE IT ON THE STACK SECONDARY TOKEN SYNTAX CHECK FOR SECONDARY TOKEN 'AS' TOKEN SYNTAX CHECK FOR 'AS' TOKEN EVALUATE VARIABLE
2313 2314 2315 2316 2317 2318	DØF6 BD B1 46 DØF9 35 44 DØFB E7 84 DØFD EF Ø2 DØFF 2Ø C2	JSR LB146 PULS B,U STB ,X STU \$02,X BRA LD0C3	'TM' ERROR IF NUMERIC VARIABLE  * PULL STRING ADDRESS AND LENGTH  * OFF OF THE STACK AND SAVE THEM  * IN STRING DESCRIPTOR CHECK FOR ANOTHER FIELD SPECIFICATION
2319 2320 2321	D101 86	* RSET COMMAND RSET LDA #\$4F	SKIP ONE BYTE
2322 2323 2324 2325 2326 2327 2328 2329 2330 2331 2332	D102 4F D103 34 02 D105 BD B3 57 D108 BD B1 46 D108 B 4 10 D109 AE 02 D10F 8C 09 89 D112 25 05 D114 BC 09 4A D117 25 05 D119 C6 46 D118 7E AC 46 D11E C6 B3 D120 BD B2 6F D123 BD 87 48	* LSET COMMAND LSET CLRA PSHS A JSR LB146 PSHS X LDX \$92,X CMPX #DFLBUF BLO LD119 CMPX FCBADR BLO LD11E LD119 LDB #2*35 JMP LAC46 LD11E LDB #\$B3 JSR LB26F JSSR LB748	LSET FLAG = 0  SAVE RSET(\$4F),LSET(00) FLAG ON THE STACK EVALUATE FIELD STRING VARIABLE 'TM' ERROR IF NUMERIC VARIABLE SAVE STRING DESCRIPTOR ON STACK POINT X TO ADDRESS OF STRING * COMPARE STRING ADDRESS TO START OF RANDOM * FILE BUFFER; 'SE' ERROR IF < RANDOM FILE BUFFER = COMPARE STRING ADDRESS TO TOP OF RANDOM FILE BUFFER 'SET TO NON-FIELDED STRING' ERROR JUMP TO ERROR HANDLER  * * SYNTAX CHECK FOR '=' TOKEN = EVALUATE DATA STRING EXPRESSION; RETURN WITH X
2338 2339 2340 2341 2342 2343 2344	D126 35 20 D128 A6 A4 D12A 27 2E D12C 34 04 D12E C6 20 D130 EE 22	* PULS Y LDA ,Y BEQ LD15A PSHS B LDB #SPACE LDU \$82,Y	=POINTING TO STRING; ACCB = LENGTH POINT Y TO FIELD STRING DESCRIPTOR GET LENGTH OF FIELD STRING RETURN IF NULL STRING SAVE LENGTH OF DATA STRING ON STACK PREPARE TO FILL DATA STRING WITH BLANKS POINT U TO FIELD STRING ADDRESS
2345 2346 2347 2348 2349 2350 2351 2352 2353 2354 2355	D132 E7 CØ D134 4A D135 26 FB D137 E6 EØ D139 27 IF D13B E1 A4 D13D 25 Ø4 D14I 6F E4	* FILL THE FIELDED STRING WITH BLANKS LD132 STB ,U+ DECA BNE LD132 LDB ,S+ BEQ LD15A CMPB ,Y BLO LD143 LDB ,Y CLR ,S *	STORE A SPACE IN FIELDED STRING DECREMENT LENGTH COUNTER KEEP FILLING W/SPACES IF NOT DONE *GET THE LENGTH OF THE DATA STRING AND *RETURN IF IT IS NULL (ZERO) =COMPARE LENGTH OF DATA STRING TO LENGTH OF FIELD =STRING, BRANCH IF FIELD STRING > DATA STRING *GET THE LENGTH OF THE FIELD STRING AND FORCE THE *RSET/LSET FLAG TO LSET (Ø) IF DATA STRING LENGTH IS *>= THE FIELD STRING LENGTH. THIS WILL CAUSE THE RIGHT
2356 2357 2358 2359 2360	D143 EE 22 D145 6D EØ D147 27 ØE	* LD143 LDU \$02,Y TST ,S+ BEQ LD157 * RSET ROUTINE	*SIDE OF THE DATA STRING TO BE TRUNCATED LOAD U WITH THE ADDRESS OF THE FIELD STRING * GET THE RSET/LSET FLAG FROM THE STACK * AND BRANCH IF LSET
2361 2362 2363 2364 2365 2366 2367 2368	D149 34 04 D14B 4F D14C 50 D14D 82 00 D14F EB A4 D151 89 00 D153 33 CB	PSHS B CLRA NEGB SBCA #\$00 ADDB ,Y ADCA #\$500 LEAU D,U	SAVE THE NUMBER OF BYTES TO MOVE INTO THE FIELD STRING  = TAKE THE 2'S COMPLEMENT OF AN UNSIGNED  = NUMBER IN ACCB - LEAVE THE DOUBLE BYTE SIGNED  = RESULT IN ACCD  * ADD THE LENGTH OF THE FIELD STRING TO THE INVERSE  * OF THE NUMBER OF BYTES TO BE MOVED  =ADD RESULT TO START OF FIELD STRING. NOW U  =WILL POINT TO (-NUMBER OF BYTES TO MOVE)
2371 2372	D155 35 Ø4 D157 7E A5 9A D15A 35 82	PULS B LD157 JMP LA59A LD15A PULS A,PC	=FROM THE RIGHT SIDE OF THE FIELD STRING GET THE NUMBER OF BYTES TO MOVE MOVE ACCB BYTES FROM X TO U (DATA TO FIELD STRING) PULL LSET/RSET FLAG OFF OF STACK AND RETURN
2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2399 2391 2392 2393 2394 2395	D15C BD 95 AC D15F FC 09 4A D162 83 09 89 D165 34 06 D167 F6 09 5B D16A 34 04 D16C 9D A5 D16E 81 2C D170 27 0F D172 BD B7 0B D175 C1 0F D177 10 22 E2 CF D17B E7 E4 D17D 9D A5 D17F 27 08 D18T BD 82 6D D184 BD 83 E6 D187 ED 61 D189 BD CA E9 D18C E6 E4 D18E 34 04	* FILES COMMAND FILES JSR L95AC LDD FCBADR SUBD #DFLBUF PSHS B, A LDB FCBACT PSHS B JSR GETCCH CMPA #',' BEQ LD181 JSR EVALEXPB CMPB #15 LBHI LB44A STB ,S JSR GETCCH BEQ LD189 LD181 JSR SYNCOMMA JSR LB3E6 STD \$Ø1,S LD189 LD189 LD180 L	RESET SAM DISPLAY PAGE AND VDG MODE GET START OF FILE BUFFERS SUBTRACT THE START OF RANDOM FILE BUFFER SPACE SAVE DEFAULT VALUE OF RANDOM FILE BUFFER SPACE ON STACK * GET CURRENT NUMBER OF FCBS * AND SAVE ON THE STACK (DEFAULT VALUE) GET CURRENT IMPUT CHAR CHECK FOR COMMA BRANCH IF COMMA - NO BUFFER NUMBER PARAMETER GIVEN EVALUATE EXPRESSION (BUFFER NUMBER) 15 FCBS MAX BRANCH IF > 15 - 'ILLEGAL FUNCTION CALL' SAVE NUMBER OF FCBS ON STACK CHECK CURRENT INPUT CHAR BRANCH IF END OF LINE SYNTAX CHECK FOR COMMA EVALUATE EXPRESSION, RETURN VALUE IN ACCD SAVE RANDOM FILE BUFFER SIZE ON STACK CLOSE FILES * GET THE NUMBER OF BUFFERS TO MAKE AND * INITIALIZE A BUFFER COUNTER ON THE STACK
2396 2397 2398 2399 2400	D190 CC 09 89 D193 E3 62 D195 25 71 D197 ED 62	ADDD \$80,S BLO LD208 STD \$02,S * RESERVE SPACE FOR FCBS	GET START OF RANDOM FILE BUFFERS ADD THE NEWLY SPECIFIED RANDOM FILE BUFFER SPACE 'OUT OF MEMORY' ERROR IF > \$FFFF SAVE START OF FCBS

2401	D199 C3 Ø1 19	LD199		#FCBLEN	FCBLEN REQUIRED FOR EACH BUFFER
2402	D19C 25 6A			LD2Ø8	'OUT OF MEMORY' ERROR IF > \$FFFF
24Ø3 24Ø4	D19E 6A E4 D1AØ 2A F7		DEC	,S LD199	DECREMENT BUFFER COUNTER *BRANCH IF NOT DONE - THE BPL WILL SET UP ONE MORE BUFFER
2405	DIAD EN 17	*	DIL	E0133	*THAN THE NUMBER REQUESTED. THIS EXTRA BUFFER IS THE SYSTEM BUFFER
2406		*			*AND IS LOCATED AT THE END OF THE NORMAL FCBS. ONLY SYSTEM ROUTINES
2407		*			*(COPY, BACKUP, MERGE ETC.) MAY ACCESS THIS BUFFER.
24Ø8	D1A2 5D		TSTB		AT AN EXACT 256 BYTE BOUNDARY?
2409	D1A3 27 Ø3			LD1A8	YES
241Ø 2411	D1A5 4C D1A6 27 60		INCA	LD2Ø8	NO - ADD 256 'OUT OF MEMORY' ERROR IF PAST \$FFFF
2412	D1A8 85 Ø1	LD1A8		#\$01	ON A 512 BYTE BOUNDARY?
2413	D1AA 27 Ø3		BEQ	LD1AF	YES
2414	D1AC 4C		INCA		NO - ADD 256
2415	D1AD 27 59	10145		LD208	'OM' ERROR IF PAST \$FFFF
2416 2417	D1AF A7 E4 D1B1 DC 1B	LD1AF	STA LDD	,S VARTAB	SAVE MS BYTE OF NEW GRAPHIC RAM START GET START OF VARIABLES
2418	D1B3 9Ø BC			GRPRAM	*SUBTRACT THE OLD GRAPHIC RAM START - ACCD CONTAINS LENGTH
2419		*			*OF PROGRAM PLUS RESERVED GRAPHIC RAM
2420	D1B5 AB E4		ADDA		ADD IN THE AMOUNT OF RAM CALCULATED ABOVE
2421	D1B7 25 4F			LD208	'OUT OF MEMORY' ERROR IF > \$FFFF
2422 2423	D1B9 1F Ø1 D1BB 4C		TFR INCA	U,X	SAVE NEW VARTAB IN X *ADD 256 - TO GUARANTEE ENOUGH ROOM SINCE ALL CALCULATIONS USE
2424	DIDD 40	*	INCA		*ONLY THE MSB OF THE ADDRESS
2425	D1BC 27 4A		BEQ	LD208	'OUT OF MEMORY' ERROR IF PAST \$FFFF
2426	D1BE 10 93 21			FRETOP	IS IT GREATER THAN THE START OF STRING SPACE
2427	D1C1 24 45			LD208	'OUT OF MEMORY' IF > START OF STRING SPACE
2428 2429	D1C3 4A D1C4 93 1B		DECA	VARTAB	SUBTRACT 256 - COMPENSATE FOR INCA ABOVE SUBTRACT START OF VARIABLES
2430	D1C6 D3 19			TXTTAB	ADD START OF BASIC
2431	D1C8 1F Ø2		TFR		Y HAS NEW START OF BASIC
2432	D1CA A6 E4		LDA		* GET THE GRAPHIC RAM START, SUBTRACT
2433	D1CC 90 BC			GRPRAM	* THE OLD GRAPHIC RAN START AND SAVE
2434 2435	D1CE 1F 89 D1DØ 9B BA		TFR	A,B BEGGRP	* THE DIFFERENCE IN ACCA AND ACCB = ADD THE OLD GRAPHIC PAGE START AND
2436	D1D2 97 BA			BEGGRP	= STORE THE NEW START OF GRAPHICS RAM
2437	D1D4 DB B7			ENDGRP	* ADD THE OLD GRAPHIC RAM END ADDRESS AND
2438	D1D6 D7 B7			ENDGRP	* STORE THE NEW END OF GRAPHICS RAM
2439	D1D8 35 46		PULS	A,B,U	= ACCA=MSB OF START OF GRAPHIC RAM; ACCB=NUMBER OF FILE BUFFERS
244Ø 2441	D1DA 97 BC	*	CTA	GRPRAM	= U=START OF FILE BUFFERS SAVE NEW START OF GRAPHIC RAM
2442	D1DC F7 Ø9 5B			FCBACT	NUMBER OF FILE BUFFERS
2443	D1DF FF Ø9 4A			FCBADR	START OF FILE BUFFERS
2444	D1E2 96 68			CURLIN	GET CURRENT LINE NUMBER
2445	D1E4 4C		INCA	1.0455	ARE WE IN DIRECT MODE?
2446 2447	D1E5 27 Ø8 D1E7 1F 2Ø			LD1EF Y,D	YES - MOVE BASIC PROGRAM MOVE NEW START OF BASIC TO ACCD
2448	D1E9 93 19			TXTTAB	SUBTRACT OLD START OF BASIC
2449	D1EB D3 A6			CHARAD	ADD OLD INPUT POINTER
2450	D1ED DD A6		STD	CHARAD	SAVE NEW INPUT POINTER
2451	D1EF DE 1B	LD1EF		VARTAB	POINT U TO OLD START OF VARIABLES
2452	D1F1 9F 1B			VARTAB	SAVE NEW START OF VARIBLES
2453 2454	D1F3 11 93 1B D1F6 22 13			VARTAB LD2ØB	* COMPARE OLD START OF VARIABLES TO NEW START OF  * VARIABLES & BRANCH IF OLD > NEW
2455	21.0 22 10	* MOVE I		PROGRAM IF OLD START ADDRESS <= N	
2456	D1F8 A6 C2	LD1F8	LDA	, - U	GET A BYTE
2457	D1FA A7 82		STA		MOVE 1T
2458 2459	D1FC 11 93 19 D1FF 26 F7			TXTTAB	AT START OF BASIC PROGRAM?
2459	D2Ø1 1Ø 9F 19			LD1F8 TXTTAB	STORE NEW START OF BASIC PROGRAM
2461	D2Ø4 6F 3F			-1,Y	RESET START OF PROGRAM FLAG
2462	D206 20 13		BRA	LD21B	CLOSE ALL FILES
2463	D2Ø8 7E AC 44	LD2Ø8		LAC44	'OUT OF MEMORY' ERROR
2464 2465	D2ØB DE 19	* MOVE I LD2ØB		PROGRAM IF OLD START ADDRESS > NE TXTTAB	W START ADDRESS POINT U TO OLD START OF BASIC
2466	D2ØD 1Ø 9F 19	LUZWB		TXTTAB	SAVE NEW START OF BASIC
2467	D210 6F 3F			-1,Y	RESET START OF BASIC FLAG
2468	D212 A6 CØ	LD212	LDA		GET A BYTE
2469	D214 A7 AØ		STA	,Y+	MOVE IT
247Ø 2471	D216 10 9C 1B D219 26 F7			VARTAB LD212	AT START OF VARIABLES NO - MOVE ANOTHER BYTE
2471			JIL		
2473				CBS AND RECALCULATE FCB START ADD	RESSES
	D21B CE Ø9 28		LDU	#FCBV1	POINT U TO FILE BUFFER POINTERS
	D21E BE Ø9 4A			FCBADR	POINT X TO START OF BUFFERS
2476 2477	D221 5F D222 AF C1	LD222	CLRB		RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE
2477	D224 6F ØØ	LULLL		,U++ FCBTYP,X	RESET FILE TYPE TO CLOSED
2479	D226 3Ø 89 Ø1 19			FCBLEN, X	GO TO NEXT FCB
2480	D22A 5C		INCB		INCREMENT FILE COUNTER
2481	D22B F1 Ø9 5B			FCBACT LD222	CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB
2482 2483	D22E 23 F2 D23Ø 7E 96 CB			L96CB	BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.
2484			2111		
2485		* UNLOAD	D COMM	AND	
2486		UNLOAD			GET DRIVE NUMBER
2487	D235 5F	1,000.0	CLRB		CLEAR FILE COUNTER
2488 2489	D236 5C D237 BD C7 49	LD236	INCB	LC749	INCREMENT FILE COUNTER POINT X TO FCB
2490	D237 BD C7 49			LD249	BRANCH IF FILE NOT OPEN
2491	D23C A6 Ø1		LDA	FCBDRV,X	CHECK DRIVE NUMBER
	D23E 91 EB			DCDRV	DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER?
	D240 26 07			LD249	NO MATCH - DO NOT CLOSE THE FILE
2494 2495	D242 34 Ø4 D244 BD CB Ø6		PSHS JSR	LCBØ6	SAVE FILE COUNTER ON THE STACK CLOSE FCB
	D247 35 Ø4		PULS		RESTORE FILE COUNTER

2497	D249 F1 Ø9 5B	LD249	CMPB	FCBACT	CHECKED ALL FILES?
2498	D24C 23 E8			LD236	NO
2499	D24E 39		RTS		
2500	DO4E EC 00 EA			MBER FROM BASIC - USE THE DEFAULT	
25Ø1 25Ø2	D24F F6 Ø9 5A D252 9D A5	LD24F		DEFDRV GETCCH	GET DEFAULT DRIVE NUMBER GET NEXT INPUT CHAR
2503	D254 27 Ø9			LD25F	USE DEFAULT DRIVE NUMBER IF NONE GIVEN
25Ø4	D256 BD B7 ØB	LD256		EVALEXPB	EVALUATE EXPRESSION
25Ø5	D259 C1 Ø3		CMPB	#\$Ø3	4 DRIVES MAX
2506	D25B 10 22 D3 C0		LBHI		'DEVICE NUMBER ERROR' IF > 3
2507	D25F D7 EB	LD25F		DCDRV	STORE IN DSKCON VARIABLE
25Ø8 25Ø9	D261 39		RTS		
2510		* BACKU	P COMMA	ND	
2511	D262 10 27 D3 B9		LBEQ		DEVICE NUMBER ERROR IF NO DRIVE NUMBERS GIVEN
2512	D266 BD 95 AC		JSR		RESET SAM DISPLAY PAGE AND VOG MODE
	> D269 BD D2 56		JSR		* GET SOURCE DRIVE NUMBER AND SAVE
2514 2515	D26C F7 Ø6 FF D26F 9D A5			DBUFØ+255 GETCCH	* IT AT TOP OF DBUFØ (TOP OF NEW STACK) GET A CHARACTER FROM BASIC
2516	D271 27 Ø8			LD27B	BRANCH IF END OF LINE
2517	D273 C6 A5			#\$A5	TOKEN FOR 'TO'
2518	D275 BD B2 6F		JSR	LB26F	SYNTAX CHECK FOR 'TO'
	> D278 BD D2 56			LD256	GET DESTINATION DRIVE NUMBER
2520	D27B 10 CE 06 FF	LD27B		#DBUFØ+255	PUT STACK AT TOP OF DBUFØ
2521 2522	D27F 34 Ø4 D281 BD A5 C7		PSHS JSR		SAVE DESTINATION DRIVE NUMBER ON STACK SYNTAX ERROR IF NOT END OF LINE
2523	D284 BD CA E9		JSR		CLOSE ALL FILES
2524	D287 6F E2			,-S	CLEAR A TRACK COUNTER ON STACK
2525	D289 8E Ø9 88			#DFLBUF-1	POINT X TO TOP OF DISK RAM VARIABLES
2526	D28C 6C E4	LD28C		,\$	INCREMENT TRACK COUNTER
2527	D28E 3Ø 89 12 ØØ			SECMAX*SECLEN,X	INCREMENT X BY ONE TRACK
2528 2529	D292 9C 27 D294 23 F6		BLS	MEMSIZ LD28C	COMPARE TO TOP OF NON RESERVED RAN KEEP GOING IF MORE FREE RAM LEFT
2530	D296 6A E4			,S	DECREMENT TRACK COUNTER
2531	D298 10 27 D9 A8		LBEQ		'OM' ERROR IF < 1 TRACK OF FREE RAM
2532	D29C 86 23		LDA	#TRKMAX	GET MAXIMUM NUMBER OF TRACKS INITIALIZE REMAINING TRACKS CTR
2533	D29E 5F		CLRB		INITIALIZE TRACKS WRITTEN COUNTER TO ZERO
2534	D29F 34 Ø6		PSHS	В, А	SAVE TRACKS WRITTEN AND REMAINING COUNTERS ON STACK
2535 2536		* AT TH	ITS DOTN	T THE STACK HAS THE FOLLOWING DAT	A ON IT.
2537				REMAINING COUNTER; 1,S = TRACKS W	
2538				OF TRACKS WHICH FIT IN RAM; 3,S	
2539		* 4,5 =		DRIVE NUMBER	
2540	D2A1 73 Ø9 5C	. 5044		DRESFL	SET THE DISK RESET FLAG TO CAUSE A RESET
2541 2542	D2A4 5F D2A5 5C	LD2A4 LD2A5	CLRB INCB		INITIALIZE WRITE TRACK COUNTER TO ZERO ADD ONE TO WRITE TRACK COUNTER
2542	D2A5 5C D2A6 6A E4	LUZAS		,\$	* DECREMENT REMAINING TRACKS COUNTER
2544	D2A8 27 Ø4		BEQ		* AND BRANCH IF NO TRACKS LEFT
2545	D2AA E1 62		CMPB		= COMPARE WRITE TRACK COUNTER TO NUMBER OF TRACKS THAT
2546	D2AC 26 F7		BNE		= WILL FIT IN RAM AND BRANCH IF ROOM FOR MORE TRACKS IN RAM
2547	D2AE D7 Ø3	LD2AE		TMPLOC	SAVE THE NUMBER OF TRACKS TO BE TRANSFERRED
2548	D2BØ E6 64			\$Ø4,S	GET SOURCE DRIVE NUMBER
2549 255Ø	D2B2 8D 48 D2B4 86 FF		BSR LDA		FILL RAM BUFFER WITH TMPLOC TRACKS OF DATA SET SOURCE/DESTINATION FLAG TO DESTINATION
	> D2B6 BD D3 22		JSR		PRINT PROMPT MESSAGE IF NEEDED
2552	D2B9 E6 63		LDB	\$Ø3,S	GET DESTINATION DRIVE NUMBER
2553	D2BB 8D 42		BSR		WRITE TMPLOC TRACKS FROM BUFFER
2554	D2BD 6D E4			,\$	TEST TRACKS REMAINING FLAG
2555 2556	D2BF 27 ØC D2C1 4F		BEQ CLRA	LDZCD	BRANCH IF BACKUP DONE SET SOURCE/DESTINATION FLAG TO SOURCE
	> D2C1 4F > D2C2 BD D3 22		JSR	1 0322	PRINT PROMPT MESSAGE IF NEEDED
2558	D2C5 E6 61		LDB		* GET THE TRACKS WRITTEN COUNTER, ADD THE NUMBER OF
2559	D2C7 DB Ø3			TMPLOC	* TRACKS MOVED THIS TIME THROUGH LOOP AND
2560	D2C9 E7 61		STB		* SAVE THE NEW TRACKS WRITTEN COUNTER
2561	D2CB 20 D7		BRA	LD2A4	COPY SOME MORE TRACKS
2562 2563	D2CD 8D Ø3	LD2CD	BSR	1 0202	CHECK FOR DOS INITIALIZATION
2564	D2CF 7E AC 73	20200	JMP		JUMP BACK TO BASIC S MAIN LOOP
2565					
	D2D2 35 40	LD2D2	PULS		PUT THE RETURN ADDRESS IN U
2567	D2D4 B6 Ø9 5C			DRESFL	TEST DISK RESET FLAG
2568	D2D7 27 16 D2D9 8E 09 28		BEQ	LD2EF #FCBV1	DON T RESET THE DOS IF FLAG NOT SET
2569 257Ø	D2D9 8E 09 28 D2DC 4F		CLRA	#FUDV1	POINT X TO TABLE OF FCB ADDRESSES SET FILE COUNTER TO ZERO
2571	D2DD 6F 91	LD2DD		[,X++]	MARK FCB AS CLOSED
2572	D2DF 4C		INCA		ADD ONE TO FILE COUNTER
2573	D2EØ B1 Ø9 5B			FCBACT	COMPARE TO NUMBER OF RESERVED FILES
2574	D2E3 23 F8		BLS		BRANCH IF ANY FILES NOT SHUT DOWN
	D2E5 9E 19 D2E7 6F 1F		LDX CLR	TXTTAB	LOAD X WITH THE START OF BASIC SET FIRST BYTE OF BASIC PROGRAM TO ZERO
2576	D2E7 6F 1F D2E9 BD AD 19		JSR		GO DO A 'NEW'
				DRESFL	RESET THE DOS RESET FLAG
2579	D2EF B6 Ø9 5D	LD2EF		DLODFL	* CHECK THE LOAD RESET FLAG AND
2580	D2F2 27 Ø6			LD2FA	* BRANCH IF NOT SET
2581	D2F4 7F Ø9 5D			DLODFL	CLEAR THE LOAD RESET FLAG
2582	D2F7 BD AD 19	10054	JSR		GO DO A 'NEW'
2583 2584	D2FA 6E C4	LD2FA	JMP	,U	JUMP BACK TO RETURN ADDRESS SAVED IN U ABOVE
	D2FC 86 Ø2	LD2FC	LDA	#\$Ø2	READ OP CODE
2586	D2FE 8C	_52.10		#\$86Ø3	SKIP TWO BYTES
2587	D2FF 86 Ø3	LD2FF	LDA	#\$Ø3	WRITE OP CODE
2588	D3Ø1 DD EA		STD		SAVE IN DSKCON VARIABLE
2589 2590	D3Ø3 A6 63		LDA		* GET THE NUMBER OF THE TRACK BEING CURRENTLY  * WITTEN AND SAVE IT IN DSYCON VARIABLE
259Ø 2591	D3Ø5 97 EC D3Ø7 8E Ø9 89		STA LDX	#DFLBUF	* WRITTEN AND SAVE IT IN DSKCON VARIABLE = TRACK BUFFER STARTS AT DFLBUF
2592			STX		= SAVE IT IN DSKCON VARIABLE
–					

2594 2595	D30C 96 03 D30E C6 01 D310 D7 ED D312 BD D6 F2	LD3ØE L LD31Ø S	ISR LD6F2	2	GET NUMBER OF TRACKS TO MOVE INITIALIZE SECTOR COUNTER TO ONE SAVE DSKCON SECTOR VARIABLE READ/WRITE A SECTOR
	D315 ØC EE D317 5C	I	NC DCBP	ī	MOVE BUFFER POINTER UP ONE SECTOR (256 BYTES) INCREMENT SECTOR COUNTER
2599	D318 C1 12	C	MPB #SEC	1AX	COMPARE TO MAXIMUM NUMBER OF SECTORS PER TRACK
2600	D31A 23 F4	В	BLS LD316 INC DCTRI DECA	,	BRANCH IF ANY SECTORS LEFT
2602	D31C ØC EC D31E 4A	D	ECA		INCREMENT TRACK COUNTER VARIABLE TO NEXT TRACK DECREMENT TRACKS TO MOVE COUNTER
2603	D31F 26 ED D321 39	В	BNE LD3ØI RTS		READ MORE TRACKS IF ANY LEFT
26Ø4 26Ø5					
2606 2607	D322 E6 65 D324 E1 66		DB \$05,5 CMPB \$06,5		* GET THE DESTINATION DRIVE NUMBER AND * COMPARE IT TO THE SOURCE DRIVE NUMBER
2608					
2609 2610		* PRINT SC LD326 B		INATION DISK SWITCH PROMPT M :	MESSAGE RETURN IF DRIVE NUMBERS NOT EQUAL
2611	D328 7F Ø9 85	C	LR RDYTI	1R	RESET THE READY TIMER
	D32B 7F FF 4Ø D32E 7F Ø9 86	0	CLR DSKRI CLR DRGRA	:G AM	CLEAR DSKREG - TURN OFF ALL DISK MOTORS CLEAR DSKREG RAM IMAGE
	D331 34 Ø2	P	SHS A		SAVE SOURCE/DESTINATION FLAG ON STACK
2615	D333 BD A9 28 D336 8E D3 5F	J L	.DX #LD3!	3 5F	CLEAR SCREEN POINT X TO 'INSERT SOURCE' MESSAGE
2617	D339 C6 ØD	L	DB #13		13 BYTES IN MESSAGE
2618	D331 34 02 D333 BD A9 28 D336 8E D3 5F D339 C6 0D D33B A6 EØ D33D 27 05 D342 C6 12 D344 BD B9 A2 D347 8E D3 7F	L B	.DA ,5+ BEQ LD344	1	GET SOURCE/DESTINATION FLAG FROM THE STACK BRANCH IF SOURCE
2620	D33F 8E D3 6C	L	DX #LD36	5C	POINT X TO 'INSERT DESTINATION' MESSAGE
2622	D344 BD B9 A2	LD344 J	ISR LB9A	2	18 BYTES IN MESSAGE SEND MESSAGE TO CONSOLE OUT
2624	D347 8E D3 7E D34A C6 1B	L L J	DX #LD37	<sup>7</sup> E	POINT X TO 'DISKETTE AND' MESSAGE 27 BYTES IN MESSAGE
2625	D34C BD B9 A2 D34F CC 64 Ø5	J	ISR LB9A	2	SEND MESSAGE TO CONSOLE OUT
2626 2627	D34F CC 64 Ø5 D352 97 8C	L	DD #\$640 STA SNDTO	95 N	* SET UP 'SOUND' PARAMETERS  * FOR A BEEP
	D354 BD A9 51 D357 BD A1 71				JUMP TO 'SOUND' - DO A BEEP
2629 2630	D357 BD A1 71 D35A 81 ØD	LD357 J	ISR LA17: CMPA #CR	1	GET A CHARACTER FROM CONSOLE IN * KEEP LOOKING AT CONSOLE IN UNTIL
2631	D35C 26 F9	В	NE LD35	7	* YOU GET A CARRIAGE RETURN
2632 2633	D35E 39	LD45E R	RTS		
2635	D35F 49 4E 53 45 52 54 D365 2Ø 53 4F 55 52 43 D36B 45		CC 'INSI	ERT SOURCE'	
2637 2638	D36C 49 4E 53 45 52 54 D372 2Ø 44 45 53 54 49	)	CC 'INSI	ERT DESTINATION'	
	D378 4E 41 54 49 4F 4E D37E 2Ø 44 49 53 4B 45		CC ' DIS	SKETTE AND'	
2641	D384 54 54 45 20 41 4E				
	D38A 44 D38B ØD	F	CB CR		
2644 2645	D38C 5Ø 52 45 53 53 2Ø D392 27 45 4E 54 45 52		CC 'PRES	SS 'ENTER''	
2646	D398 27	•			
2647 2648		* PUSH FIL	LENAME.EXT	AND DRIVE NUMBER ONTO THE S	STACK
2649	D399 35 2Ø D39B C6 ØB	10200 0	uu e v		SAVE RETURN ADDRESS IN Y
2651	D30D 0F 00 F7	L	.DX #DNAN	1BF+11	11 CHARACTERS IN FILENAME AND EXTENSION POINT X TO TOP OF DISK NAME/EXT BUFFER
2652	D3AØ A6 82 D3A2 34 Ø2	LD3AØ L	DA ,-X	MBF+11	* GET A CHARACTER FROM FILENAME.
2654	D3A4 5A	г	ECB		* EXT BUFFER AND PUSH IT ONTO THE  * STACK - DECREMENT COUNTER AND
2655	D3A5 26 F9		BNE LD3A( LDA DCDR)		* KEEP LOOPING UNTIL DONE = GET DRIVE NUMBER AND PUSH
2657	D3A2 34 02 D3A4 5A D3A5 26 F9 D3A7 96 EB D3A9 34 02	P	SHS A		= IT ONTO THE STACK
2658 2659	D3AB 6E A4	J	IMP ,Y		PSEUDO - RETURN TO CALLING ROUTINE
2660	D2AD A6 00			AND DRIVE NUMBER FROM (X) T	
2661 2662	D3AD A6 8Ø D3AF 97 EB		.DA ,X+ STA DCDR\	ı	* GET DRIVE NUMBER AND SAVE * IT IN DSKCON VARIABLE
2663	D3B1 C6 ØB	L	DB #11		11 BYTES IN FILENAME AND EXTENSION POINT U TO DISK NAME BUFFER
2664 2665	D3B3 CE Ø9 4C D3B6 7E A5 9A		.DU #DNAN IMP LA59/		MOVE FILENANE.EXT FROM (X) TO DNAMBF
2666 2667		* COPY			
2668			Y PROCESS	IS PERFORMED BY COPYING DATA	FROM THE SOURCE FILE
2669 2670				OPYING IT TO THE DESTINATION ARE OPENED AS RANDOM FILES	
2671		* FCB ABOV	VE THE RES	ERVED FCBS. ALL OF AVAILABLE	FREE RAM ABOVE THE
2672 2673				) AS A COPY BUFFER WHICH SPEE THE METHOD USED WILL ALLOW	
2674		* THE COPY	Y PROCESS	TO 'HANG' THE SYSTEM. THIS I	S CAUSED BY POINTING THE FCB'S
2675 2676				R POINTER (FCBBUF,X) TO THE HE OPEN FILE TO BE CLOSED WI	
2677		* AREA IN	RAM WHERE		SE ROUTINE (LCAE2) WILL NEVER
2678 2679	D3B9 BD C9 35	* LOOK FOR	K II ISR LC93!	5	* GET SOURCE FILENAME.EXT & DRIVE NUMBER FROM BASIC
2680	D3BC 8D DB D3BE 6F E2	В	SSR LD399		* AND SAVE THEM ON THE STACK
2681 2682	D3CØ 9D A5	J	CLR ,-S ISR GETCO		CLEAR A BYTE ON STACK - SINGLE DISK COPY (SDC) FLAG GET CURRENT INPUT CHARACTER
2683 2684	D3C2 27 ØA D3C4 63 E4		BEQ LD3CI		BRANCH IF END OF LINE - SINGLE DISK COPY SET SOC FLAG TO \$FF (NO SINGLE DISK COPY)
2685	D3C6 C6 A5	L	DB #\$A5		TOKEN FOR 'TO'
2686 2687	D3C8 BD B2 6F D3CB BD C9 35		ISR LB261 ISR LC93!		SYNTAX CHECK FOR 'TO' GET DESTINATION FILENAME.EXT AND DRIVE NUMBER
2688	D3CE 8D C9		SSR LD399		SAVE DESTINATION FILENAME.EXT & DRIVE NUMBER ON STACK

2689 269Ø	D3DØ BD A5 C7 D3D3 BD CA E9		JSR LA5C7 JSR DVEC7	SYNTAX ERROR IF MORE CHARACTERS ON LINE CLOSE ALL FILES
2691 2692		* COUNT	THE NUMBER OF SECTORS WORTH	DE EDEE DAM AVATIADIE
2693	D3D6 6F E2	COOM	CLR ,-S	CLEAR A SECTOR COUNTER ON THE STACK
2694 2695	D3D8 3Ø E9 FF ØØ D3DC 6C E4	LD3DC	LEAX -SECLEN,S INC ,S	POINT X ONE SECTOR LENGTH DOWN FROM THE TOP OF STACK INCREMENT SECTOR COUNTER
2696	D3DE 30 89 FF 00	LUSUC	LEAX -SECLEN,X	DECREMENT X BY ONE SECTOR
2697 2698	D3E2 9C 1F D3E4 24 F6		CMPX ARYEND BHS LD3DC	COMPARE TO TOP OF ARRAYS
2699	D3E4 24 F6 D3E6 6A E4		DEC ,S	BRANCH IF NOT AT BOTTOM OF FREE RAM DECREMENT SECTOR COUNTER
2700 2701	D3E8 10 27 D8 58		LBEQ LAC44	'OM' ERROR IF NOT AT LEAST ONE FULL SECTOR OF FREE RAM
2701	D3EC 3Ø 6E D3EE 8D BD		LEAX 14,S BSR LD3AD	POINT X TO START OF SOURCE DATA PUT SOURCE DATA INTO DNAMBF AND DSKCON
2703	D3FØ BD C6 8C		JSR LC68C	SCAN DIRECTORY FOR A MATCH
27Ø4 27Ø5	D3F3 BD C6 E5 D3F6 BE Ø9 74		JSR LC6E5 LDX V974	'NE' ERROR IF MATCH NOT FOUND POINT X TO DIRECTORY RAM IMAGE OF FOUND FILE
27Ø6 27Ø7	D3F9 EE ØE D3FB AE ØB		LDU DIRLST,X LDX DIRTYP,X	* GET NUMBER OF BYTES IN LAST SECTOR AND * SOURCE FILE TYPE AND ASCII FLAG
2707	D3FD 34 5Ø		PSHS U,X	* AND SAVE THEM ON THE STACK
27Ø9 271Ø	D3FF BD C7 9D		JSR LC79D LDB V976	GET VALID FAT DATA
2710	D402 F6 09 76 D405 BD CD 1E		JSR LCD1E	GET NUMBER OF FIRST GRANULE IN FILE * GET THE NUMBER OF GRANULES IN FILE
2712	D4Ø8 34 Ø2		PSHS A	* AND SAVE IT ON THE STACK
2713 2714	D4ØA 4A D4ØB C4 3F		DECA ANDB #\$3F	SUBTRACT OFF THE LAST GRANULE  * MASK OFF LAST GRANULE FLAG BITS AND SAVE THE
2715	D4ØD 34 Ø4		PSHS B	* NUMBER OF SECTORS IN LAST GRANULE ON STACK
2716 2717	D4ØF 1F 89 D411 4F		TFR A,B CLRA	SAVE THE NUMBER OF GRANULES IN ACCB CLEAR THE MS BYTE OF ACCD
2718	D412 BD C7 79 D415 EB E4		JSR LC779	MULTIPLY ACCD BY NINE
2719 2720	D415 EB E4 D417 89 ØØ		ADDB ,S ADCA #\$00	* ADD THE NUMBER OF SECTORS IN THE LAST  * GRANULE TO ACCD
2721 2722	D419 8E 00 01 D41C 34 16		LDX #\$0001 PSHS X,B,A	INITIALIZE RECORD COUNTER TO ONE INITIALIZE SECTOR AND RECORD COUNTERS ON THE STACK
2723	D41C 34 10		r3n3 A,D,A	INTITALIZE SECTOR AND RECORD COUNTERS ON THE STACK
2724 2725			S POINT THE CONTROL VARIABLE S = REMAINING SECTORS COUNTE	FOR COPY ARE STORED ON THE STACK.
2726		* 4,	S = NUMBER OF SECTORS TO BE	COPIED. INITIALLY SET TO NUMBER OF
2727 2728		* * 5.	SECTORS IN THE LAST GRAN	JLE. / SET TO NUMBER OF GRANS IN FILE
2729		* 6,	S = FILE TYPE; 7,S = ASCII F	AG; 8 9,S = NUMBER OF BYTES IN LAST SECTOR
273Ø 2731			S = NUMBER OF SECTORS WHICH S = DESTINATION FILENAME.EXT	VILL FIT IN THE CURRENTLY AVAILABLE FREE RAM
2732		* 23,	S = SINGLE DISK COPY FLAG; 2	I-35,S = SOURCE FILENAME.EXT AND DRIVE NUMBER
2733 2734	D41E 5F D41F AE E4	LD41E	CLRB LDX ,S	SET SECTOR COUNTER TO ZERO GET THE NUMBER OF SECTORS REMAINING IN THE FILE
2735	D421 27 Ø9		BEQ LD42C	BRANCH IF NO SECTORS LEFT
2736 2737	D423 5C D424 3Ø 1F	LD423	INCB LEAX -1,X	ADD A SECTOR TO TEMPORARY SECTOR COUNTER DECREMENT REMAINING SECTORS COUNTER
2738	D426 27 Ø4		BEQ LD42C	BRANCH IF NO SECTORS LEFT
2739 2740	D428 E1 6A	*	CMPB 10,S	*COMPARE TEMPORARY COUNTER TO NUMBER OF SECTORS WHICH MAY *BE STORED IN FREE RAM
2741	D42A 26 F7		BNE LD423	BRANCH IF STILL ROOM FOR MORE SECTORS
2742 2743	D42C AF E4 D42E E7 64	LD42C	STX ,S STB \$04,S	SAVE THE NUMBER OF UNCOPIED SECTORS REMAINING IN THE FILE SAVE THE NUMBER OF SECTORS TO BE COPIED THIS TIME THROUGH LOOP
2744	D43Ø 8D 5Ø		BSR LD482	'GET' ACCB SECTORS TO RAM BUFFER
2745 2746	D432 86 FF D434 8D 4Ø		LDA #\$FF BSR LD476	SET SOURCE/DESTINATION FLAG TO DESTINATION PRINT PROMPT MESSAGE IF REQUIRED
2747	D436 6D 65		TST \$05,S	* CHECK THE GRAN TEST FLAG. IF ⇔ Ø, IT CONTAINS THE
2748 2749	D438 27 25	*	BEQ LD45F	* NUMBER OF GRANS IN THE FILE AND THE DESTINATION DISK * MUST BE CHECKED FOR ENOUGH ROOM. IF IT IS =Ø
2750	D404 00 CD	*	1517 44 0	* THEN THE CHECK HAS ALREADY BEEN DONE
2751 2752	D43A 3Ø 6B D43C BD D3 AD		LEAX 11,S JSR LD3AD	POINT TO DESTINATION FILE PARAMETERS GET DESTINATION FILE PARAMETERS FROM STACK
2753	D43F BD DØ 59		JSR LDØ59	SCAN DIRECTORY FOR FILE - 'AE' ERROR IF IT EXISTS
2754 2755	D442 BD C7 9D		JSR LC79D	GET VALID FAT DATA
2756 2757	D445 BD C7 55	* MAKE S	URE THERE ARE ENOUGH FREE GR JSR LC755	NULES ON THE DESTINATION DISK POINT X TO FAT
2758	D448 3Ø Ø6		LEAX FATCON,X	SKIP PAST THE FAT CONTROL BYTES
2759 276Ø	D44A A6 65 D44C C6 44		LDA \$05,S LDB #GRANMX	GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM
2761	D44E 63 84	LD44E	COM ,X	* CHECK TO SEE IF A BRAN IS FREE
2762 2763	D450 26 03 D452 4A		BNE LD455 DECA	* AND BRANCH IF IT IS NOT FREE = DECREMENT COUNTER AND BRANCH IF
2764	D453 27 Ø8		BEQ LD45D	= THERE ARE ENOUGH FREE GRANULES
2765 2766	D455 63 8Ø D457 5A	LD455	COM ,X+ DECB	RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER
2767	D458 26 F4		BNE LD44E	BRANCH IF ALL GRANS NOT CHECKED
2768 2769	D45A 7E C7 F8 D45D 63 84	LD45D	JMP LC7F8 COM ,X	'DISK FULL' ERROR RESTORE FAT BYTE
2770	D45F 8D 1B	LD45F	BSR LD47C	'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE
2771 2772	D461 AE E4 D463 27 ØD		LDX ,S BEQ LD472	GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT
2773	D465 EC 62		LDD \$02,S	*
2774 2775	D467 EB 64 D469 89 ØØ		ADDB \$04,S ADCA #\$00	* GET THE CURRENT RECORD COUNTER, ADD * THE NUMBER OF SECTORS (RECORDS) MOVED
2776	D46B ED 62		STD \$02,S	* AND SAVE THE NEW RECORD COUNTER
2777 2778	D46D 4F D46E 8D Ø6		CLRA BSR LD476	SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED
2779	D47Ø 2Ø AC		BRA LD41E	KEEP COPYING SECTORS
278Ø 2781	D472 32 E8 24	LD472	LEAS 36,S	REMOVE TEMPORARY STORAGE VARIABLES FROM STACK
2782	D475 39		RTS	**** COPY DONE ****
2783 2784	D476 6D E8 19	LD476	TST 25,S	*CHECK SINGLE DISK COPY FLAG - IF ⇔ ZERO, THEN DON'T

2785 2786	D479 7E D3 26	*	JMP	LD326	*PRINT THE PROMPT MESSAGE PRINT THE PROMPT MESSAGE IF REQUIRED
2787	04/9 /2 03 20				
2788 2789	D47C 86 FF	* 'PUT'. LD47C	'GET' LDA	DATA FROM THE DESTINATION/SOURCE #\$FF	FILES 'PUT' FLAG
2790	D47E 3Ø 6D	LD470		13,5	POINT X TO DESTINATION FILENAME DATA
2791 2792	D48Ø 2Ø Ø4 D482 4F	LD482		LD486	GO 'PUT' SOME DATA ZERO IS THE 'GET' FLAG
2792	D483 3Ø E8 1A	LU462	CLRA LEAX	26,5	POINT X TO THE SOURCE FILENAME DATA
2794	D486 97 D8	LD486	STA		SAVE THE 'GET'/'PUT' FLAG
2795 2796	D488 BD D3 AD D48B AE 68			LD3AD \$08,S	GET FILENAME AND DRIVE DATA FROM THE STACK * GET ASCII FLAG AND FILE TYPE AND SAVE
2797	D48D BF Ø9 57		STX	DFLTYP	* THEM IN THE DISK RAM VARIABLES
2798 2799	D490 8E 01 00 D493 BF 09 7C			#SECLEN DFFLEN	= SAVE ONE SECTOR LENGTH IN = RAM RECORD LENGTH VARIABLE
2800	D496 86 52		LDA	#'R'	RANDOM FILE TYPE FLAG
2801	D498 F6 Ø9 5B			FCBACT	* GET THE HIGHEST RESERVED FCB NUMBER, ADD ONE * AND OPEN A RANDOM FILE WHOSE FCB WILL BE ONE ABOVE
28Ø2 28Ø3	D49B 5C D49C BD C4 8D		INCB JSR	LC48D	* THE HIGHEST RESERVED FCB (THE SYSTEM FCB)
28Ø4	D49F 9E F1			FCBTMP	POINT X TO THE 'SYSTEM' FCB
28Ø5 28Ø6	D4A1 CC Ø1 ØØ D4A4 ED 88 13			#SECLEN FCBLST,X	* SET THE NUMBER OF BYTES IN THE LAST SECTOR * OF THE FILE EQUAL TO ONE SECTOR LENGTH
2807	D4A7 E6 66			\$Ø6,S	=GET THE NUMBER OF SECTORS TO MOVE AND
28Ø8 28Ø9	D4A9 27 29 D4AB D6 D8			LD4D4 VD8	=BRANCH IF NONE LEFT
2810	D4AD E4 67			\$07,S	*GRAB THE 'GET'/'PUT' FLAG, 'AND' IT WITH THE *GRAN TEST FLAG - BRANCH IF 'GET'ING DATA OR THIS IS
2811	D4AF 27 Ø9			LD4BA	*NOT THE FIRST TIME THROUGH THE LOOP
2812 2813	D4B1 EC 62 D4B3 EB 66			\$02,S \$06,S	=GET THE NUMBER OF SECTORS REMAINING TO BE COPIED AND =ADD THE NUMBER TO BE COPIED THIS TIME THROUGH LOOP
2814	D4B5 89 ØØ		ADCA	#\$00	=
2815 2816	D4B7 BD C2 E6	*	JSR	LC2E6	*'PUT' THE LAST RECORD IN THE FILE TO THE SYSTEM FCB.  *THE RECORD NUMBER IS IN ACCD.
2817	D4BA 9E F1	LD4BA	LDX	FCBTMP	POINT X TO THE SYSTEM FCB
2818 2819	D4BC EE 64		LDU STU	\$Ø4,S	* GET THE CURRENT RECORD NUMBER * AND SAVE IT IN THE FCB
2820	D4BE EF Ø7 D4CØ E6 66			FCBREC,X \$06,S	GET THE NUMBER OF THE RECORD (SECTOR) TO MOVE
2821	D4C2 DE 1F		LDU	ARYEND	END OF ARRAYS IS THE START OF THE COPY FREE RAM BUFFER
2822 2823	D4C4 34 44 D4C6 9E F1	LD4C4	PSHS LDX	U,B FCBTMP	SAVE SECTOR COUNTER AND BUFFER POINTER ON THE STACK POINT X TO SYSTEM FCB
2824	D4C8 EF ØB		STU	FCBBUF,X	*SET THE RANDOM FILE BUFFER POINTER TO THE 'COPY' RAM BUFFER
2825 2826	D4CA BD C2 EA	*	JSR	LC2EA	*THIS WILL CAUSE THE SYSTEM TO 'HANG' IF AN ERROR OCCURS DURING COPY. GO 'GET' OR 'PUT' DATA TO THE SYSTEM FCB
2827	D4CD 6C 61			\$Ø1,S	ADD 256 (ONE SECTOR) TO THE BUFFER POINTER
2828	D4CF 35 44		PULS	B,U	GET THE SECTOR COUNTER AND BUFFER POINER
2829 283Ø	D4D1 5A D4D2 26 FØ		DECB BNE	LD4C4	DECREMENT SECTOR COUNTER BRANCH IF ALL SECTORS NOT DONE
2831	D4D4 9E F1	LD4D4	LDX	FCBTMP	POINT X TO SYSTEM FCB
2832 2833	D4D6 CE Ø9 89 D4D9 EF ØB		LDU STU	#DFLBUF FCBBUF,X	* RESET THE RANDOM FILE BUFFER POINTER FOR THE SYSTEM * FCB TO THE BOTTOM OF RANDOM FILE BUFFER AREA
2834	D4DB D6 D8		LDB	VD8	=GRAB THE 'GET'/'PUT' FLAG, 'AND' IT WITH THE GRAN
2835	D4DD E4 67			\$07,S	=TEST FLAG - CLOSE THE FILE IF 'GET'ING DATA AND
2836 2837	D4DF 27 Ø9 D4E1 6F 67			LD4EA \$07,S	=THIS IS NOT THE FIRST TIME THROUGH THE LOOP RESET THE GRAN TEST FLAG IF FIRST TIME THROUGH LOOP
2838	D4E3 EC 6A		LDD	10,5	*GET THE NUMBER OF BYTES IN THE LAST SECTOR,
2839 284Ø	D4E5 8A 8Ø D4E7 ED 88 13		ORA STD	#\$80 FCBLST,X	*'OR' IN THE PRE-SAVED FLAG AND *SAVE THE NUMBER OF BYTES IN THE LAST SECTOR IN THE FCB
2841	D4EA 7E CB Ø6	LD4EA	JMP	LCBØ6	CLOSE THE FILE
2842 2843		* DSKI\$	CUMMVI	ND.	
2844	D4ED 8D 38	DSKI		LD527	GET THE DRIVE, TRACK AND SECTOR NUMBERS
2845	D4EF 8D 2B			LD51C	* EVALUATE STRING VARIABLE 1 AND SAVE
2846 2847	D4F1 34 10 D4F3 8D 27		PSHS BSR	LD51C	* THE DESCRIPTOR ADDRESS ON THE STACK = EVALUATE STRING VARIABLE 2 AND SAVE
2848	D4F5 34 1Ø		PSHS	X	= THE DESCRIPTOR ADDRESS ON THE STACK
2849 285Ø	D4F7 C6 Ø2 D4F9 BD D5 8F			#\$Ø2 LD58F	DSKCON READ OP CODE REAO A SECTOR INTO DBUFØ
2851	D4FC CE Ø6 8Ø		LDU	#DBUFØ+128	POINT U TO TOP HALF OF DBUFØ
2852 2853	D4FF 35 10 D501 8D 05		PULS BSR	X LD5Ø8	GET STRING 2 DESCRIPTOR ADDRESS PUT STRING 2 INTO STRING SPACE
2854	D503 CE 06 00		LDU	#DBUFØ	POINT U TO BOTTOM HALF OF DBUFØ
2855 2856	D506 35 10 D508 34 50	LD5Ø8	PULS PSHS		GET STRING 1 DESCRIPTOR ADDRESS PUT STRING DESCRIPTOR & SOURCE POINTER ON THE STACK
2855	D5ØA C6 8Ø	FDSNQ	LDB	#128	*
2858	D5ØC BD B5 ØF		JSR	LB5ØF	* RESERVE 128 BYTES IN STRING SPACE
2859 286Ø	D50F 33 84 D511 35 10		LEAU PULS		POINT U TO RESERVED STRING SPACE GET STRING DESCRIPTOR ADDRESS
2861	D513 E7 84		STB	,Х	* SAVE DESCRIPTOR DATA (LENGTH AND ADDRESS)
2862 2863	D515 EF Ø2 D517 35 1Ø		STU PULS	\$Ø2,X X	* OF THE NEW STRING GET THE SOURCE (DBUFØ) POINTER
2864	D517 35 10 D519 7E A5 9A	LD519		LA59A	MOVE SECTOR DATA FROM DBUFØ TO STRING SPACE
2865 2866	D51C BD B2 6D	LD51C	JSR	SYNCOMMA	SYNTAX CHECK FOR A COMMA
2867	D51F 8E B3 57	LDSIC		#LB357	POINT X TO EVALUATE VARIABLE ROUTINE
2868	D522 8D 2F			LD553	EVALUATE A VARIABLE
2869 287Ø	D524 7E B1 46	LD524	JMP	LB146	'TM' ERROR IF NUMERIC VARIABLE
2871				IVE, TRACK AND SECTOR NUMBERS	
2872 2873	D527 BD B7 ØB D52A C1 Ø3	LD527		EVALEXPB #\$Ø3	EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) -
2873	D52C 22 1C			#\$03 LD54A	* 'FC' ERROR IF IT S > 3
2875	D52E 34 Ø4		PSHS	В	SAVE DRIVE NUMBER ON THE STACK
2876 2877	D53Ø BD B7 38 D533 C1 22			LB738 #TRKMAX-1	SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER
2878	D535 22 13		BHI	LD54A	* 'FC' ERROR IF TRACK NUMBER > 34
2879 288Ø	D537 34 Ø4 D539 BD B7 38		PSHS JSR	B LB738	SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER)
_502	00				The state of the s

2881 2882 2883 2884 2885 2886 2887 2888 2889 289Ø	D53C D7 ED D53E 5A D53F C1 11 D541 22 Ø7 D543 35 Ø6 D545 97 EC D547 D7 EB D549 39 D544 7E B4 4A	LD54A	STB DSEC DECB CMPB #SECMAX-1 BHI LD54A PULS A,B STA DCTRK STB DCDRV RTS JMP LB44A	SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER TOO BIG * GET TRACK AND DRIVE NUMBER OFF OF * THE STACK AND SAVE IN DSKCON * VARIABLES  JUMP TO 'FC' ERROR
2891 2892 2893 2894 2895 2896 2897 2898 2899 2900 2901	D54D BD B2 6D D550 8E B1 56 D553 D6 EB D555 DE EC D557 34 44 D559 AD 84 D558 B5 44 D55B D7 EB D55F DF EC	LD54D LD553	JSR SYNCOMMA LDX #LB156 LDB DCDRV LDU DCTRK PSHS S,B JSR ,X PULS B,S STB DCDRV STU DCTRK RTS	SYNTAX CHECK FOR COMMA POINT X TO 'EVALUATE EXPRESSION' ROUTINE ADDRESS * GET THE DSKCON DRIVE, TRACK AND * SECTOR VALUES AND SAVE THEM ON THE STACK * GO EVALUATE AN EXPRESSION OR A VARIABLE * GET THE DRIVE, TRACK AND SECTOR * NUMBERS OFF OF THE STACK AND PUT * THEM BACK INTO THE DSKCON VARIABLES
2902 2903 2904 2905 2906 2907 2908 2909 2910 2911 2912 2913	D562 8D C3 D564 8D E7 D566 8D BC D568 9E 52 D56A 34 10 D56C 8D DF D56E BD B6 54 D571 34 14 D573 5F D574 8E Ø6 Ø0 D577 6F 80	DSKO	COMMAND BSR LD527 BSR LD540 BSR LD524 LDX FPAG+2 PSHS X BSR LD54D JSR LB654 PSHS X,B CLRB LDX #DBUFØ	GET THE DRIVE, TRACK AND SECTOR NUMBERS GET THE DESCRIPTOR OF STRING 1 'TM' ERROR IF NUMERIC EXPRESSION * GET STRING 1 DESCRIPTOR ADDRESS * AND SAVE IT ON THE STACK GET THE DESCRIPTOR OF STRING 2 *GET LENGTH AND ADDRESS OF STRING 2 AND *SAVE THEM ON THE STACK SET CLEAR COUNTER TO 256 (FULL SECTOR BUFFER) USE DBUFF AS THE DSKOS I/O BUFFER
2913 2914 2915 2916 2917 2918 2919 2920 2921 2922 2923 2924 2925 2926 2927	D579 5A D578 26 FB D57C 35 14 D57E CE Ø6 8Ø D581 8D 96 D583 35 1Ø D585 BD B6 59 D588 CE Ø6 ØØ D588 BD 8C D580 CE Ø6 ØØ D588 BB 8C D580 CE Ø6 ØØ D580 CE Ø6 ØØ D580 CE Ø6 ØØ D592 9F EE D594 D7 EA	LD577	CLR ,X+ DECB BNE LD577 PULS B,X LDU #DBUFØ+128 BSR LD519 PULS X JSR LB659 LDU #DBUFØ BSR LD519 LDB #\$Ø3 LDX #DBUFØ STX DCBPT STB DCOPC JMP LD6F2	CLEAR A BYTE IN 1/O BUFFER DECREMENT CLEAR COUNTER BRANCH IF ALL 256 BYTES NOT CLEARED GET THE LENGTH AND ADDRESS OF STRING 2 POINT X TO STRING 2 DESTINATION MOVE STRING 2 DATA INTO DBUFØ POINT X TO STRING 1 DESCRIPTOR GET THE LENGTH AND ADDRESS OF STRING 1 POINT U TO STRING 1 DESTINATION MOVE STRING 1 DATA INTO DBUFØ DSKCON WRITE OP CODE POINT X TO 1/O BUFFER (DBUFØ)  *  * SAVE NEW DSKCON BUFFER POINTER AND OP CODE VARIABLES GO WRITE OUT A SECTOR
2928 2929 2930 2931 2932 2933 2934 2935 2936 2937 2938 2940 2941 2942 2943 2944 2945 2944 2945	D599 10 27 D0 82 D599 BD D2 56 D5A0 C6 04 D5A2 9D A5 D5A4 27 0C D5A6 BD B7 38 D5A9 C1 11 D5AB 10 24 DE 9B D5AF BD A5 C7 D5B2 34 04 D5B4 8E 07 12 D5B7 C6 12 D5B9 6F 82 D5B9 6F 82 D5B9 6F 82 D5BB 5A D5BE 24 D5BB 5A D5BE 24 D5BB 5A		II COMMAND  LBEQ LA61F  JSR LD256  LDB #\$04  JSR GETCCH  BEQ LD5B2  JSR LB738  CMPB #17  LBHS LB44A  JSR LA5C7  PSHS B  LDX #DBUF1+SECMAX  LDB #SECMAX  CLR ,-X  DECB  BNE LD5B9  CLRA  BRA LD5CE	BRANCH TO 'DN' ERROR IF NO DRIVE NUMBER SPECIFIED CALCULATE DRIVE NUMBER SKIP FACTOR DEFAULT VALUE GET CURRENT INPUT CHAR FROM BASIC BRANCH IF END OF LINE SYNTAX CHECK FOR COMMA AND EVALUATE EXPRESSION MAX VALUE OF SKIP FACTOR = 16 'ILLEGAL FUNCTION CALL' IF BAD SKIP FACTOR SYNTAX ERROR IF MORE CHARACTERS ON THE LINE SAVE SKIP FACTOR ON THE STACK POINT TO END OF LOGICAL SECTOR NUMBER STORAGE AREA 18 SECTORS PER TRACK CLEAR A BYTE IN THE BUFFER CLEARED ALL 18? KEEP GOING IF NOT RESET PHYSICAL SECTOR COUNTER START WITH FIRST PHYSICAL SECTOR = 1
2947 2948 2949 2950 2951 2952 2953 2956 2956 2957 2960 2961 2962 2963 2964 2965 2966 2967 2966 2977 2969 2970 2971 2972 2973 2973	D5C1 EB E4 D5C3 5C D5C4 CØ 12 D5C6 24 FC D5C8 CB 12 D5CA 6D 85 D5CC 26 F5 D5CC 4C D5CF A7 85 D5D1 81 12 D5D3 25 EC D5D5 32 61 D5D7 8E 22 ØF D5DA 9C 27 D5DC 1Ø 22 D6 64 D5EØ 8D CA E9 D5EØ 37 3 Ø9 5C D5EØ 8D 9C 8C	LDSC1 LDSC3 LDSC4	LATE LOGICAL SECTOR NUMBERS ADDB ,S INCB SUBB #SECMAX	ADD SKIP FACTOR TO LOGICAL SECTOR COUNTER ADD ONE TO LOGICAL SECTOR COUNTER SUBTRACT MAX NUMBER OF SECTORS BRANCH UNTIL Ø > ACCB >= -18 ADD 18, NOW ACCB IS Ø-17 IS ANYTHING STORED HERE ALREADY? YES - GET ANOTHER SECTOR * INCREMENT PHYSICAL SECTOR NUMBER AND * SAVE IT IN THE RAM BUFFER FINISHED WITH ALL SECTORS? NO - KEEP GOING REMOVE SKIP FACTOR FROM STACK GET TOP OF RAM USED BY DSKINI IS IT > CLEARED AREA? 'OUT OF MEMORY' ERROR IF > CLEARED AREA

2977	D6Ø3 7E D6 88		JMP	LD688	ERROR IF DRIVES NOT READY
2978	D606 81 16	LD6Ø6	CMPA		= CHECK FOR TRACK 22 (PRECOMPENSATION)
2979	D608 25 08			LD612	= AND BRANCH IF < TRACK 22 - NO PRECOMP
2980	D6ØA B6 Ø9 86		LDA	DRGRAM	* GET THE RAM IMAGE OF DSKREG, 'OR'
2981	D6ØD 8A 1Ø		ORA	#\$10	* IN THE PRECOMPENSATION FLAG AND
2982	D60F B7 FF 40 D612 86 53	10612	STA	DSKREG #\$53	* SEND IT TO DSKREG
2983	D612 86 53 D614 B7 FF 48	LD612	LDA		= GET STEP IN COMMAND
2984 2985	D614 B7 FF 46 D617 1E 88		STA EXG	FDCREG A,A	= AND SEND IT TO THE 1793 * DELAY AFTER ISSUING COMMAND TO 1793
2986	D619 1E 88		EXG	A,A	* DELAT AFTER 13301NG COMMAND TO 1793
2987	D61B BD D7 D1			LD7D1	CHECK DRIVE READY
2988	D61E 26 68			LD688	BRANCH IF NOT READY - ISSUE AN ERROR
2989	D62Ø BD D7 FØ	LD62Ø		LD7FØ	WAIT A WHILE
2990	D623 8D 6C		BSR	LD691	BUILD A FORMATTED TRACK IN RAM
2991	D625 10 8E FF 4B		LDY	#FDCREG+3	Y POINTS TO 1793 DATA REGISTER
2992	D629 1A 5Ø		ORCC		DISABLE INTERRUPTS
2993	D62B 8E D6 4F		LDX	#LD64F	* GET RETURN ADDRESS AND STORE
2994	D62E BF Ø9 83		STX	DNMIVC	* IT IN THE NON MASKABLE INTERRUPT VECTOR
2995 2996	D631 8E Ø9 89		LDX LDA	#DFLBUF	POINT X TO THE FORMATTED TRACK RAM IMAGE RESET STATUS OF THE 1793
2997	D634 B6 FF 48 D637 86 FF		LDA	FDCREG #\$FF	* ENABLE THE NMI FLAG TO VECTOR
2998	D639 B7 Ø9 82		STA	NMIFLG	* OUT OF AN I/O LOOP UPON AN NMI INTERRUPT
2999	D63C C6 F4		LDB	#\$F4	= GET WRITE TRACK COMMAND AND
3000	D63E F7 FF 48			FDCREG	= SEND TO 1793
3001	D641 B6 Ø9 86		LDA	DRGRAM	* GET THE DSKREG RAM IMAGE AND 'OR' IN THE
3002	D644 8A 8Ø		ORA	#\$80	* FLAG WHICH WILL ENABLE THE 1793 TO HALT
3003	D646 B7 FF 40		STA	DSKREG	* THE 6809. SEND RESULT TO DSKREG
3004	D649 E6 8Ø	LD649	LDB	, χ+	= GET A BYTE FROM THE FORMATTED TRACK
3005	D64B E7 A4		STB	, Y	= RAM IMAGE, SEND IT TO THE 1793 AND
3006	D64D 20 FA		BRA	LD649	= LOOP BACK TO GET ANOTHER BYTE
3007 3008	D64F B6 FF 48	LD64F	I DA	FDCREG	GET STATUS
3008 3009	D64F B6 FF 48	LU041	ANDCC		GEL STATUS ENABLE INTERRUPTS
3010	D654 84 44		ANDA		* KEEP ONLY WRITE PROTECT & LOST DATA
3010	D656 97 FØ		STA	DCSTA	* AND SAVE IT IN THE DSKCON STATUS BYTE
3012	D658 26 2E			LD688	BRANCH IF ERROR
3013	D65A ØC EC		INC	DCTRK	SKIP TO THE NEXT TRACK
3014	D65C 96 EC		LDA	DCTRK	GET THE TRACK NUMBER
3Ø15	D65E 81 23		CMPA	#TRKMAX	WAS IT THE LAST TRACK
3Ø16	D660 26 A4		BNE	LD606	NO - KEEP GOING
3Ø17					
3Ø18		* VERIF		ALL SECTORS ARE READABLE	
3019	D662 86 Ø2			#\$02	= GET THE DSKCON READ OP CODE
3020	D664 97 EA		STA	DCOPC	= AND SAVE IT IN THE DSKCON VARIABLE
3Ø21 3Ø22	D666 8E Ø6 ØØ D669 9F EE		LDX STX	#DBUFØ DCBPT	* POINT THE DSKCON BUFFER POINTER * TO DBUFØ
3022	D66B CE Ø7 ØØ		LDU	#DBUF1	POINT U TO THE LOGICAL SECTOR NUMBERS
3024	D66E 4F		CLRA	#DD011	RESET THE TRACK COUNTER TO ZERO
3025	D66F 97 EC	LD66F	STA	DCTRK	SET THE DSKCON TRACK VARIABLE
3026	D671 5F		CLRB		RESET THE SECTOR COUNTER
3027	D672 A6 C5	LD672	LDA	B,U	GET THE PHYSICAL SECTOR NUMBER
		LD672	LDA STA	B,U DSEC	GET THE PHYSICAL SECTOR NUMBER SAVE DSKCON SECTOR VARIABLE
3027 3028 3029	D672 A6 C5 D674 97 ED D676 BD D6 F2	LD672	STA JSR		SAVE DSKCON SECTOR VARIABLE READ A SECTOR
3027 3028 3029 3030	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C	LD672	STA JSR INCB	DSEC LD6F2	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER
3027 3028 3029 3030 3031	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12	LD672	STA JSR INCB CMPB	DSEC LD6F2 #SECMAX	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER
3027 3028 3029 3030 3031 3032	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4	LD672	STA JSR INCB CMPB BLO	DSEC LD6F2 #SECMAX LD672	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT
3027 3028 3029 3030 3031 3032 3033	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC	LD672	STA JSR INCB CMPB BLO LDA	DSEC LD6F2 #SECMAX	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER
3027 3028 3029 3030 3031 3032 3033 3034	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D68Ø 4C	LD672	STA JSR INCB CMPB BLO LDA INCA	DSEC LD6F2 #SECMAX LD672 DCTRK	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK
3027 3028 3029 3030 3031 3032 3033 3034 3035	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC	LD672	STA JSR INCB CMPB BLO LDA INCA CMPA	DSEC LD6F2 #SECMAX LD672 DCTRK #TRKMAX	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF
3027 3028 3029 3030 3031 3032 3033 3034	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67C 96 EC D68Ø 4C D68I 81 23	LD672	STA JSR INCB CMPB BLO LDA INCA	DSEC LD6F2 #SECMAX LD672 DCTRK	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D68B 4C D681 B1 23 D683 25 EA D685 7E D2 CD D688 7F Ø9 86	LD672	STA JSR INCB CMPB BLO LDA INCA CMPA BLO	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F	SAVE DSKCON SECTOR VARIABLE READ A SECTOR  * INCREMENT THE SECTOR COUNTER  * AND COMPARE IT TO MAXIMUM SECTOR NUMBER  * AND KEEP LOOPING IF MORE SECTORS LEFT  GET THE CURRENT TRACK NUMBER  = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK  = NUMBER AND KEEP LOOPING IF  THERE ARE STILL TRACKS TO DO
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D68Ø 4C D681 81 23 D683 25 EA D685 7E D2 CD		STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP	DSEC LD6F2 #SECMAX LD672 DCTRK #TRKMAX LD66F LD2CD	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3038 3039 3040	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D68B 4C D681 B1 23 D683 25 EA D685 7E D2 CD D688 7F Ø9 86		STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR	DSEC LD6F2 #SECMAX LD672 DCTRK #TRKMAX LD66F LD2CD DRGRAM	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3038 3039 3040 3041	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F M9 86 D688 7F FF 40	LD688	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3038 3039 3040 3041 3042	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F M9 86 D688 7F FF 40	LD688	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR
3027 3028 3029 3030 3031 3032 3033 3034 3036 3037 3038 3039 3041 3042 3043	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F FF 40	LD688 * BUILD	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR ING AT DFLBUF.
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3040 2036 3041 3041 3042 3043 3044	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D688 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F Ø9 86 D688 7F D7 Ø1	LD688	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP A FORM LDX	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD7Ø1  MATTED TRACK OF DATA IN RAM STARTI	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3038 3039 3040 3041 3042 3043 3044 3045	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F M9 86 D688 7F FF 40 D688 7F D7 Ø1  D691 8E Ø9 89 D694 CC 20 4E	LD688 * BUILD	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP A FORM LDX LDD	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE AND SEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3038 3049 3041 3042 3043 3044 3044 3044 3044 3046	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D688 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F Ø9 86 D688 7F D7 Ø1	LD688 * BUILD	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP A FORM LDX LDD	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD7Ø1  MATTED TRACK OF DATA IN RAM STARTI	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3038 3041 3041 3044 3045 3044 3045	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D672 25 F4 D67E 96 EC D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F FF 40 D688 7F D7 Ø1  D691 8E Ø9 89 D694 CC 2Ø 4E D697 8D 29	LD688 * BUILD	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP A FORM LDX LDD BSR	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM START1 #DFLBUF #\$204E LD6C2	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV
3027 3028 3030 3031 3032 3033 3034 3035 3036 3037 3038 3040 3041 3044 3044 3044 3044 3044 3044	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 25 F4 D67C 25 F4 D67E 96 EC D688 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F FF 4Ø D694 CC 2Ø 4E D697 8D 29 D699 5F D694 34 Ø4 D69C CE Ø7 ØØ	LD688 * BUILD LD691	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR JMP A FORM LDX LDD BSR CLR CLR LDX LDD BSR LDX LDD BSR LDX LDD	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD7Ø1  MATTED TRACK OF DATA IN RAM STARTI #BFLBUF #\$204E LD6C2  B #DBUF1	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS
3027 3028 3029 3030 3031 3032 3033 3034 3036 3037 3044 3045 3044 3045 3044 3045 3044 3045 3044 3045 3046	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D688 4C D688 325 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F Ø9 86 D688 7F D7 Ø1  D691 8E Ø9 89 D694 CC 2Ø 4E D699 5F D690 34 Ø4 D69C CE Ø7 ØØ D69F E6 C5	LD688 * BUILD LD691	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP A FORN LDX LDD BSR CLRB PSHS LDU LDB	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM START1  #DFLBUF #\$204E LD6C2 B #BDBUF1 B,U	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARD STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND
3027 3028 3029 3030 3031 3032 3033 3034 3036 3037 3038 3040 3041 3042 3043 3044 3045 3046 3047 3048 3049 3051	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F M9 86 D688 7F B9 86 D688 7F B7 Ø7 D69C C2 Ø 4E D697 8D 29 D699 5F D699 5F D690 CE Ø7 ØØ D696 F6 C5 D671 BD D671 BD	LD688 * BUILD LD691	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP A FORN LDX LDD LDX LDD LDX LDU LDBSR CLRB PSHS LDU LDB STB	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3040 3041 3042 3043 3044 3044 3045 3044 3045 3044 3045 3046 3047 3048 3049 3050 3050 3050 3050 3050 3050 3050 305	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 5C D67A C1 12 D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F 09 86 D688 7F FF 40 D691 8E 09 89 D694 CC 20 4E D697 8D 29 D699 5F D69A 34 04 D69C CE 07 00 D69F E6 C5 D6A1 D7 ED D6A1 D7 ED	LD688 * BUILD LD691	STA JSR INCB CMPB BLO LDA INCA CMPA BLO JMP CLR CLR JMP A FORN LDX LDD BSR CLRB LDX LDD BSR LDU LDB LDU	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD7Ø1  MATTED TRACK OF DATA IN RAM STARTI ##DFLBUF #\$204E LD6C2  ##DBUF1 ##,U DSEC #LD6D4	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3045 3044 3045 3044 3045 3046 3047 3048 3049 3050 3051 3050 3051 3055 3055 3055 3055	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D688 4C D688 325 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F Ø9 86 D688 7F Ø9 87 Ø9 D694 CC 2Ø 4E D697 BD 29 D699 5F D69A 34 Ø4 D69C CE Ø7 ØØ D69F E6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3	LD688 * BUILD LD691	STA JSR JSR CMPB BLO LDA LDA CMPA BLO JMP CLR CLR JMP  A FORN LDX LDD BSR CLRB PSHS LDU LDB LDB LDU LDB	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  #ATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6D4 #\$83	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATING DATA * GET FIRST 3 DATA BLOCKS AND
3027 3028 3029 3030 3031 3032 3033 3034 3036 3037 3040 3041 3042 3043 3044 3045 3046 3047 3048 3049 3051 3052 3053 3051 3053	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D688 76 D9 86 D688 7F D9 86 D688 7F B0 86 D688 7F B7 80 D688 7F B7 87 D691 8E 89 89 D694 CC 28 4E D697 8D 29 D694 SC 28 4E D697 8D 29 D694 SC 28 4E D697 8D 29 D699 5F D698 34 84 84 D69C CE 87 88 D696	LD688 * BUILD LD691	STA JSR JSR LOR LDA	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  # #DBUF1 B,U DSEC #LD6C4 #\$03 LD6C4	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3040 3041 3042 3044 3044 3044 3045 3047 3048 3050 3050 3051 3052 3053 3055 3055	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 5C D67A C1 12 D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F 09 86 D688 7F D7 01  D691 8E 09 89 D694 CC 20 4E D697 8D 29 D699 5F D69A 34 04 D69C CE 07 00 D69F E6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 03 D6A8 BD 1E D6A4 OF ED	LD688 * BUILD LD691	STA JSR JSR JSR JSR CMPB BLO LDA INCA CMPA BBLO JMP CLR JMP A FORN LDX LDD BSR CLRB LDU LDB LDB LDD LDB LDD LDB LDB LDD LDB LDB	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD7Ø1  MATTED TRACK OF DATA IN RAM STARTI #BFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6D4 #\$43 LD6C8 LD6C8 LD6C8 LD6C8 LD6C8 LD6C8 LD6C8 LD6CR	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3045 3044 3045 3044 3045 3045 3055 305	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D688 4C D688 325 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F Ø9 86 D688 7F Ø9 87 D694 CC 2Ø 4E D697 8D 29 D699 5F D694 A1 Ø4 D696 CE Ø7 ØØ D697 B6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3 D6A8 BD 1E D6AA 96 EC D6AC A7 8Ø	LD688 * BUILD LD691	STA JSR JSR LDS LDA INCB LDA INCA LDA INCA CMPA BLO JMP CLR CLR LDX LDD LDX LDD LDB SSR CLRB STB LDU LDB STB LDU LDB LDB LDB LDB LDD LDB LDA	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI  #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6D4 ##\$03 LD6CB DCTRK ,X+	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER
3027 3028 3039 3031 3032 3033 3034 3035 3036 3037 3041 3041 3042 3043 3044 3044 3045 3047 3046 3047 3046 3047 3055 3056 3057	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 96 EC D680 4C D681 81 23 D683 25 EA D688 7F Ø9 86 D688 7F D2 CD D688 7F Ø7 Ø1  D691 8E Ø9 89 D694 CC 2Ø 4E D697 8D 29 D699 5F D69A 34 Ø4 D69C CE Ø7 ØØ D69F E6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3 D6A8 B0 1E D6AA 96 EC D6AC A7 8Ø D6AC D6F SØ	LD688 * BUILD LD691	STA JSR JSR JSR GMPB BLO LDA INCA CMPA BLO JMP CLR JMP A FORN LDD BSR LDD LDB LDU LDB LDD LDB LDU LDB LDD LDB LDB	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI  #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6C4 #\$03 LD6C8 DCTRK ,X+ ,X+ ,X+	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT
3027 3028 3029 3030 3031 3032 3033 3035 3036 3037 3040 3041 3042 3043 3044 3044 3045 3047 3048 3050 3051 3050 3055 3055 3056 3055 3056 3055 3056 3055 3056 3055 3056 3055 3056 3055 3056 3055 3056 3056	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D688 4C D688 325 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F Ø9 86 D688 7F Ø9 87 D694 CC 2Ø 4E D697 8D 29 D699 5F D694 A1 Ø4 D696 CE Ø7 ØØ D697 B6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3 D6A8 BD 1E D6AA 96 EC D6AC A7 8Ø	LD688 * BUILD LD691	STA JSR JSR LDS LDA INCB LDA INCA LDA INCA CMPA BLO JMP CLR CLR LDX LDD LDX LDD LDB SSR CLRB STB LDU LDB STB LDU LDB LDB LDB LDB LDD LDB LDA	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD7Ø1  MATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  #B #BBUF1 B,U DSEC #LD6CB DCTRK ,X+ ,X+ DSEC	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3045 3044 3045 3045 3051 3051 3053 3054 3055 3055 3056 3057 3057 3058	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 5C D67A C1 12 D680 4C D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F D7 Ø1  D691 8E Ø9 89 D694 CC 2Ø 4E D697 8D 29 D699 5F D69A 34 Ø4 D69C CE Ø7 ØØ D69F E6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3 D6A8 BD 1E D6AA G6 EC D6AC A7 8Ø D6BØ 96 ED	LD688 * BUILD LD691	STA JSRA JSRA JSRA JSRA JSRA JSRA LDA LDA INCA LDA INCA CMPA BLO JMP CCLR CLR JMP A FORN LDX LDD BSR CCLR LDB STB LDU LDB LDA STA CLR LDA LDA LDA LDA LDA LDA	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD7Ø1  MATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6CB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ ,X+ DSEC ,X+ ,X+ DSEC	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE AND SEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER SAVE SECTOR COUNTER SAVE SECTOR COUNTER TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT UT OT TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3044 3044 3045 3047 3048 3050 3051 3051 3055 3056 3057 3058 3059 3059 3059 3059 3059 3059 3059 3059	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 5C D67A C1 12 D67C 25 F4 D67E 96 EC D688 7E D2 CD D688 7F Ø9 86 D688 7F Ø9 86 D688 7F D7 Ø1  D691 8E Ø9 89 D694 CC 2Ø 4E D697 8D 29 D699 SF D694 A34 Ø4 D69C CE Ø7 ØØ D695 E6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3 D6A8 BD 1E D6A6 C7 BØ D6A9 CC D6AC A7 8Ø D6A6 C6 Ø3 D6A8 BD 1E D6B8 A8 Ø9 EC D6AC A7 8Ø D6B8 A8 Ø0 EC D6AC A7 8Ø D6B8 A8 Ø0 EC D6B8 A7 8Ø D6B8 A6 EC D6B8 A7 8Ø	LD688 * BUILD LD691	STA JSRA JSRA JSRA JSRA JSRA JSRA LDA LDA LDA LDA LDA LDA LDA LDA LDD BSR CCLR LDB STB LDB LDB LDB LDB LDB LDB LDB LDB LDB LD	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  #B #BBUF1 B,U DSEC #LD6C8 DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$409 LD6C8	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE AND SEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER SAVE SECTOR COUNTER SAVE SECTOR COUNTER TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT UT OT TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER  GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER  GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3045 3044 3045 3045 3051 3051 3053 3055 3055 3055 3055 305	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D672 96 EC D688 4C D688 7F Ø9 86 D688 7F Ø9 86 D688 7F Ø9 86 D688 7F Ø7 Ø1 D697 8D 29 D699 8D 29 D699 8D 29 D699 8D 29 D699 8D 29 D694 CC 2Ø 4E D697 8D 29 D699 8D 29 D690 8D 20 D691 8D 29 D692 8D 29 D693 8D 29 D694 8D 29 D695 8D 29 D696 8D 20 D688 8D 1E D688 8D 1E D688 8D 1E D688 96 ED D688 47 8Ø D688 6B D100 D688 8D 100	LD688 * BUILD LD691	STA JSRA JSRA JSRA JSRA JSRA JSRA LDA LDA LDA LDA LDA LDA LDX LDD BSR LDU LDB BSR LDU LDB BSR LDU LDB BSR LDU LDB BSR LDC LDB BSR LDC LDB BSR LDC LDB BSR LDC LDB BSR LDD LDB BSR LDD LDB BSR LDD BSR LDB BSR BSR BSR BSR BSR BSR BSR BSR BSR BS	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  #B #BBUF1 B,U DSEC #LD6C8 DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$409 LD6C8	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND  * WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND  * WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND  * WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND  * WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND  = WRITE THEM TO THE BUFFER GET SECTOR COUNTER
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3041 3042 3043 3044 3045 3047 3044 3045 3047 3055 3055 3055 3055 3055 3056 3056 3056	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 25 F4 D67E 96 EC D688 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F D7 Ø1  D691 8E Ø9 89 D694 CC 2Ø 4E D697 8D 29 D699 5F D694 34 Ø4 D697 8D 29 D699 F6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3 D6A8 B0 1E D6AA 96 EC D6AC A7 8Ø D68Ø 96 ED D6BØ 97 8Ø D6BØ 96 ED	LD688 * BUILD LD691	STA JSRA JSRA JSRA JSRA JSRA LDA INCA CMPA BLO INCA CMPA BLO JMP CLR JMP A FORN LDD BSR CLRB LDU LDB SSTB LDD LDB SSR LDA CLR LDA LDB SSR LDA LDB SSR LDA LDB SSR LDA LDB	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI  #DFLBUF #\$204E LD6C2  ##BBUF1 B,U DSEC #LD6C8 DCTRK ,X+ ,X+ DSEC ,X+ #\$09 LD6C8 B	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NIME DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER BUFFER GET SECTOR COUNTER BUFFER GET SECTOR TOWNER GET SECTOR COUNTER  * GET SECTOR TOWNER  * STORE IT IN THE BUFFER GET SECTOR COUNTER  * GET SECTOR COUNTER  * MRITE THEM TO THE BUFFER GET SECTOR COUNTER  * MRITE THEM TO THE BUFFER GET SECTOR COUNTER  * MRITE THEM TO THE BUFFER GET SECTOR COUNTER  * MRITE THEM TO THE BUFFER GET SECTOR COUNTER  * MRITE THEM TO THE BUFFER GET SECTOR COUNTER  ** MRITE THEM TO THE BUFFER GET SECTOR COUNTER  ** MRITE THEM TO THE BUFFER GET SECTOR COUNTER  ** MRITE THEM TO THE BUFFER GET SECTOR COUNTER  ** MRITE THEM TO THE BUFFER GET SECTOR COUNTER  ** MRITE THEM TO THE BUFFER GET SECTOR COUNTER  ** MRITE THEM TO THE BUFFER GET SECTOR COUNTER  ** MRITE THEM TO THE BUFFER GET SECTOR COUNTER  ** MRITE THEM TO THE BUFFER **
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3044 3045 3044 3045 3051 3051 3055 3055 3056 3055 3056 3057 3058 3059 3059 3059 3059 3059 3059 3059 3059	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 5C D67A C1 12 D67B 96 EC D688 4C D688 325 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F FF 4Ø D696 BC 2Ø D697 8D 29 D699 5F D694 A34 Ø4 D696 C6 Ø7 ØØ D695 E6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3 D6A8 BD 1E D6A6 C6 Ø3 D6A8 BD 1E D6A6 C7 BØ D6A9 CC D6AC A7 8Ø D6A6 C6 Ø3 D6B8 C7 BØ D6B8 C6 Ø9 D6B8 A7 BØ D6B8 B0 D6B8 B0 D6B8 B0 D6B8 B0 DØ D6B8 B0 D6B8 B0 DØ D6B8 B0 D6B8 B0 DØ	LD688 * BUILD LD691	STA JSRA JSRA JSRA JSRA JSRA JSRA LDA LDA LDA LDA LDA LDA LDA LDA LDB	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  B BBUBUF1 B,U DSEC #LD6C8 DCTRK ,X+ ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LD6C8 B #\$204 #\$30 #\$204 #\$30 #\$30 #\$30 #\$30 #\$30 #\$30 #\$30 #\$30	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER SAVE SECTOR COUNTER * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER SECTOR SUMBER AND * STORE IT IN THE BUFFER GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND * STORE IT IN THE BUFFER GET SECTOR COUNTER REST SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR COUNTER REST SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMTER REST SECTOR SOUMTER
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3041 3042 3043 3044 3045 3044 3045 3051 3051 3053 3054 3055 3055 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3057 3057 3057 3057 3057 3057 3057	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D670 25 F4 D67E 96 EC D688 4C D688 7F 09 86 D688 7F 09 86 D688 7F 07 01  D691 8E 09 89 D694 CC 20 4E D697 8D 29 D696 8F C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 03 D6A8 8D 1E D6AA 96 EC D6AA 96 EC D6BB A7 80 D6BB A7 80 D6BB A7 80 D6BB A7 80 D6AA 96 EC D6AA 96 EC D6AA 96 EC D6BB A7 80 D7	LD688 * BUILD LD691	STA JSRA JSRA JSRA JSRA JSRA JSRA LDA LDA LDA LDA LDA LDA LDC LCR LDX LDD BSR LDU LDB BSR LDU LDB BSR LDU LDB BSR LDU LDB BSR LDA LDB BSR BSR BSR BSR BSR BSR BSR BSR BSR BS	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD7Ø1  MATTED TRACK OF DATA IN RAM STARTI  #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6C4 #\$63 LD6C8 DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$89 LD6C8 B #SECMAX LD6C9 ##SECMAX LD69A	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND * STORE IT IN THE BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND  * WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER REXT SECTOR COUNTER NEXT SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3045 3047 3044 3045 3047 3045 3055 3056 3055 3056 3056 3056 3056 305	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 5C D67A C1 12 D67B 96 EC D688 4C D688 325 EA D685 7E D2 CD D688 7F Ø9 86 D688 7F FF 4Ø D696 BC 2Ø D697 8D 29 D699 5F D694 A34 Ø4 D696 C6 Ø7 ØØ D695 E6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3 D6A8 BD 1E D6A6 C6 Ø3 D6A8 BD 1E D6A6 C7 BØ D6A9 CC D6AC A7 8Ø D6A6 C6 Ø3 D6B8 C7 BØ D6B8 C6 Ø9 D6B8 A7 BØ D6B8 B0 D6B8 B0 D6B8 B0 D6B8 B0 DØ D6B8 B0 D6B8 B0 DØ D6B8 B0 D6B8 B0 DØ	LD688 * BUILD LD691	STA JSRA JSRA JSRA JSRA JSRA JSRA LDA LDA LDA LDA LDA LDA LDC LCR LDX LDD BSR LDU LDB BSR LDU LDB BSR LDU LDB BSR LDU LDB BSR LDA LDB BSR BSR BSR BSR BSR BSR BSR BSR BSR BS	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  B BBUBUF1 B,U DSEC #LD6C8 DCTRK ,X+ ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LD6C8 B #\$204 #\$30 #\$204 #\$30 #\$30 #\$30 #\$30 #\$30 #\$30 #\$30 #\$30	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER SAVE SECTOR COUNTER * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER SECTOR SUMBER AND * STORE IT IN THE BUFFER GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND * STORE IT IN THE BUFFER GET SECTOR COUNTER REST SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR COUNTER REST SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMBER AND  * STORE IT IN THE BUFFER GET SECTOR SOUMTER REST SECTOR SOUMTER
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3044 3045 3044 3045 3050 3051 3055 3056 3055 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3057 3057 3057 3057 3057 3057 3057	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D670 25 F4 D67E 96 EC D688 4C D688 7F 09 86 D688 7F 09 86 D688 7F 07 01  D691 8E 09 89 D694 CC 20 4E D697 8D 29 D696 8F C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 03 D6A8 8D 1E D6AA 96 EC D6AA 96 EC D6BB A7 80 D6BB A7 80 D6BB A7 80 D6BB A7 80 D6AA 96 EC D6AA 96 EC D6AA 96 EC D6BB A7 80 D7	LD688 * BUILD LD691 LD69A	STA JSRA JSRA JSRA JSRA JSRA JSRA BLO LDA LDA LDA LDA CLR JMP A FORN LDX LDD BSR CLR LDB STB LDU LDB STB LDU LDB STA CLR LDB STA LDA STA LDB STA LDB STA LDB STA LDD LDD STA LDD LDD LDD LDD LDD LDD LDD LDD LDD LD	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI  #DFLBUF #\$204E LD6C2  B #B BDBUF1 B,U DSEC #LD6D4 #\$403 LD6C8 DCTRK ,X+	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND * STORE IT IN THE BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND  * WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER REXT SECTOR COUNTER NEXT SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE
3027 3028 3029 3030 3031 3032 3033 3034 3035 3034 3041 3042 3043 3044 3045 3044 3045 3055 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3066 3066 3066 3066 3066 3066 3066 306	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D672 96 EC D688 4C D688 7F Ø9 86 D688 7F Ø9 86 D688 7F Ø9 86 D688 7F D7 Ø1  D691 8E Ø9 89 D694 CC 2Ø 4E D699 5F D69A 34 Ø4 D69C CE Ø7 ØØ D69F E6 C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 Ø3 D6A8 8D 1E D6AA 96 EC D6AA 96 EC D6BB 7F 8Ø D6BB 7F BD 9 D6AA 96 EC D6AA 96 EC D6AB 7F 8Ø D6BB 7F BB D	LD688 * BUILD LD691 LD69A	STA JSR JINCB GMPB BLO LDA INCA CMPA BLO INCA CMPA BLO INCA CMPA A FORM LDX LDD BSR LDU LDB BSR LDU LDB BSR LDU LDB BSR LDA LDB BSR LDD LDB BSR LDA LDB BSR LDD LDD BSR LDD BSR LDD BSR LDD LDD BSR LDD BSR LDD BSR LDD BSR LDD LDD BSR LD	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI  #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6D4 #\$63 LD6C8 DCTRK ,X+ ,X+ ,X+ DSEC ,X+ ,X+ #\$99 LD6CB B ##SECMAX LD69A #\$C84E  BYTES OF ACCB INTO BUFFER	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND * STORE IT IN THE BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER = GET THE LAST NINE DATA BLOCKS AND  * WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER  BY STORE IT IN THE BUFFER HERT SECTOR SECTOR OF TRACK  BY SECTOR SECTOR SOOT DONE WRITE THEM TO THE BUFFER HERT SECTOR COUNTER  BY SECTOR SECTOR SOOT DONE WRITE THEM TO THE BUFFER HERT SECTOR SOOT DONE WRITE THEM TO THE BUFFER HERT SECTOR SOOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3044 3045 3044 3045 3050 3051 3055 3056 3055 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3057 3057 3057 3057 3057 3057 3057	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D670 25 F4 D67E 96 EC D688 4C D688 7F 09 86 D688 7F 09 86 D688 7F 07 01  D691 8E 09 89 D694 CC 20 4E D697 8D 29 D696 8F C5 D6A1 D7 ED D6A3 CE D6 D4 D6A6 C6 03 D6A8 8D 1E D6AA 96 EC D6AA 96 EC D6BB A7 80 D6BB A7 80 D6BB A7 80 D6BB A7 80 D6AA 96 EC D6AA 96 EC D6AA 96 EC D6BB A7 80 D7	LD688  * BUILD LD691  LD69A	STA JSRA JSRA JSRA JSRA JSRA JSRA BLO LDA LDA LDA LDA CLR JMP A FORN LDX LDD BSR CLR LDB STB LDU LDB STB LDU LDB STA CLR LDB STA LDA STA LDB STA LDB STA LDB STA LDD LDD STA LDD LDD LDD LDD LDD LDD LDD LDD LDD LD	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI  #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6D4 #\$63 LD6C8 DCTRK ,X+ ,X+ ,X+ DSEC ,X+ ,X+ #\$99 LD6CB B ##SECMAX LD69A #\$C84E  BYTES OF ACCB INTO BUFFER	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND * STORE IT IN THE BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND  * WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER REXT SECTOR COUNTER NEXT SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3045 3044 3045 3045 3050 3051 3055 3056 3055 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3057 3057 3057 3057 3057 3057 3057	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D679 95 EC D680 4C D681 81 23 D683 25 EA D685 7E D2 CD D688 7F 09 86 D688 7F D7 01  D691 8E 09 89 D694 CC 20 4E D697 8D 29 D699 5F D69A 34 04 D69C CE 07 00 D69F E6 C5 D6A1 D7 ED D6A1 D7 ED D6A2 G 6 B3 D6A8 B D 1E D6A4 96 EC D6A6 A7 80 D6A6 B  10 D6B6 B  10 D6B6 B  10 D6B6 B  10 D6B8 CC C  84 D6BB CC C  84 D6BB CC C  84 D6BB CC C  84 D6BB CC C C 8 4E	LD688  * BUILD LD691  LD69A	STA JSR JINCB JNCB BLO LDA LINCA CMPA BLO LINCA CMPA BLO LDC LDC BLO STB LDU LDB STB LDU LDB BSR CLR LDB BSR LDU LDB BSR LDD ACCA LDB BSR LDD ACCA LDB BSR PULS INCB BSR LDD ACCA LDB BSR BLO LDD ACCA LDB BCB BCB BCB BCB BCB BCB BCB BCB BCB B	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DRGRAM DSKREG LD701  MATTED TRACK OF DATA IN RAM STARTI  #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6D4 #\$63 LD6C8 DCTRK ,X+ ,X+ ,X+ DSEC ,X+ ,X+ #\$99 LD6CB B ##SECMAX LD69A #\$C84E  BYTES OF ACCB INTO BUFFER	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER * AND KEEP LOOPING IF MORE SECTORS LEFT = GET THE CURRENT TRACK NUMBER = ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF = THERE AND SETE LOOPING IF = THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER SAVE IT IN THE DSKCON VARIABLE POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET SECTOR COUNTER BUFFER GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET SECTOR COUNTER BUFFER GET SECTOR COUNTER SECTORS PER TRACK BRANCH IF ALLS TINE DATA BLOCKS AND WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER
3027 3028 3029 3030 3031 3032 3033 3034 3035 3036 3037 3044 3045 3044 3045 3045 3050 3051 3055 3056 3055 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3057 3057 3057 3057 3057 3057 3057	D672 A6 C5 D674 97 ED D676 BD D6 F2 D679 5C D67A C1 12 D670 96 EC D688 76 96 EC D688 77 99 86 D688 77 99 86 D688 77 99 86 D688 77 99 87 D697 8D 29 D699 5F D694 CC 20 4E D697 8D 29 D699 5F D694 34 04 D696 C6 07 00 D696 C6 07 00 D696 C6 07 00 D697 8D 1E D640 A7 80 D640 C6 09 D640 C6 09 D640 C7 09 D650 C6 09 D650 C7 09 D	LD688  * BUILD LD691  LD69A	STA JSR JINCB JNCB BLO LDA LINCA CMPA BLO LINCA CMPA BLO LDC LDC BLO STB LDU LDB STB LDU LDB BSR CLR LDB BSR LDU LDB BSR LDD ACCA LDB BSR LDD ACCA LDB BSR PULS INCB BSR LDD ACCA LDB BSR BLO LDD ACCA LDB BCB BCB BCB BCB BCB BCB BCB BCB BCB B	DSEC LD6F2  #SECMAX LD672 DCTRK  #TRKMAX LD66F LD2CD DAGRAM DSKREG LD701  #ATTED TRACK OF DATA IN RAM STARTI #DFLBUF #\$204E LD6C2  B #DBUF1 B,U DSEC #LD6C4 #\$403 LD6C8 DCTRK ,X+ ,X+ ,X+ ,X+ DSEC ,X+ #\$99 LD6C8 B #SECMAX LD69A #\$C84E  #\$71ES OF ACCB INTO BUFFER ,X+	SAVE DSKCON SECTOR VARIABLE READ A SECTOR * INCREMENT THE SECTOR COUNTER * AND COMPARE IT TO MAXIMUM SECTOR NUMBER AND KEEP LOOPING IF MORE SECTORS LEFT GET THE CURRENT TRACK NUMBER ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK NUMBER AND KEEP LOOPING IF THERE ARE STILL TRACKS TO DO GO CHECK FOR A DOS RESET CLEAR RAM IMAGE OF DSKREG CLEAR DSKREG - TURN DISK MOTORS OFF PROCESS DRIVES NOT READY ERROR  ING AT DFLBUF.  START TRACK BUFFER AT DFLBUF GET SET TO WRITE 32 BYTES OF \$4E GO WRITE GAP IV RESET SECTOR COUNTER SAVE SECTOR COUNTER POINT U TO THE TABLE OF LOGICAL SECTORS * GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER GET TRACK INDRER AND * STORE IT IN THE BUFFER GET THE LAST NINE DATA BLOCKS AND  * WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR OUNTER NEXT SECTOR SECTOR OUNTER NEXT SECTOR OUNTER NEXT SECTOR OUNTER NEXT SECTOR SECTOR OUNTER NEXT SECTOR OUNTER NEXT SECTOR SECTOR OUNTER SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK

3Ø73					
	D6C8 34 Ø4	LD6C8	PSHS	В	SAVE THE COUNTER ON THE STACK
3074	D6CA EC C1		LDD	,U++	GET TWO BYTES OF DATA FROM THE TABLE
3Ø75	D6CC 8D F4		BSR	LD6C2	WRITE ACCA BYTES OF ACCB INTO THE BUFFER
3Ø76	D6CE 35 Ø4		PULS	В	* GET THE COUNTER BACK, DECREMENT
3Ø77	D6DØ 5A		DECB		* IT AND BRANCH IF ALL DATA BLOCKS
3Ø78	D6D1 26 F5		BNE	LD6C8	* NOT DONE
3Ø79	D6D3 39		RTS		
3080					
3Ø81		* DATA U	JSED TO	O FORMAT A SECTOR ON THE DISK	
3Ø82					NAT FOR OUR RYTE OFFICE
3Ø83				ARE CLOSE TO THE IBM SYSTEM 34 FOR	
3084				ITY. THE FORMAT GENERALLY CONFORMS	
3085				HEET. THE GAP SIZES HAVE BEEN REDU THE IBM FORMAT USES \$40 AS THE FIU	
3Ø86 3Ø87				E COLOR DOS USES AN \$FF AS THE FIL	
3088	D6D4 Ø8 ØØ	LD6D4	FCB		SYNC FIELD
3Ø89	D6D6 Ø3 F5	LDOD4		3,\$F5	SINC FIELD
3090	D6D8 Ø1 FE			1,\$FE	ID ADDRESS MARK (AM1)
3Ø91		* TRACK.		, AND SECTOR NUMBERS ARE INSERTED	
3Ø92	D6DA Ø1 Ø1		FCB		SECTOR SIZE (256 BYTE SECTORS)
3Ø93	D6DC Ø1 F7			1,\$F7	CRC REQUEST
3Ø94	D6DE 16 4E			22,\$4E	GAP II (POST-ID GAP)
3Ø95	D6EØ ØC ØØ		FCB	12,0	SYNC FIELD
3Ø96	D6E2 Ø3 F5		FCB	3,\$F5	
3Ø97	D6E4 Ø1 FB		FCB	1,\$FB	DATA ADDRESS MARK (AM2)
3Ø98	D6E6 ØØ FF		FCB	Ø,\$FF	DATA FIELD (256 BYTES)
3Ø99	D6E8 Ø1 F7		FCB	1,\$F7	CRC REQUEST
3100	D6EA 18 4E		FCB	24,\$4E	GAP III (POST DATA GAP)
3101					
3102					
3103	DC50 0C 5:	* DOS CO		1.5740	DETUDU TE ADQUIVENT ATTE
3104	D6EC 26 54	DOS			RETURN IF ARGUMENT GIVEN
3105	D6EE 6E 9F CØ ØA		JMP	[DOSVEC]	JUMP TO THE DOS COMMAND
3106	D6E2 24 44	IDEEO	Deric	P	SAVE ACCD
31Ø7 31Ø8	D6F2 34 Ø4 D6F4 C6 Ø5	LD6F2	PSHS	B #\$Ø5	SAVE ACCB 5 RETRIES
31Ø9 311Ø	D6F6 F7 Ø9 88 D6F9 35 Ø4		PULS	ATTCTR	SAVE RETRY COUNT RESTORE ACCB
3111	D6FB 8D 62	LD6FB		DSKCON	GO EXECUTE COMMAND
3112	D6FD ØD FØ	LDOID	TST	DCSTA	CHECK STATUS
3113	D6FF 27 ØD			LD7ØE	BRANCH IF NO ERRORS
3114	D7Ø1 96 FØ	LD7Ø1		DCSTA	GET DSKCON ERROR STATUS
3115	D7Ø3 C6 3C			#2*30	'WRITE PROTECTED' ERROR
3116	D7Ø5 85 4Ø			#\$40	CHECK BIT 6 OF STATUS
3117	D7Ø7 26 Ø2			LD7ØB	BRANCH IF WRITE PROTECT ERROR
3118	D7Ø9 C6 28	LD7Ø9		#2*20	'I/O ERROR'
3119	D7ØB 7E AC 46	LD7ØB	JMP	LAC46	JUMP TO ERROR DRIVER
3120	D7ØE 34 Ø2	LD7ØE	PSHS	A	SAVE ACCA
3121	D71Ø 96 EA		LDA	DCOPC	GET OPERATION CODE
3122	D712 81 Ø3		CMPA		CHECK FOR WRITE SECTOR COMMAND
3123	D714 35 Ø2		PULS		RESTORE ACCA
3124	D716 26 2A			LD742	RETURN IF NOT WRITE SECTOR
3125	D718 7D Ø9 87		TST		CHECK VERIFY FLAG
	D71D 07 0F				RETURN IF NO VERIFY
3126	D71B 27 25		BEQ		CAVE DECICEDS
3126 3127	D71D 34 56		PSHS	U,X,B,A	SAVE REGISTERS
3126 3127 3128	D71D 34 56 D71F 86 Ø2		PSHS LDA	U, X, B, A #\$02	READ OPERATION CODE
3126 3127 3128 3129	D71D 34 56 D71F 86 Ø2 D721 97 EA		PSHS LDA STA	U,X,B,A #\$Ø2 DCOPC	READ OPERATION CODE STORE TO DSKCON PARAMETER
3126 3127 3128 3129 3130	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE		PSHS LDA STA LDU	U,X,B,A #\$02 DCOPC DCBPT	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS
3126 3127 3128 3129 3130 3131	D71D 34 56 D71F 86 Ø2 D721 97 EA		PSHS LDA STA LDU LDX	U,X,B,A #\$02 DCOPC DCBPT #DBUF1	READ OPERATION CODE STORE TO DSKCON PARAMETER
3126 3127 3128 3129 3130	D71D 34 56 D71F 86 02 D721 97 EA D723 DE EE D725 8E 07 00		PSHS LDA STA LDU LDX STX	U,X,B,A #\$02 DCOPC DCBPT	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT UTO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER
3126 3127 3128 3129 3130 3131 3132	D71D 34 56 D71F 86 02 D721 97 EE D723 DE EE D725 8E 07 00 D728 9F EE		PSHS LDA STA LDU LDX STX BSR	U,X,B,A #\$02 DCOPC DCBPT #DBUF1 DCBPT	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE
3126 3127 3128 3129 3130 3131 3132 3133	D71D 34 56 D71F 86 02 D721 97 EA D723 DE EE D725 8E 07 00 D728 9F EE D72A 8D 33		PSHS LDA STA LDU LDX STX BSR STU	U,X,B,A #\$02 DCOPC DCBPT #DBUF1 DCBPT DSKCON	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR
3126 3127 3128 3129 3130 3131 3132 3133 3134	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE		PSHS LDA STA LDU LDX STX BSR STU	U,X,B,A #\$92 DCOPC DCBPT #DBUF1 DCBPT DSKCON DCBPT	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3		PSHS LDA STA LDU LDX STX BSR STU LDA	U,X,B,A #\$82 DCOPC DCBPT #DBUF1 DCBPT DSKCON DCBPT #\$83	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3138	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD		PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA BNE	U,X,B,A #\$82 DCOPC DCOPC DCBPT #DBUF1 DCSPT DSKCON DCBPT #\$63 DCOPC DCSTA LD743	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT UTO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3138 3139	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D73Ø 96 FØ D734 26 ØD D736 5F		PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA BNE CLRB	U,X,B,A #\$92 DCOPC DCBPT #DBUF1 DCBPT DSKCON DCBPT #\$93 DCOPC DCSTA LD743	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3138 3139 3140	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D728 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D734 56 ØD D735 A6 8Ø		PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA BNE CLRB LDA	U,X,B,A #\$02 DCOPC DCBPT #DBUF1 DCBPT DSKCON DCBPT #\$03 DCOPC DCSTA LD743 ,X+	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT UT OWRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3138 3139 3140 3141	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ		PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA BNE CLRB LDA CMPA	U, X, B, A #\$82 DCOPC DCOPC DCBPT #DBUF1 DCBPT DSKCON DCBPT #\$83 DCOPC DCSTA LD743 , X+ ,U+	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT UTO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3138 3139 3141 3141	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D738 P1 CØ D738 26 Ø6		PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA BNE CLRB LDA CMPA BNE	U,X,B,A #\$92 DCOPC DCOPC DCBPT #DBUF1 DCBPT DSKCON DCBPT #\$93 DCOPC DCSTA LD743	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO EADS SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3138 3139 3140 3141 3142 3143	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D738 56 D738 26 Ø6 D738 56		PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA BNE CLRB LDA CMPA BNE DECB	U, X, B, A #\$82 #\$82 #\$82 #\$82 #\$85 #\$85 #\$85 #\$85 #\$85 #\$85 #\$85 #\$85	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT UT OWRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND
3126 3127 3128 3129 3130 3131 3132 3133 3134 3136 3137 3138 3140 3141 3142 3143 3144	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D738 26 Ø6 D738 26 Ø6 D738 26 Ø7	LD737	PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA BNE CLRB LDA CMPA BNE DECB	U, X, B, A #\$82 DCOPC DCOPC DCBPT #DBUF1 DCBPT DSKCON DCBPT #\$83 DCOPC DCSTA LD743 , X+ , U+ LD743 LD743	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT UT OWRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * BERANCH IF NOT DONE
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3138 3139 3140 3141 3142 3143 3144 3145	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D738 26 Ø6 D738 26 Ø6 D738 26 Ø7	LD737	PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA BNE CLRB LDA CMPA BNE DECB	U, X, B, A #\$82 #\$82 #\$82 #\$82 #\$85 #\$85 #\$85 #\$85 #\$85 #\$85 #\$85 #\$85	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT UT OWRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3140 3141 3142 3143 3144 3144 3145	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D734 26 ØD D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D730 5A D738 26 F7 D74Ø 35 56 D742 39	LD737 LD742	PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA CMPA BNE CLRB LDA CMPA BNE DECB BNE PULS RTS	U,X,B,A #\$82 DCOPC DCOPC DCOPT #BDBUF1 DCSPT DSKCON DCEPT #\$83 DCOPC DCSTA LD743 ,X+ ,U+ LD743 LD737 A,B,X,U	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT UT OWRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3140 3141 3142 3143 3144 3145 3144 3145	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D738 26 Ø6 D738 26 Ø6 D738 26 Ø6 D738 57 D739 5A D732 56 D737 5A D738 26 Ø7 D738 26 Ø7 D748 35 56 D742 39 D743 35 56	LD737	PSHS LDA STA LDU LDX STX BSR STU LDA BNE CLRB LDA BNE CMPA BNE DECB BNE PULS	U,X,B,A #\$92 DCOPC DCOPC DCEPT #DBUF1 DCEPT DSKCON DCBPT #\$93 DCOPC DCSTA LD743 ,X+ ,U+ LD743 LD743 LD743 LD737 A,B,X,U A,B,X,U	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3140 3141 3142 3143 3144 3145 3146 3147	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D739 5A D739 5B D749 35 56 D742 7A Ø9 88	LD737 LD742	PSHS LDA STA LDU LDX STX BSR STU LDA STA LDA CMPA CMPA BNE DECB BNE PULS RTS PULS DEC	U,X,B,A #\$92 DCOPC DCOPT DCOPT #DBUF1 DCEPT DSKCON DCEPT #\$93 DCOPC DCSTA LD743  LD743  LD743  LD743  LD757 A,B,X,U A,B,X,U A,B,X,U A,B,X,U ATTCTR	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT UT OWRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS
3126 3127 3128 3129 3131 3131 3132 3133 3134 3135 3136 3140 3141 3142 3143 3144 3145 3146 3147 3148 3149	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D738 26 Ø6 D738 26 Ø7 D739 A1 CØ D738 26 Ø7 D739 A1 CØ D738 26 Ø7 D739 A5 Ø7 D739 A5 Ø7 D739 A5 Ø7 D739 A6 Ø7 D739 A6 Ø7 D739 A7 D740 35 56 D742 39 D743 35 56 D745 7A Ø9 88 D748 26 B1	LD737 LD742	PSHS LDA STA LDU LDX STX BST LDA STA LDA CMPA BNE DECB BNE PULS RTS PULS BNE BNE	U,X,B,A #\$82 DCOPC DCOPC DCOPT #DBUF1 DCOPT DSKCON DCBPT #\$83 DCOPC DCSTA LD743  ,X+ ,U+ LD743  LD743  LD737 A,B,X,U A,B,X,U A,B,X,U ATTCTR LD6FB	READ OPERATION CODE  STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS  * ADDRESS OF VERIFY BUFFER  * TO DSKCON VARIABLE GO EAD SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL  * DECREMENT BYTE COUNTER AND RESTORE REGISTERS  RESTORE REGISTERS DECREMENT THE VERIFY COUNTER
3126 3127 3128 3129 3131 3131 3132 3133 3134 3135 3136 3140 3141 3142 3143 3144 3145 3146 3147 3148 3149	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D739 5A D739 5B D749 35 56 D742 7A Ø9 88	LD737 LD742	PSHS LDA STA LDU LDX STX BSR LDA STA LDA BNE LDB BNE PULS RTS PULS BNE LDB	U,X,B,A #\$92 DCOPC DCOPT DCOPT #DBUF1 DCEPT DSKCON DCEPT #\$93 DCOPC DCSTA LD743  LD743  LD743  LD743  LD757 A,B,X,U A,B,X,U A,B,X,U A,B,X,U ATTCTR	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TILES LEFT
3126 3127 3128 3129 3130 3131 3132 3133 3135 3136 3137 3138 3140 3141 3142 3143 3144 3144 3145 3147 3148 3147 3148 3149 3150	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D738 26 Ø6 D738 26 Ø6 D738 56 D739 57 D740 35 56 D742 39 D743 35 56 D745 7A Ø9 88 D748 26 B1	LD737 LD742	PSHS LDA STA LDU LDX STX BSR LDA STA LDA BNE LDB BNE PULS RTS PULS BNE LDB	U,X,B,A #\$82 DCOPC DCOPT DCOPT DCOPT DSNCON DCBPT DSNCON DCBPT #\$83 DCOPC DCSTA LD743 ,X+ ,U+ LD743 LD743 LD737 A,B,X,U ATTCTR LD6FB #2*36	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR'
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3140 3141 3142 3143 3144 3145 3146 3147 3148 3159 3151 3151 3151	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D738 26 Ø6 D73D 5A D73B 26 Ø6 D73D 5A D73E 26 F7 D74Ø 35 56 D742 39 D743 35 56 D742 39 D743 35 56 D745 7A Ø9 88 D748 26 BD	LD737 LD742 LD743 * VERIFY	PSHS LDA STA LDU LDX STX LDU LDX STX LDA STA LDA LDA BNE CLRB LDA BNE DECB BNE DECB BNE LDA LDA LDA LDA RTS PULS RTS PULS LDB BNE LDB BNE LDB BNE LDB	U, X, B, A #\$82 DCOPC DCOPC DCOPT #DBUF1 DCOPT DSKCON DCBPT #\$83 DCOPC DCSTA LD743  ,X+ ,U+ LD743  LD747 A, B, X, U ATICTR LD6FB #2*36 LD76B	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR'
3126 3127 3128 3129 3131 3132 3133 3134 3135 3136 3137 3148 3149 3144 3144 3144 3145 3146 3147 3148 3149 3151 3151 3151 3151 3151 3151	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D728 9F EE D728 9F EE D728 80 Ø3 D730 97 EA D734 26 ØD D734 26 ØD D736 5F D737 A6 80 D738 26 Ø6 D738 26 Ø6 D738 26 Ø6 D738 26 Ø6 D739 26 Ø6 D734 26 ØD D736 5F D747 37 A6 80 D738 26 Ø6 D739 A1 CØ D738 26 Ø6 D730 5A D73E 26 F7 D74Ø 35 56 D742 39 D743 35 56 D742 39 D743 35 56 D745 7A Ø9 88 D748 26 B1 D74A C6 48 D74C 2Ø BD	LD737 LD742 LD743	PSHS LDA LDA LDV LDX STX BSR STU LDA BNE CLRB BNE PULS RPULS RPULS LDA LDA LDA LDA LDA LDA CMPA BNE CLRB CCRB LDA	U,X,B,A #\$92 DCOPC DCOPC DCOPT #DBUF1 DCEPT DSKCON DCEPT #\$93 DCOPC DCSTA LD743 ,X+ ,U+ LD743 LD737 A,B,X,U A,B,X,U ATTCTR LD6FB #2*36 LD70B	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO DSKCON VARIABLE GO DSKCON VARIABLE GO TO THE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3140 3141 3142 3143 3144 3145 3146 3147 3146 3147 3150 3150 3151 3152 3153 3153	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D738 26 Ø6 D735 26 F7 D740 35 56 D742 39 D743 35 56 D745 7A Ø9 88 D748 C6 48 D74C 2Ø BD D74E 5F D74F 81 AA	LD737 LD742 LD743 * VERIFY	PSHS LDA STA LDU LDX STX BSR STU LDA	U, X, B, A #\$82 DCOPC DCOPC DCOPT #DBUF1 DCOPT DSKCON DCBPT #\$63 DCOPC DCSTA LD743  ,X+ ,U+ LD743  LD737 A, B, X, U ATTCTR LD6FB #2*36 LD70B AND #\$AA	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MOT DONE RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ?
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3140 3141 3142 3143 3144 3145 3146 3147 3148 3151 3151 3151 3151 3151 3151 3151 315	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72A 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D738 26 Ø6 D73D 5A D73B 26 Ø6 D73D 5A D73E 26 F7 D74Ø 35 56 D742 39 D748 26 B1 D748 26 B1 D748 26 B1 D749 27 88 D749 28 BD	LD737 LD742 LD743 * VERIFY	PSHS LDA LDU LDX STX BSR STU LDA BNE LDA LDA BNE DECB BNE PULS PULS DEC LDB BNE LDB BNE LDB BNE LDB BNE LDB BNE BNE LD	U,X,B,A #\$92 DCOPC DCOPC DCOPT #DBUF1 DCEPT DSKCON DCEPT #\$93 DCOPC DCSTA LD743 ,X+ ,U+ LD743 LD737 A,B,X,U A,B,X,U ATTCTR LD6FB #2*36 LD70B	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS  RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MOTRE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES
3126 3127 3128 3129 3131 3132 3133 3134 3135 3136 3137 3138 3140 3141 3142 3143 3144 3145 3146 3147 3148 3149 3151 3151 3151 3151 3151 3151 3151 315	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D728 9F EE D728 9F EE D728 86 Ø7 ØØ D738 97 EA D738 97 EA D738 97 EA D734 26 ØD D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 F7 D740 35 56 D742 39 D743 35 56 D742 39 D743 35 56 D745 7A Ø9 88 D748 26 B1 D74A C6 48 D74C 2Ø BD D74E 5F D74F 81 AA D751 27 Ø7 D753 53	LD737 LD742 LD743 * VERIFY	PSHS LDA LDU LDX STA LDA STA LDA STA LDA STA LDA BNE CLRB BNE CMPA BNE DECB BNE DECB BNE CMPA CCMPA CMPA CMPA CMPA CMPA COMMB COMBA	U, X, B, A #\$82 DCOPC DCOPC DCOPT #DBUF1 DCEPT DSKCON DCEPT #\$83 DCOPC DCSTA LD743 ,X+ ,U+ LD743  LD737 A, B, X, U ATICTR LD6FB #2*36 LD70B AND #\$AA LD75A	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO DSKCON VARIABLE GO DSKCON VARIABLE GO TO THE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF
3126 3127 3128 3129 3131 3132 3133 3134 3135 3136 3137 3140 3141 3142 3143 3144 3145 3147 3150 3151 3150 3151 3152 3153 3155 3155 3155 3155 3155	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D739 5A D732 56 F7 D740 35 56 D745 7A Ø9 88 D748 26 81 D74A C6 48 D74C 2Ø BD	LD742 LD743 * VERIFY VERIFY	PSHS LDA LDU LDX STX BSR STDA LDA STA LDA STA LDA BNE CLRB BNE DECB BNE PULS PULS BNE LDB BNE CLBB BNE CMPA BNE CCRPA BNE CCRPA CCRPA BNE CCRPA BNE CCRPA CCRPA CCRPA CCRPA COMMB	U,X,B,A #\$82 DCOPC DCOPT DCOPT DCOPT DSNCON DCBPT DSNCON DCBPT BSNCON DCBPT JSNCON DCSTA LD743 ,X+ ,U+ LD743 LD743 LD737 A,B,X,U ATTCTR LD6FB #2*36 LD70B AND #\$AA LD75A #\$88	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS  RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3140 3141 3142 3143 3144 3145 3146 3147 3148 3151 3151 3151 3151 3151 3151 3151 315	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72A 8D 63 D730 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D738 26 Ø6 D739 A1 CØ D738 26 Ø6 D730 5A D730 5A D740 35 56 D742 39 D744 7A Ø9 D748 26 B1 D744 C6 48 D744 2Ø BD D745 FF D747 A0 98 D748 5F D748 26 B1 D749 26 BD	LD742 LD743 * VERIFY VERIFY	PSHS LDA LDU LDX STX BSR STDA LDA STA LDA STA LDA BNE CLRB BNE DECB BNE PULS PULS BNE LDB BNE CLBB BNE CMPA BNE CCRPA BNE CCRPA CCRPA BNE CCRPA BNE CCRPA CCRPA CCRPA CCRPA COMMB	U,X,B,A #\$82 DCOPC DCOPT DCOPT DCOPT DSNCON DCBPT DSNCON DCBPT BSNCON DCBPT JSNCON DCSTA LD743 ,X+ ,U+ LD743 LD743 LD737 A,B,X,U ATTCTR LD6FB #2*36 LD70B AND #\$AA LD75A #\$88	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS  DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF
3126 3127 3128 3129 3131 3132 3133 3134 3135 3136 3137 3148 3149 3141 3144 3145 3146 3147 3148 3149 3151 3151 3152 3153 3154 3155 3156 3157 3158 3157 3158 3157 3158	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D728 9F EE D728 9F EE D728 86 Ø7 ØØ D738 97 EA D739 97 EA D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D730 5A D738 26 Ø6 D734 26 ØD D736 5F D742 39 D743 35 56 D744 26 B1 D746 C6 48 D747 C7 Ø7 D748 18 AA D751 27 Ø7 D753 53 D754 81 88 D756 1Ø 26 DB 1D D756 F7 Ø9 87	LD737 LD742 LD743 * VERIFY	PSHS LDA LDU LDX STX BSR STU LDA STA LDA BNE CLRB BNE PULS RDE LDA BNE CLRB BNE CLRB BNE CMPA LBNE CMPA LBNE CMPA LBNE CMPA LBNE LDB	U, X, B, A #\$82 DCOPC DCOPC DCOPT #DBUF1 DCSPT DSKCON DCSPT #\$83 DCOPC DCSTA LD743  ,X+ ,U+ LD743  LD737 A, B, X, U A, B, X, U A, B, X, U ATICTR LD6FB #2*36 LD70B #\$AA LD75A #\$A8 LD75A	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS RESTORE REGISTERS  RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG
3126 3127 3128 3129 3131 3131 3132 3133 3134 3135 3136 3140 3141 3144 3144 3145 3147 3146 3150 3151 3150 3151 3152 3153 3153 3155 3156 3157 3158 3159 3169	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72A 8D 63 D730 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D738 26 Ø6 D739 A1 CØ D738 26 Ø6 D730 5A D730 5A D740 35 56 D742 39 D744 7A Ø9 D748 26 B1 D744 C6 48 D744 2Ø BD D745 FF D747 A0 98 D748 5F D748 26 B1 D749 26 BD	LD742 LD743 * VERIFY VERIFY	PSHS LDA LDU LDX STX BSR STU LDA STA LDA BNE CLRB BNE PULS RDE LDA BNE CLRB BNE CLRB BNE CMPA LBNE CMPA LBNE CMPA LBNE CMPA LBNE LDB	U,X,B,A #\$82 DCOPC DCOPT DCOPT DCOPT DSNCON DCBPT DSNCON DCBPT BSNCON DCBPT JSNCON DCSTA LD743 ,X+ ,U+ LD743 LD743 LD737 A,B,X,U ATTCTR LD6FB #2*36 LD70B AND #\$AA LD75A #\$88	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS  DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3140 3141 3142 3143 3144 3145 3146 3147 3148 3151 3151 3152 3151 3152 3151 3152 3153 3154 3157 3156 3157 3156 3157 3156 3157 3156 3157 3156 3157 3156 3157 3156 3157 3156 3157 3156 3157 3156 3157 3156 3157 3156 3157 3156 3157 3157 3157 3157 3157 3157 3157 3157	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D728 9F EE D728 9F EE D728 86 Ø7 ØØ D738 97 EA D739 97 EA D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D730 5A D738 26 Ø6 D734 26 ØD D736 5F D742 39 D743 35 56 D744 26 B1 D746 C6 48 D747 C7 Ø7 D748 18 AA D751 27 Ø7 D753 53 D754 81 88 D756 1Ø 26 DB 1D D756 F7 Ø9 87	LD737  LD742 LD743  * VERIFY  VERIFY	PSHS LDA LDU LDX STA LDU LDX STX BSR STU LDA LDA LDA BNE CLRB LDA BNE DECB BNE PULS RTS DEC BNE BNE CMPA CLRB LDB BRA COMMM. CLRB CMPA CLRB CMPA CMPA CMPA CMPA CMPA CMPA CMBA CMPA CMBA CMBA CMBA CMBA CMBA CMBA CMBA CMB	U,X,B,A #\$92 DCOPC DCOPT DCOPT DCOPT DSKCON DCBPT DSKCON DCBPT #\$93 DCOPC DCSTA LD743 LD743 LD743 LD743 LD743 LD754 LD756 AND #\$AA LD756 #\$88 LB277 DVERFL GETNCH	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS RESTORE REGISTERS  RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG
3126 3127 3128 3129 3131 3132 3133 3134 3135 3136 3137 3140 3141 3144 3145 3146 3147 3148 3149 3151 3152 3153 3154 3155 3156 3157 3158 3157 3158 3157 3158 3160 3161 3161 3161 3161 3161 3161 3161	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D730 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D73B 26 Ø6 D735 5A D73E 26 F7 D740 35 56 D742 39 D743 35 56 D745 7A Ø9 88 D748 26 B1 D74A C6 48 D74B 26 B1 D74A C6 48 D74C 2Ø BD  D74F 81 AA D751 27 Ø7 D753 53 D754 81 88 D756 1Ø 26 DB 1D D756 F7 Ø9 87 D750 ØE 9F	LD737  LD742 LD743  * VERIFY VERIFY  LD75A  * DSKCON	PSHS LDA LDU LDX STX BSR LDA STA LDU LDA STB LDA BNE CLRB BNE PULS DEC BNE LDB BNE CMPA LBNE CMPA LBNE CMPA LBNE CMPA LBNE LDB	U, X, B, A #\$82 DCOPC DCOPC DCOPT #DBUF1 DCSPT DSKCON DCSPT #\$83 DCOPC DCSTA LD743  ,X+ ,U+ LD743  LD737 A, B, X, U A, B, X, U ATICTR LD6FB #2*36 LD70B #\$AA LD75A #\$AB LD75A #\$88 LB277 DVERFL GETNCH INE	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS RESTORE REGISTERS RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3140 3141 3142 3143 3144 3145 3146 3147 3150 3151 3152 3153 3155 3156 3157 3158 3159 3161 3162 3161 3162 3163 3161 3162 3163 3161 3162 3163 3161 3162 3162	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72A 8D 67 ØØ D73A 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D73B 26 Ø6 D73B 26 Ø6 D73B 55 D74E 26 F7 D74Ø 35 56 D745 7A Ø9 88 D748 26 ØB D748 7A Ø9 88 D748 26 ØB D748 7A Ø9 88 D748 7A ØB D750 ØB D75	LD737  LD742 LD743  * VERIFY VERIFY  LD75A  * DSKCON	PSHS LDA LDU LDX STX BSR STDA LDA STA LDA STA LDA LDA STA LDA BNE CLRB BNE DECB BNE PULS PULS BNE LDB BRA CMMA CLRB BRA CMMA LBB CMPA BRA LBB CMPA LBB BRA LCHPA LBB BRA LCHPA LBB LBB LCHPA LBB LBB LCHPA LBB LBB LCHPA LBB LCHPA LBB LCHPA LBB LCHPA LBB LCHPA LBB LCHPA LBB LBB LCHPA LBB L	U,X,B,A #\$82 DCOPC DCOPT #DBUF1 DCOPT DSKCON DCBPT #\$83 DCOPC DCSTA LD743 ,X+ ,U+ LD743 LD743 LD737 A,B,X,U ATTCTR LD6FB #2*36 LD70B AND #\$AA LD75A #\$88 LB277 DVERFL GETNCH INE U,Y,X,B,A	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3140 3141 3142 3143 3144 3145 3146 3147 3148 3151 3152 3153 3154 3155 3155 3156 3157 3158 3159 3160 3161 3161 3162 3163 3166 3161 3162 3163 3166 3161 3166 3161 3166 3161 3166 3161 3166 3161 3166 3161 3166 3	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D732 96 FØ D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D73B 26 Ø6 D73D 5A D73B 26 F7 D74W 35 56 D742 39 D748 35 56 D742 39 D748 26 B1 D740 20 BD D740 20 BD D740 35 56 D741 39 D742 39 D743 35 56 D742 39 D743 35 56 D745 7A Ø9 88 D748 26 B1 D74A C6 48 D74C 2Ø BD D74F 81 AA D751 27 Ø7 D753 53 D754 81 88 D756 1Ø 26 DB 1D D75A F7 Ø9 87 D75D ØE 9F	LD737  LD742 LD743  * VERIFY VERIFY  LD75A  * DSKCON	PSHS LDA LDU LDX STA LDU LDX STX BSR STU LDA STA LDA BNE CLRB LDA BNE DECB BNE PULS RTS DEC BNE BNE CMPA BNE CMPA BNE CMPA BNE CMPA BNE LDA COMMB CLRB LDB LDB LDB LDB LDB LDB LDB LDB LDB LD	U,X,B,A #\$92 DCOPC DCOPC DCOPT #DBUF1 DCOPT DSKCON DCBPT #\$93 DCOPC DCSTA LD743 LD743 LD743 LD743 LD737 A,B,X,U ATTCTR LD6FB #2*36 LD70B AND #\$AA LD75A #\$88 LB277 DVERFL GETNCH LNE U,Y,X,B,A	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO DSKCON VARIABLE GO DSKCON VARIABLE GO DSKCON VARIABLE GO DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS  DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY FEROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND
3126 3127 3128 3129 3131 3132 3133 3134 3135 3136 3137 3140 3141 3144 3145 3146 3147 3148 3149 3150 3151 3151 3152 3153 3155 3156 3157 3158 3157 3158 3160 3161 3161 3161 3161 3161 3161 3161	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72E 86 Ø3 D73Ø 97 EA D734 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D738 26 Ø6 D738 26 Ø6 D735 5A D73E 26 F7 D74Ø 35 56 D742 39 D743 35 56 D745 7A Ø9 88 D748 26 B1 D74A C6 48 D74B 26 WB D74B 27 Ø7 D75B 37 Ø7 D75B 38 D75B 38 D75B 58 D75B 58 D75B 58 D75B 58 D75B 18 18	LD737  LD742 LD743  * VERIFY VERIFY  LD75A  * DSKCON	PSHS LDA LDU LDX STA LDU LDX STX BSR LDA	U, X, B, A #\$82 DCOPC DCOPC DCOPT #DBUF1 DCSPT DSKCON DCBPT #\$63 DCOPC DCSTA LD743  , X+ , U+ LD743  LD743  LD737  A, B, X, U A, B, X, U ATCTR LD6FB #2*36 LD7ØB AND #\$AA LD75A #\$88 LB277 DVERFL GETNCH INE U, Y, X, B, A #\$95 A	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO READ SECTOR RESTORE WRITE BUFFER WRITE OP CODE SAVE IN DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS BECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY ERROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK
3126 3127 3128 3129 3130 3131 3132 3133 3134 3135 3136 3137 3140 3141 3142 3143 3144 3145 3146 3147 3148 3151 3152 3153 3154 3155 3155 3156 3157 3158 3159 3160 3161 3161 3162 3163 3166 3161 3162 3163 3166 3161 3166 3161 3166 3161 3166 3161 3166 3161 3166 3161 3166 3	D71D 34 56 D71F 86 Ø2 D721 97 EA D723 DE EE D725 8E Ø7 ØØ D728 9F EE D72A 8D 33 D72C DF EE D72A 8D 6F Ø D73A 26 ØD D736 5F D737 A6 8Ø D739 A1 CØ D73B 26 Ø6 D73B 26 Ø6 D73B 55 D74E 26 F7 D74W 35 56 D745 7A Ø9 88 D748 26 ØB D748 7A Ø9 ØB D748 7A Ø9 ØB D748 7A Ø9 ØB D748 7A ØB D758 7A ØB D759 7A ØB D750 7A ØB D751 7A ØB D751 7A ØB D753 7A ØB D754 81 88 D756 1Ø 26 DB 1D D75A F7 Ø9 ØF D755 34 76 D761 86 Ø5 D763 34 Ø2 D765 7F Ø9 Ø5	LD737  LD742 LD743  * VERIFY VERIFY  LD75A  * DSKCON	PSHS LDA LDU LDX STX BSR STDA LDA STA LDA STA LDA LDA STA LDA LDA LDA LDA LDA LDA LDA LDA LDA LD	U, X, B, A #\$82 DCOPC DCOPC DCOPT #DBUF1 DCSPT DSKCON DCBPT #\$63 DCOPC DCSTA LD743  , X+ , U+ LD743  LD743  LD737  A, B, X, U A, B, X, U ATCTR LD6FB #2*36 LD7ØB AND #\$AA LD75A #\$88 LB277 DVERFL GETNCH INE U, Y, X, B, A #\$95 A	READ OPERATION CODE STORE TO DSKCON PARAMETER POINT U TO WRITE BUFFER ADDRESS * ADDRESS OF VERIFY BUFFER * TO DSKCON VARIABLE GO DSKCON VARIABLE GO DSKCON VARIABLE GO DSKCON VARIABLE GO DSKCON VARIABLE CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR CHECK 256 BYTES GET BYTE FROM WRITE BUFFER COMPARE TO READ BUFFER BRANCH IF NOT EQUAL * DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE RESTORE REGISTERS  RESTORE REGISTERS  DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT 'VERIFY FEROR' JUMP TO ERROR HANDLER  OFF FLAG = Ø OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND

3169	D76A 8E D8 9D		LDX	#LD89D	POINT X TO DRIVE ENABLE MASKS
3170	D76D B6 Ø9 86		LDA	DRGRAM	GET DSKREG IMAGE
3171	D77Ø 84 A8			#\$A8	KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE
3172	D772 AA 85		ORA	B,X	'OR' IN DRIVE SELECT DATA
3173	D774 8A 2Ø		ORA	#\$20	'OR' IN DOUBLE DENSITY
3174	D776 D6 EC		LDB	DCTRK	GET TRACK NUMBER
3175	D778 C1 16		CMPB		PRECOMPENSATION STARTS AT TRACK 22
3176	D77A 25 Ø2		BLO	LD77E	BRANCH IF LESS THAN 22
3177	D77C 8A 1Ø		ORA	#\$10	TURN ON WRITE PRECOMPENSATION IF >= 22
3178	D77E 1F 89	LD77E	TFR	A,B	SAVE PARTIAL IMAGE IN ACCB
3179	D78Ø 8A Ø8		ORA	#\$Ø8	'OR' IN MOTOR ON CONTROL BIT
3180	D782 B7 Ø9 86		STA	DRGRAM	SAVE IMAGE IN RAM
3181	D785 B7 FF 40		STA	DSKREG	PROGRAM THE 1793 CONTROL REGISTER
3182	D788 C5 Ø8			#\$Ø8	= WERE MOTORS ALREADY ON?
3183	D78A 26 Ø6		BNE	LD792	= DON'T WAIT FOR IT TO COME UP TO SPEED IF ALREADY ON
3184	D78C BD A7 D1		JSR	LA7D1	* WAIT A WHILE
3185	D78F BD A7 D1		JSR	LA7D1	* WAIT SOME MORE FOR MOTOR TO COME UP TO SPEED
3186	D792 8D 3D	LD792	BSR	LD7D1	WAIT UNTIL NOT BUSY OR TIME OUT
3187	D794 26 ØA		BNE	LD7AØ	BRANCH IF TIMED OUT (DOOR OPEN. NO DISK, NO POWER. ETC.)
3188	D796 ØF FØ		CLR	DCSTA	CLEAR STATUS REGISTER
3189	D798 8E D8 95		LDX	#LD895	POINT TO COMMAND JUMP VECTORS
3190	D79B D6 EA		LDB	DCOPC	GET COMMAND
3191	D79D 58		ASLB		2 BYTES PER COMMAND JUMP ADDRESS
3192	D79E AD 95		JSR		GO DO IT
3193	D7AØ 35 Ø2	LD7AØ	PULS		GET RETRY COUNT
3194	D7A2 D6 FØ		LDB	DCSTA	GET STATUS
3195	D7A4 27 ØB		BEQ	LD7B1	BRANCH IF NO ERRORS
3196	D7A6 4A		DECA		DECREMENT RETRIES COUNTER
3197	D7A7 27 Ø8		BEQ	LD7B1	BRANCH IF NO RETRIES LEFT
3198	D7A9 34 Ø2		PSHS	A	SAVE RETRY COUNT ON STACK
3199	D7AB 8D ØB		BSR	LD7B8	RESTORE HEAD TO TRACK Ø
3200	D7AD 26 F1		BNE	LD7AØ	BRANCH IF SEEK ERROR
3201	D7AF 20 B4		BRA	LD765	GO TRY COMMAND AGAIN IF NO ERROR
3202	D7B1 86 78	LD7B1	LDA	#120	120*1/60 = 2 SECONDS (1/60 SECOND FOR EACH IRQ INTERRUPT)
3203	D7B3 B7 Ø9 85		STA	RDYTMR	WAIT 2 SECONDS BEFORE TURNING OFF MOTOR
3204	D7B6 35 F6		PULS	A,B,X,Y,U,PC	RESTORE REGISTERS - EXIT DSKCON
3205		* RESTOR		D TO TRACK Ø	
3206	D7B8 8E Ø9 7E	LD7B8		#DRØTRK	POINT TO TRACK TABLE
3207	D7BB D6 EB		LDB	DCDRV	GET DRIVE NUMBER
3208	D7BD 6F 85		CLR	В,Х	ZERO TRACK NUMBER
3209	D7BF 86 Ø3		LDA	#\$Ø3	* RESTORE HEAD TO TRACK Ø, UNLOAD THE HEAD
3210	D7C1 B7 FF 48		STA	FDCREG	* AT START, 30 MS STEPPING RATE
3211	D7C4 1E 88		EXG	A,A	= '
3212	D7C6 1E 88		EXG	A,A	= WAIT FOR 1793 TO RESPOND TO COMMAND
3213	D7C8 8D Ø7		BSR	LD7D1	WAIT TILL DRIVE NOT BUSY
3214	D7CA 8D 24		BSR	LD7FØ	WAIT SOME MORE
3215	D7CC 84 10		ANDA	#\$10	1793 STATUS : KEEP ONLY SEEK ERROR
3216	D7CE 97 FØ		STA	DCSTA	SAVE IN DSKCON STATUS
3217	D7DØ 39	LD7DØ	RTS		
3217 3218	D7DØ 39	LD7DØ * WAIT F		E 1793 TO BECOME UNBUSY. IF IT DOE	S NOT BECOME UNBUSY,
	D7DØ 39	* WAIT F	OR TH	E 1793 TO BECOME UNBUSY. IF IT DOE TERRUPT AND ISSUE A DRIVE NOT REA	
3218	D7DØ 39 D7D1 9E 8A	* WAIT F	OR TH	TERRUPT AND ISSUE A DRIVE NOT REA	DY 1793 ERROR.
3218 3219		* WAIT F * FORCE LD7D1	OR TH AN IN LDX	TERRUPT AND ISSUE A DRIVE NOT REA ZERO	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT
3218 3219 3220	D7D1 9E 8A	* WAIT F * FORCE	OR TH AN IN LDX	TERRUPT AND ISSUE A DRIVE NOT REA	DY 1793 ERROR.
3218 3219 3220 3221	D7D1 9E 8A D7D3 3Ø 1F	* WAIT F * FORCE LD7D1	OR TH AN IN LDX LEAX	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT
3218 3219 3220 3221 3222	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8	* WAIT F * FORCE LD7D1	FOR TH AN IN LDX LEAX BEQ LDA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST
3218 3219 3220 3221 3222 3223	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48	* WAIT F * FORCE LD7D1	FOR TH AN IN LDX LEAX BEQ LDA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT
3218 3219 3220 3221 3222 3223 3224	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1	* WAIT F * FORCE LD7D1	FOR TH AN IN LDX LEAX BEQ LDA BITA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT
3218 3219 3220 3221 3222 3223 3224 3225	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5	* WAIT F * FORCE LD7D1	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT
3218 3219 3220 3221 3222 3223 3224 3225 3226	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39	* WAIT F * FORCE LD7D1 LD7D3	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1, X LD7DF FDCREG #\$01 LD7D3	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 86 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ	* WAIT F * FORCE LD7D1 LD7D3	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48	* WAIT F * FORCE LD7D1 LD7D3	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA STA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1, X LD7DF FDCREG #\$Ø1 LD7D3 #\$DØ FDCREG	DY 1793 ERROR. GET ZERO TO X. REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER 1F NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 87 FF 48 D7E4 1E 88	* WAIT F * FORCE LD7D1 LD7D3	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA STA EXG	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3 #\$DØ FFDCREG A,A	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER 1F NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 86 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88	* WAIT F * FORCE LD7D1 LD7D3	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA STA EXG EXG	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48	* WAIT F * FORCE LD7D1 LD7D3	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA STA EXG EXG LDA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$Ø1 LD7D3 #\$DØ FDCREG A,A A,A FDCREG	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 B6 FF 48 D7E8 B6 FF 48 D7E8 B6 FF 48	* WAIT F * FORCE LD7D1 LD7D3	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA STA EXG EXG LDA LDA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1.X LD7DF FDCREG #\$00 #\$DØ FDCREG A,A A,A FDCREG #\$88	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  1F NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY   * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  *  RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7EB 86 FF 48 D7EB 86 8Ø D7ED 97 FØ D7EF 39	* WAIT F * FORCE LD7D1 LD7D3 LD7DF	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA EXG EXG LDA LDA STA RTS	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$800 DCSTA	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 86 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 87 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7E8 86 FF 48 D7E8 86 FF 48 D7EB 97 FØ D7EF 39 D7FØ 8E 22 2E	* WAIT F * FORCE LD7D1 LD7D3 LD7D5 LD7DF  * MEDIUN LD7FØ	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA STA EXG EXG LDA STA RTS A DELA LDX	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$800 DCSTA	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY   * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235 3236 3237	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E8 86 8Ø D7E8 86 8Ø D7E8 97 FØ D7E9 97 FØ D7FØ 8E 22 2E D7FØ 3Ø 1F	* WAIT F * FORCE LD7D1 LD7D3 LD7D5 LD7DF  * MEDIUN LD7FØ	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE EXG EXG EXG LDA LDA STA EXG EXG LDA LDA STA EXG LDA LDA STA LDA STA LDA STA LDA LDA LDA LDA LDA STA LDA LDA LDA LDA STA EXG LDA LDA LDA LDA LDA LDA LDA LDA LDA LDA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A FDCREG #\$80 DCSTA  Y #8750 -1,X	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER 1F NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND
3218 3219 3220 3221 3222 3223 3224 3227 3226 3227 3230 3231 3232 3233 3234 3235 3235 3236 3237 3238	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E8 B6 FF 48 D7EB 86 68 D7EB 97 FØ D7EF 39 D7FF 39 D7FF 8E 22 E D7F3 3Ø 1F D7F5 26 FC	* WAIT F * FORCE LD7D1 LD7D3 LD7D5 LD7DF  * MEDIUN LD7FØ	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA STA EXG LDA STA RTS LDA STA LDA STA RTS LDA RTS LDA RTS LDA RTS LDA RTS RTS RTS RTS RTS RTS RTS RTS RTS RTS	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA Y #8750	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY   * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235 3236 3237 3236	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E8 86 8Ø D7E8 86 8Ø D7E8 97 FØ D7E9 97 FØ D7FØ 8E 22 2E D7FØ 3Ø 1F	* WAIT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7F7	FOR TH AN IN LDX LEAX BEQ LDA BITA BNE RTS LDA EXG EXG LDA STA RTS 1 DELA LDX LEAX LDX LEAX RTS	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,FDCREG #\$80 DCSTA Y #8750 -1,X LD7F3	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER 1F NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3231 3232 3231 3232 3233 3234 3235 3236 3237 3238 3239	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E8 86 8Ø D7ED 97 FØ D7EF 39 D7FF 39 D7FØ 8E 22 2E D7F3 3Ø 1F	* WAIT F * FORCE LD701 LD703  LD7DF  * MEDIUN LD7F3  * READ (0	FOR THE AN IN LOX LEAX BEQ LDA BITA EXG EXG LDA LDA RTS TA LDX LEAX LEAX LEAX LEAX LEAX BRTS CONE SE STORE THE ANDRE SE STORE SE	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA Y #8750 -1,X LD7F3 CTOR	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  *  RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3230 3231 3232 3233 3234 3235 3236 3237 3238 3238 3239 3241	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7EB 86 FF 48 D7EB 86 8Ø D7ED 97 FØ D7EF 39 D7FF 86 E2 2E D7F3 3Ø 1F D7F5 26 FC D7F7 39	* MAIT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7F3  * READ ( LD7F8	FOR THAN IN LDX LEAX BEQ LDA BITA EXG EXG LDA LDA STA LDA LDA LDX LEAX BRES RTS SINE SE LDA LDX LEAX BRES SINE SE LDA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA  Y #8750 -1,X LD7F3  CTOR #\$80	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR)
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3231 3232 3231 3232 3233 3234 3235 3236 3237 3238 3239	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E8 86 8Ø D7ED 97 FØ D7EF 39 D7FF 39 D7FØ 8E 22 2E D7F3 3Ø 1F	* WAIT F * FORCE LD7D1 LD7D3  LD7D5  * MEDIUN LD7F0 LD7F3  * READ C LD7F8 LD7FA	FOR THAN IN LDX BEQ LDA BITA BBHE RTS LDA STA LDA STA LDA STA LDA RTS LDA RTS LDA LDX BBHE RTS SONE SE COMPX	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FOCREG A,A A,FOCREG #\$80 DCSTA Y #8750 -1,X LD7F3  CTOR #\$80	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  *  RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3238 3231 3232 3233 3234 3235 3236 3237 3238 3237 3238 3237 3238 3237 3238 3237 3244 3244 3244 3243	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7E8 86 FF 48 D7E8 86 SØ D7E9 97 FØ D7E7 39 D7FØ 8E 22 2E D7FØ 39 D7FØ 8E 22 2E D7FØ 39 D7FØ 86 8Ø D7FØ 86 80 D7FØ 86 80 D7FØ 86 80 D7FØ 86 80	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7F0 LD7F3  * READ ( LD7F8 LD7F8 LD7F8 LD7F8 LD7F8 * WRITE	FOR THAN IN LDX LEAX LEAX LDA BITA BENE EXG LDA STA LDA STA LDA LDA LDA LDA LDA LDA LDA LDA LDA LD	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA  Y #8750 -1,X LD7F3  CTOR #\$880 #\$86A0 ECTOR	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3240 3241	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 86 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7EB 86 FF 48 D7EB 86 8Ø D7ED 97 FØ D7EF 39 D7FF 86 BØ	* WAIT F * FORCE LD7D1 LD7D3  LD7D5  * MEDIUN LD7F0 LD7F3  * READ C LD7F8 LD7FA	FOR THAN IN LDX BEQ LDA BITA BITA BITA STA EXG EXG LDA STA RTS LDA LDA CMPX CMPX CDA LDA LDA LDA LDA LDA LDA LDA LDA LDA L	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA  Y #8750 -1,X LD7F3  CTOR #\$80 #\$80 #\$86A0 ECTOR	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR)
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3241 3242 3243 3244 3244 3244 3244	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 86 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E8 B6 FF 48 D7E8 B6 FF 48 D7EB 86 FF 48 D7EB 7F 80 D7EF 39 D7FØ 8E 22 2E D7F3 3Ø 1F D7F5 26 FC D7F7 39 D7F8 86 8Ø D7FA 8C D7FB 86 AØ D7FA 8C	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7F0 LD7F3  * READ ( LD7F8 LD7F8 LD7F8 LD7F8 LD7F8 * WRITE	FOR THAN IN LDX BEQ LDA BITA BITA BITA RTS LDA CMPX BNE RTS L	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X L070F FDCREG #\$01 L0703  #\$DØ FOCREG A,A A,A FOCREG #\$80 DCSTA Y #8750 -1,X LD773  CTOR #\$80 #\$880 ECTOR #\$880 ECTOR #\$80 #\$86A0 ECTOR #\$480 A	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3238 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3241 3242 3243 3244 3244 3244	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7E8 86 FF 48 D7E8 86 FF 48 D7E9 39 D7F9 8E 22 2E D7F3 3Ø 1F D7F5 26 FC D7F7 39 D7F8 86 8Ø D7FA 8C D7FB 86 AØ D7FA 8C D7FB 86 AØ D7FD 34 Ø2 D7FF 8E Ø9 7E	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7F0 LD7F3  * READ ( LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LDX LEAX BEQ LDA BITA EXG EXG LDA LDA LDA LEAX TO BE SE LDA RTS COMPX ONE S LDA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA Y #8750 -1,X LD7F3  CTOR #\$880 #\$860 ESTOR #\$880 #\$860 ESTOR #\$100 #\$860 #\$100	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES  \$40 IS WRITE FLAG (1793 WRITE SECTOR) SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM
3218 3219 3220 3221 3222 3223 3224 3227 3226 3227 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3241 3242 3243 3244 3245 3244	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 86 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7EB 86 FF 48 D7EB 86 8Ø D7ED 97 FØ D7EF 39 D7FF 39 D7FF 86 8Ø D7FF 86 99 7E D8Ø2 D6 EB	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7F0 LD7F3  * READ ( LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ LDA BBITA BBE RTS A LDA LDA LDA LDA LDA CMPX ONE S LDA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X L070F FDCREG #\$01 L0703  #\$DØ FOCREG A,A A,FOCREG #\$80 DCSTA Y #8750 -1,X LD773  CTOR #\$80 #\$880 ECTOR #\$880 ECTOR #\$80 A	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY   * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  *  RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK  POINT X TO TRACK NUMBER TABLE IN RAM  GET DRIVE NUMBER
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3238 3239 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3241 3242 3243 3244 3245 3244 3244 3244 3244	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E8 B6 FF 48 D7E8 B6 FF 48 D7E8 B6 FF 48 D7E9 97 FØ D7F7 39 D7F8 8C 22 E D7F3 3Ø 1F D7F5 26 FC D7F7 39 D7F8 8C 8Ø D7FA 8C D7FF 8E Ø9 D7E BØØ D7FF 8E Ø9 D7FF 8E Ø9 D7FF 8E Ø9 D7E BØØ D7FF 8E Ø9 D7E BØØ D7E BØØ D7E BØØ D7FB BE Ø9 D7FB BE Ø9 D7E BØØ D7FB BE Ø9 D7E BØØ D8ØØ D8ØØ D8ØØ D8ØØ D8ØØ D8ØØ D8ØØ D	* WALT F * FORCE LD7D1 LD7D3  LD7D7  * MEDIUN LD7FØ LD7FØ * READ ( LD7F8 LD7F8 LD7FA * WRITE LD7FB	FOR THAN IN LEAX LEAX BEQ LDA BITA BITA STA EXG EXG EXG LDA LDA STA LDA CMPX ONE S LOX LDX LDX LDX LDX LDX LDX LDX LDX LDX LD	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LO7DF FDCREG #\$01 LO7D3  #\$DØ FOCREG A,A A,FOCREG #\$880 DCSTA Y #8750 -1,X LD7F3  CTOR #\$880 ECTOR #\$880 ECTOR #\$880 ECTOR #\$880 A #0RØTRK DCDRV	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK  POINT X TO TRACK NUMBER TABLE IN RAM  GET DRIVE NUMBER  POINT X TO CORRECT DRIVE'S TRACK BYTE
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3238 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3241 3242 3243 3244 3244 3244 3244 3244	D701 9E 8A D703 3Ø 1F D705 27 Ø8 D707 86 FF 48 D70A 85 Ø1 D70C 26 F5 D70E 39 D70F 86 DØ D72E B7 FF 48 D764 1E 88 D766 1E 88 D768 86 FF 48 D768 86 FF 48 D768 97 FØ D769 97 FØ D767 39 D776 86 B2 22 2E D777 39 D778 86 80 D768 86 80 D769 86 80 D769 87 FØ D769 87 FØ D769 88 22 2E D778 3Ø 1F D769 86 80 D769 86 86 D804 30	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7F0 LD7F3  * READ C LD7F8 LD7F8 LD7FA * WALT LD7FB	FOR THAN IN LEAX LOX LEAX BEQ LDA STA EXG LDA LDA LDA LDA COMP STA LOX LEAX ONE SE LDA	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA  Y #8750 -1,X LD7F3  CTOR #\$80 #\$886A0 ECTOR #\$A0 A HOROTIK DCDRV ,X	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR) SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3241 3242 3243 3244 3244 3245 3244 3245 3248 3247 3248 3249	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7EB 86 68 Ø D7ED 97 FØ D7EF 39 D7FR 86 BØ D7FR BØ D8Ø D8Ø D8Ø D8Ø D8Ø D8Ø D8Ø D8Ø D8Ø D8	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7F8 LD7F8 LD7FA * WRITE LD7FB	FOR THAN IN LOX LEAX BEQ LDA BITA BNE EXG EXG LDA STA RTS LDA STA RTS LDA STA CMPX CONE STA RTS LDA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 CSTA Y #8750 -1,X LD7F3  CTOR #\$80 #\$86AØ ECTOR #\$80 A #DRØTRK DCDRV ,X FDCREG+1	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR) SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 TRACK REGISTER
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3241 3242 3243 3244 3244 3244 3244 3244	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E8 B6 FF 48 D7EB 86 87 D7EF 39 D7FF 39 D7FØ 8E 22 2E D7F7 39 D7F8 86 80 D7E7 B7 FØ D7F7 39 D7F8 86 80 D7F9 36 FC D7F7 39 D7F8 86 80 D7F9 36 FC D7F7 39 D7F8 86 80 D7F9 34 Ø2 D7FF 8E Ø9 7E D8Ø4 3A D8Ø5 E6 84 D8Ø7 F7 FF 49 D8Ø4 D1 EC	* WALT F * FORCE LD7D1 LD7D3  LD7D7  * MEDIUN LD7FØ LD7FØ * READ (LD7F8 LD7F8 LD7FA * WRITE LD7FB	FOR THAN IN LEAX BEQ LDA BITA EXG EXG EXG LDA STA LDA STA LDA STA LDA STA LDA STA LDA LDA LDA LDA LDA LDA LDA LDA LDA LD	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X L070F FDCREG #\$901 L0703  #\$D0 FOCREG A,A A,A FOCREG #\$80 OCSTA Y #8750 -1,X LD773  CTOR #\$880 #\$86A0 ECTOR #\$880 A #0R0TRK DCDRV ,X FDCREG+1 DCTRK	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$8Ø IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$AØ IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM  GET DRIVE NUMBER  POINT X TO CORRECT DRIVE'S TRACK BYTE  GET TRACK NUMBER OF CURRENT HEAD POSITION  SEND TO 1793 TRACK REGISTER  COMPARE TO DESIRED TRACK
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3241 3242 3244 3242 3244 3244 3245 3244 3245 3246 3247 3248 3259 3250 350 350 350 350 350 350 350 3	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7E8 86 FF 48 D7EB 86 BØ D7EB 7 FØ D7EB 86 BØ D7FB 86 BØ D7FF 86 BØ D8ØF FF FF 49 D8ØF D8ØF FF FF 49 D8ØF D8ØF TF TF TF 49 D8ØF D8ØF TF	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7F0 LD7F3  * READ C LD7F8 LD7FA * WRITE LD7FB	FOR THAN IN LOX LEAX BEQ LDA BITA EXG EXG EXG LDA STA LDX BNE RTS LDA LDX BNE STA LDA LDX LDA LDX LDA LDX LDA LDX LDB ABX LDB ABX LDB STB BEQ CMPX BNE STB BEQ BEQ BEQ	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA  Y #8750 -1,X LD7F3  CTOR #\$80 #\$86A0 ECTOR #\$A0 A A FOCREG #\$A0 A FOCREGETOR #\$A0 A A FOCREGETOR #\$A0 A A FOCREGETOR #\$A0 A A A A A A A A A A A A A A A A A A	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  *  RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$8Ø IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$AØ IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK  POINT X TO TRACK NUMBER TABLE IN RAM  GET DRIVE NUMBER  POINT X TO CORRECT DRIVE'S TRACK BYTE  GET TRACK NUMBER OF CURRENT HEAD POSITION  SEND TO 1793 TRACK REGISTER  COMPARE TO DESIRED TRACK  BRANCH IF ON CORRECT TRACK
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3241 3242 3243 3244 3245 3244 3245 3246 3257 3268 327 327 328 329 329 329 329 329 329 329 329	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7EB 86 68 D7EB 97 FØ D7EF 39 D7FR 86 BØ D7FR 86 BØ D7ED 97 FØ D7EF 39 D7FR 86 8Ø D7FR 86 BØ D7FR BØ D8Ø D7FR BØ D7FR BØ D8Ø D7FR BØ D8Ø D7FR BØ D8Ø D7FR BØ D7FR	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7F7  * READ (LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ BLDA BITA STA EXG EXG EXG LDA LDA LDA LDA BRITS LDA LDA LDA LDA BRITS DIVE SE LDA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA Y #8750 -1,X LD7F3  CTOR #\$880 ECTOR #\$80 A #ORØTRK DCDRV ,X FDCREG+1 DCTRV	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR) SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 TRACK REGISTER COMPARE TO DESIRED TRACK BRANCH IF ON CORRECT TRACK BEANCH IF ON CORRECT TRACK BRANCH IF ON CORRECT TRACK BEANCH IF ON CORRECT TRACK BEANCH IF ON CORRECT TRACK BET TRACK DESIRED
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3231 3232 3231 3232 3233 3234 3235 3236 3240 3241 3242 3243 3244 3244 3245 3246 3247 3248 3249 3251 3266	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7EB 86 8Ø D7ED 97 FØ D7EF 39 D7FØ 8E 22 2E D7FØ 3Ø 1F D7FØ 3Ø 3Ø 1F D7FØ 3Ø 1F D7	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7F7  * READ (LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ BLDA BITA STA EXG EXG EXG LDA LDA LDA LDA BRITS LDA LDA LDA LDA BRITS DIVE SE LDA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$901 LD7D3  #\$D0 FOCREG A,A A,A FOCREG #\$80 DCSTA Y #8750 -1,X LD7F3  CTOR #\$80 ECTOR #\$80 #\$86A0 ECTOR #\$80 A #0R0TRK DCDRV ,X FDCREG+1 DCTRK FDCREG+3	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER  POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 TRACK REGISTER  COMPARE TO DESIRED TRACK BRANCH IF ON CORRECT TRACK GET TRACK DESIRED SEND TO 1793 DATA REGISTER
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3241 3242 3243 3244 3245 3246 3247 3248 3250 3251 3252 3253 3255	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7E8 86 BØ D7E9 7 FØ D7F7 39 D7F8 86 8Ø D7F7 39 D7F8 86 8Ø D7F7 39 BP D7F8 86 8Ø D7F8 86 AØ D7F7 SE BØ7 FF D8Ø7 FF FF D8Ø7 FF 49 D8Ø7 FF FF D8Ø7 FF FF D8Ø7 FF 48 D8Ø8 PF FF D8Ø8 PF	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7F7  * READ (LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ BLDA BITA STA EXG EXG EXG LDA LDA LDA LDA BRITS LDA LDA LDA LDA BRITS DIVE SE LDA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$901 LD7D3  #\$D0 FOCREG A,A A,A FOCREG #\$80 DCSTA Y #8750 -1,X LD7F3  CTOR #\$80 ECTOR #\$80 #\$86A0 ECTOR #\$80 A #0R0TRK DCDRV ,X FDCREG+1 DCTRK FDCREG+3	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$8Ø IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES  \$AØ IS WRITE FLAG (1793 WRITE SECTOR) SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 TRACK REGISTER COMPARE TO DESIRED TRACK GET TRACK OBSIRED SEND TO 1793 DATA REGISTER SAVE IN RAM TRACK IMAGE
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3236 3237 3241 3242 3244 3245 3244 3245 3244 3245 3256 3251 3256 3257 3258	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E8 B6 B6 FF 48 D7E8 B6 B7 D7E7 39 D7F7 39 D7F8 8C D7F7 39 D7F8 8C D7F7 39 D7F8 8C D7F7 39 D7F8 B6 B8 D7E8 B6 B8 D7E8 B6 B8 D7E8 B6 B8 D7E9 D7F7 B8 D7E9 B6 B8 D7E9 B7 D7E9 B7 D7E9 B7 D7E9 B7 D7E9 B7 D7E9 B7 D7E9 B8 D7	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7F7  * READ (LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ BLDA BITA STA EXG EXG EXG LDA LDA LDA LDA BRITS LDA LDA LDA LDA BRITS DIVE SE LDA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$901 LD7D3  #\$D0 FOCREG A,A A,A FOCREG #\$80 DCSTA Y #8750 -1,X LD7F3  CTOR #\$80 ECTOR #\$80 #\$86A0 ECTOR #\$80 A #0R0TRK DCDRV ,X FDCREG+1 DCTRK FDCREG+3	DY 1793 ERROR. GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR) SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 TRACK REGISTER COMPARE TO DESIRED TRACK BRANCH IF ON CORRECT TRACK GET TRACK DESIRED SEND TO 1793 DATA REGISTER SAVE IN RAM TRACK IMAGE * SEEK COMMAND FOR 1793: DO NOT LOAD THE
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3241 3242 3243 3244 3244 3245 3246 3257 3266 327 327 328 329 329 329 329 329 329 329 329 329 329	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7E8 86 FF 48 D7EB 86 BF 48 D7EB 86 BF 48 D7EB 86 BØ D7ED 97 FØ D7F7 39 D7F8 86 Z2 ZE D7F7 39 JE SE	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7F7  * READ (LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ BLDA BITA STA EXG EXG EXG LDA LDA LDA LDA BRITS LDA LDA LDA LDA BRITS DIVE SE LDA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$901 LD7D3  #\$D0 FOCREG A,A A,A FOCREG #\$80 DCSTA Y #8750 -1,X LD7F3  CTOR #\$80 ECTOR #\$80 #\$86A0 ECTOR #\$80 A #0R0TRK DCDRV ,X FDCREG+1 DCTRK FDCREG+3	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$8Ø IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$AØ IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION  SEND TO 1793 TRACK REGISTER COMPARE TO DESIRED TRACK BRANCH IF ON CORRECT TRACK GET TRACK DESIRED  SAVE IN RAM TRACK IMAGE  * SEEK COMMAND FOR 1793: DO NOT LOAD THE  * HEAD AT START, VERIFY DESTINATION TRACK,
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3241 3242 3243 3244 3245 3246 3247 3248 3250 3251 3256 3257 3258	D701 9E 8A D703 3Ø 1F D705 27 Ø8 D707 86 FF 48 D70A 85 Ø1 D70C 26 F5 D70E 39 D7F 86 DØ D7F1 87 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7E8 86 BØ D7E9 7 FØ D7F7 39 D7F8 86 8Ø D7F7 39 D7F8 86 8Ø D7F7 39 D7F8 86 8Ø D7F7 8C D7F7 39 D7F8 86 AØ D7F8 86 AØ D7F8 86 AØ D7F8 86 AØ D7F7 8C D7F7 8E Ø9 7E D8ØZ D6 EB D8Ø4 3A D8Ø5 E6 84 D8Ø7 F7 FF 49 D8ØZ D7 IE D8ØZ D8 EC D8ØZ D7 IE D8ØZ D8 FF FF B8 D8ØZ D8 IF FF FF D8ØZ D7 FFF B8 D8ØZ D8 IF FF FF D8ØZ D8 IF FF FF D8ØZ D7 FFF FF D8ØZ D7 FFF B8 D8ØZ D8 IF FF FF D8ØZ D8 IF FF FF D8ØZ D7 FFF FF D8ØZ D7 FFF B8 D8ØZ D8 IF D7 FFF B8 D8 II D7 FFF B8	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7F7  * READ (LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ BLDA BITA STA EXG EXG EXG LDA LDA LDA LDA BRITS LDA LDA LDA LDA BRITS DIVE SE LDA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$901 LD7D3  #\$D0 FOCREG A,A A,A FOCREG #\$80 DCSTA Y #8750 -1,X LD7F3  CTOR #\$80 ECTOR #\$80 #\$86A0 ECTOR #\$80 A #0R0TRK DCDRV ,X FDCREG+1 DCTRK FDCREG+3	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES  \$40 IS WRITE FLAG (1793 WRITE SECTOR) SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 TRACK REGISTER COMPARE TO DESIRED TRACK GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 DATA REGISTER SAVE IN RAM TRACK IMAGE * SEEK COMMAND FOR 1793: DO NOT LOAD THE * HEAD AT START, VERIFY DESTINATION TRACK, * 30 MS STEPPING RATE - WAIT FOR
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3231 3231 3232 3233 3234 3235 3236 3237 3238 3240 3241 3242 3243 3244 3245 3246 3247 3248 3252 3253 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E8 B6 B7 D7E7 39 D7F7 39 D7F8 B6 B7 D7F7 39 D7F8 B6 B7 D7F8 B6 B8 D7F8 B6 B8 D7F9 B7 F8 D7F9 B7 B7 D7F9 B7 D7F9 B7 B7 D7F9	* WALT F * FORCE LD7D1 LD7D3  LD7DF  * MEDIUN LD7FØ LD7FØ LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ LDA STA CHAN S	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$901 LD7D3  #\$D0 FOCREG A,A A,A FOCREG #\$80 DCSTA Y #8750 -1,X LD7F3  CTOR #\$80 ECTOR #\$80 #\$86A0 ECTOR #\$80 A #0R0TRK DCDRV ,X FDCREG+1 DCTRK FDCREG+3	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 WRITE SECTOR)  SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM  GET DRIVE NUMBER  POINT X TO CORRECT DRIVE'S TRACK BYTE  GET TRACK NUMBER OF CURRENT HEAD POSITION  SEND TO 1793 TRACK REGISTER  COMPARE TO DESIRED TRACK  BRANCH IF ON CORRECT TRACK  GET TRACK DESIRED  SEND TO 1793 DATA REGISTER  SAVE IN RAM TRACK IMAGE  * SEEK COMMAND FOR 1793: DO NOT LOAD THE  * HEAD AT START, VERIFY DESTINATION TRACK,  * 30 MS STEPPING RATE - WAIT FOR  * VALID STATUS FROM 1793  * VALID STATUS FROM 1793  * VALID STATUS FROM 1793
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3236 3237 3238 3234 3242 3244 3245 3244 3245 3244 3245 3244 325 325 3256 3257 3258 3256 3257 3258	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7E8 86 FF 48 D7E9 39 D7F8 86 DØ D7E7 39 D7F8 86 BØ D7E7 33 Ø 1F D7F7 39 D7F8 86 8Ø D7E7 39 D7F8 86 8Ø D7F9 86 E 22 2E D7F3 3Ø 1F D7F5 26 FC D7F7 39 D7F8 86 8Ø D7FA 8C D7FB 86 AØ D7FB 86 AØ D7FD 34 Ø2 D7FF 88 Ø9 7E D8Ø2 D6 EB D8Ø4 3A D8Ø5 E6 84 D8Ø7 F7 FF 49 D8Ø6 27 1E D8Ø6 27 1E D8Ø6 96 EC D8Ø6 27 1E D8Ø8 96 EC D8Ø8 37 FF 48 D813 A7 84 D815 86 17 D817 B7 FF 48 D813 A7 84 D816 88 D81C 1E 88 D81C 1E 88 D81E 80 B1	* WALT F * FORCE LD701 LD703  LD7DF  * MEDIUN LD7F0 LD7F3  * READ ( LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEDA BITA STA EXG EXG LDA LDA LDA STA LDA LDB STB LDA STA LDB STB LDA STA LDB STB LDA STA LDB STB LDB	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA  Y #8750 -1,X LD7F3  CTOR #\$80 #\$86A0 ECTOR #\$A0 A A FDCREG+1 DCTRK DCDRV ,X FDCREG+1 DCTRK FDCREG+3 ,X #\$17 FDCREG A,A A,A A A A A A A A A A A A A A A A	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR) SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 TRACK REGISTER COMPARE TO DESIRED TRACK BRANCH IF ON CORRECT TRACK GET TRACK DESIRED SEND TO 1793 DATA REGISTER SAVE IN RAM TRACK IMAGE * SEEK COMMAND FOR 1793: DO NOT LOAD THE * HEAD AT START, VERIFY DESTINATION TRACK, * 30 MS STEPPING RATE - WAIT FOR * VALID STATUS FROM 1793 WAIT TILL NOT BUSY
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3241 3242 3243 3244 3245 3246 3247 3248 3259 3250 3251 3255 3256 3257 3258 3259 3266 3257	D701 9E 8A D703 3Ø 1F D705 27 Ø8 D707 86 FF 48 D70A 85 Ø1 D7DC 26 F5 D7DE 39 D7F 86 DØ D7F1 87 FF 48 D7E4 1E 88 D7E8 86 FF 48 D7E8 86 8Ø D7E9 97 FØ D7F7 39 D7FØ 8E 22 2E D7F3 3Ø 1F D7F5 26 FC D7F7 39 D7F8 86 8Ø D7FA 8C D7F8 86 AØ D7FA 8C D7FF 86 D9 7E D8Ø2 D6 EB D8Ø4 3A D8Ø5 E6 84 D8Ø7 F7 FF 49 D8Ø5 D6 EC D8Ø6 27 1E D8Ø6 96 EC D8Ø6 27 1E D8Ø6 96 EC D8Ø7 B7 FF 4B D8Ø3 A7 84 D815 86 17 D817 B7 FF 4B D813 A7 84 D815 86 17 D817 B7 FF 4B D814 1E 88 D816 B8 B1 D826 26 Ø8	* WALT F * FORCE LD701 LD703  LD7DF  * MEDIUN LD7F0 LD7F3  * READ ( LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ LDA BITA BNE EXG EXG LDA STA LDA LDA LDA LDA LDA STA LDA STB LDB STB	TERRUPT AND ISSUE A DRIVE NOT REA ZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$880 DCSTA  Y #8750 -1,X LD7F3  CTOR #\$80 #\$86A0 ECTOR #\$1 DCTRK DCDRV ,X FDCREG+1 DCTRK LD82C DCTRK FDCREG+3 ,X #\$17 FDCREG A,A A,A LD7D1 LD82A	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT DECREMENT LONG WAIT COUNTER IF NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST * BUSY STATUS BIT BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST * WAIT BEFORE READING 1793 * RESET INTRQ (FDC INTERRUPT REQUEST) RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE * DECREMENT DELAY COUNTER AND * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR) SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR) SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER POINT X TO CORRECT DRIVE'S TRACK BYTE GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 TRACK REGISTER COMPARE TO DESIRED TRACK GET TRACK NUMBER OF CURRENT HEAD POSITION SEND TO 1793 DATA REGISTER COMPARE TO DESIRED TRACK GET TRACK DESIRED SEND TO 1793 DATA REGISTER SAVE IN RAM TRACK IMAGE * SEEK COMMAND FOR 1793: DO NOT LOAD THE * HEAD AT START, VERIFY DESTINATION TRACK, * 30 MS STEPPING RATE - WAIT FOR * VALID STATUS FROM 1793 WAIT TILL NOT BUSY RETURN IF TIMED OUT
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3241 3242 3243 3244 3245 3246 3247 3248 3248 3249 3256 3257 3258 3256 3257 3258 3256 3257 3258 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3257 3258 3256 3257 3258 3259 3250 3250 3251 3252 3253 3256 3257 3258 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3258 3259	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E9 97 FØ D7F7 39 D7F8 8C 22 2E D7F3 3Ø 1F D7F5 26 FC D7F7 39 D7F8 8C 8C D7F8 8C 8C D7F8 8C 8C D7F8 B6 AØ D7FB 8C D7F8 B7 FF 48 D802 D6 EB D804 3A D805 E6 84 D807 F7 FF 49 D806 D1 EC D806 C7 1E D806 P7 FF F4 D807 F7 FF 49 D808 D1 EC D808 D7 F8 B6 T7 D818 B7 FF 48 D813 A7 84 D815 B6 17 D817 B7 FF 48 D813 A7 B7 FF 48 D814 B8 D816 E 88 D816 E	* WALT F * FORCE LD701 LD703  LD7DF  * MEDIUN LD7F0 LD7F3  * READ ( LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEQ LOA BITTA STA EXG EXG EXG LOA LOA STA LOX LOB BNE STA LOX LOB STA LOX LOB STA LOX LOB STA EXG BNE EXG BNE EXG BNE EXG BNE EXG BNE	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FOCREG A,A A,A FOCREG #\$880 DCSTA Y #88750 -1,X LD7F3 CTOR #\$880 #\$86A0 ECTOR #\$80 #\$80A0 ECTOR #\$1 DCTRK LD82C DCTRK FDCREG+1 DCTRK LD82C DCTRK FOCREG+3,X #\$17 FOCREG+3,X #\$17 FOCREG A,A LD7D1 LD82A LD7F0	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM  GET DRIVE NUMBER  POINT X TO CORRECT DRIVE'S TRACK BYTE  GET TRACK NUMBER OF CURRENT HEAD POSITION  SEND TO 1793 TRACK REGISTER  COMPARE TO DESIRED TRACK  BRANCH IF NO CORRECT TRACK  GET TRACK DESIRED  SEND TO 1793 DATA REGISTER  SAVE IN RAM TRACK IMAGE  * SEEK COMMAND FOR 1793: DO NOT LOAD THE  * HEAD AT START, VERIEFY DESTINATION TRACK,  * 30 MS STEPPING RATE - WAIT FOR  * VALID STATUS FROM 1793  WAIT TILL NOT BUSY  RETURN IF TIMED OUT  WAIT SOME MORE
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3236 3237 3238 3234 3245 3240 3241 3242 3244 3245 3244 3255 3256 3257 3258 3255 3256 3257 3258 3257 3258 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 86 FF 48 D7E8 86 FF 48 D7E9 39 D7F8 86 DØ D7E7 39 D7F8 86 DØ D7E7 39 D7F8 86 BØ D7E9 B7 FØ D7E7 39 D7F8 86 BØ D7E9 B8 62 22 2E D7F3 3Ø 1F D7F5 26 FC D7F7 39 D7F8 86 8Ø D7FA 8C D7FB 86 AØ D7FB 34 Ø2 D7FB 86 AØ D7FD 34 Ø2 D7FF 86 BØ D7FF 48 BØ9 7E D8ØC 27 1E D8ØE 96 EC D8ØE 28 DC	* WALT F * FORCE LD701 LD703  LD7DF  * MEDIUN LD7F0 LD7F3  * READ ( LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEDA BITA STA EXG EXG LDA LDA LDA STA LDA LDB STB LDA LDB STB LDA LDB STB LDA STA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FDCREG A,A A,A FDCREG #\$80 DCSTA  Y #8750 -1,X LD7F3  CTOR #\$80 #\$86A0 ECTOR #\$80 A A A A A A A A A A A A A A A A A A A	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  *  RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$8Ø IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$AØ IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK  POINT X TO TRACK NUMBER TABLE IN RAM  GET DRIVE NUMBER  POINT X TO CORRECT DRIVE'S TRACK BYTE  GET TRACK NUMBER OF CURRENT HEAD POSITION  SEND TO 1793 TRACK REGISTER  COMPARE TO DESIRED TRACK  BRANCH IF ON CORRECT TRACK  GET TRACK OESTRED  SEND TO 1793 DATA REGISTER  SAVE IN RAM TRACK IMAGE  * SEEK COMMAND FOR 1793: DO NOT LOAD THE  * HEAD AT START, VERIFY DESTINATION TRACK,  * 30 MS STEPPING RATE - WAIT FOR  * VALID STATUS FROM 1793  WAIT TILL NOT BUSY  RETURN IF TIMED OUT  WAIT SOME MORE  KEEP ONLY SEEK ERROR OR CRC ERROR IN ID FIELD
3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235 3236 3237 3238 3239 3240 3241 3242 3243 3244 3245 3246 3247 3248 3248 3249 3256 3257 3258 3256 3257 3258 3256 3257 3258 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3256 3257 3258 3257 3258 3256 3257 3258 3259 3250 3250 3251 3252 3253 3256 3257 3258 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3256 3257 3258 3259 3258 3259	D7D1 9E 8A D7D3 3Ø 1F D7D5 27 Ø8 D7D7 B6 FF 48 D7DA 85 Ø1 D7DC 26 F5 D7DE 39 D7DF 86 DØ D7E1 B7 FF 48 D7E4 1E 88 D7E6 1E 88 D7E8 B6 FF 48 D7E9 97 FØ D7F7 39 D7F8 8C 22 2E D7F3 3Ø 1F D7F5 26 FC D7F7 39 D7F8 8C 8C D7F8 8C 8C D7F8 8C 8C D7F8 B6 AØ D7FB 8C D7F8 B7 FF 48 D802 D6 EB D804 3A D805 E6 84 D807 F7 FF 49 D806 D1 EC D806 C7 1E D806 P7 FF F4 D807 F7 FF 49 D808 D1 EC D808 D7 F8 B6 T7 D818 B7 FF 48 D813 A7 84 D815 B6 17 D817 B7 FF 48 D813 A7 B7 FF 48 D814 B8 D816 E 88 D816 E	* WALT F * FORCE LD701 LD703  LD7DF  * MEDIUN LD7F0 LD7F3  * READ ( LD7F8 LD7F8 LD7F8 LD7FB LD7FB	FOR THAN IN LOX LEAX BEDA BITA STA EXG EXG LDA LDA LDA STA LDA LDB STB LDA LDB STB LDA LDB STB LDA STA	TERRUPT AND ISSUE A DRIVE NOT REAZERO -1,X LD7DF FDCREG #\$01 LD7D3  #\$DØ FOCREG A,A A,A FOCREG #\$880 DCSTA Y #88750 -1,X LD7F3 CTOR #\$880 #\$86A0 ECTOR #\$80 #\$80A0 ECTOR #\$1 DCTRK LD82C DCTRK FDCREG+1 DCTRK LD82C DCTRK FOCREG+3,X #\$17 FOCREG+3,X #\$17 FOCREG A,A LD7D1 LD82A LD7F0	DY 1793 ERROR.  GET ZERO TO X REGISTER - LONG WAIT  DECREMENT LONG WAIT COUNTER  IF NOT READY BY NOW, FORCE INTERRUPT  * GET 1793 STATUS AND TEST  * BUSY STATUS BIT  BRANCH IF BUSY  * FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND  * IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST  * WAIT BEFORE READING 1793  * RESET INTRQ (FDC INTERRUPT REQUEST)  RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY  SAVE DSKCON STATUS BYTE  DELAY FOR A WHILE  * DECREMENT DELAY COUNTER AND  * BRANCH IF NOT DONE  \$80 IS READ FLAG (1793 READ SECTOR)  SKIP TWO BYTES  \$A0 IS WRITE FLAG (1793 WRITE SECTOR)  SAVE READ/WRITE FLAG ON STACK POINT X TO TRACK NUMBER TABLE IN RAM  GET DRIVE NUMBER  POINT X TO CORRECT DRIVE'S TRACK BYTE  GET TRACK NUMBER OF CURRENT HEAD POSITION  SEND TO 1793 TRACK REGISTER  COMPARE TO DESIRED TRACK  BRANCH IF NO CORRECT TRACK  GET TRACK DESIRED  SEND TO 1793 DATA REGISTER  SAVE IN RAM TRACK IMAGE  * SEEK COMMAND FOR 1793: DO NOT LOAD THE  * HEAD AT START, VERIEFY DESTINATION TRACK,  * 30 MS STEPPING RATE - WAIT FOR  * VALID STATUS FROM 1793  WAIT TILL NOT BUSY  RETURN IF TIMED OUT  WAIT SOME MORE

3265 3266	D828 97 FØ D82A 35 82	STA DCSTA LD82A PULS A,PC	SAVE IN DSKCON STATUS
3267		* HEAD POSITIONED ON CORRECT TRACK	
3268 3269	D82C 96 ED D82E B7 FF 4A	LD82C LDA DSEC STA FDCREG+2	GET SECTOR NUMBER DESIRED SEND TO 1793 SECTOR REGISTER
3270	D831 8E D8 8B	LDX #LD88B	* POINT X TO ROUTINE TO BE VECTORED
3271 3272	D834 BF Ø9 83 D837 9E EE	STX DNMIVC LDX DCBPT	* TO BY NMI UPON COMPLETION OF DISK I/O AND SAVE VECTOR POINT X TO I/O BUFFER
3273	D839 B6 FF 48	LDA FDCREG	RESET INTRQ (FDC INTERRUPT REQUEST)
3274	D83C B6 Ø9 86	LDA DRGRAM	GET DSKREG IMAGE
3275 3276	D83F 8A 8Ø D841 35 Ø4	ORA #\$8Ø PULS B	SET FLAG TO ENABLE 1793 TO HALT 6809 GET READ/WRITE COMMAND FROM STACK
3277	D843 10 9E 8A	LDY ZERO	ZERO OUT Y - TIMEOUT INITIAL VALUE
3278 3279	D846 CE FF 48 D849 73 Ø9 82	LDU #FDCREG COM NMIFLG	U POINTS TO 1793 INTERFACE REGISTERS NMI FLAG = \$FF: ENABLE NMI VECTOR
3280	D84C 1A 5Ø	ORCC #\$5Ø	DISABLE FIRQ, IRQ
3281 3282	D84E F7 FF 48 D851 1E 88	STB FDCREG EXG A,A	* SEND READ/WRITE COMMAND TO 1793: SINGLE RECORD, COMPARE * FOR SIDE Ø, NO 15 MS DELAY, DISABLE SIDE SELECT
3283	D853 1E 88	EXG A,A	* COMPARE, WRITE DATA ADDRESS MARK (FB) - WAIT FOR STATUS
3284 3285	D855 C1 80 D857 27 1C	CMPB #\$8Ø BEQ LD875	WAS THIS A READ? IF SO, GO LOOK FOR DATA
3286	0037 27 10	* WAIT FOR THE 1793 TO ACKNOWLEDGE READY T	
3287	D859 C6 Ø2	LDB #\$Ø2	DRQ MASK BIT
3288 3289	D85B E5 C4 D85D 26 ØC	LD85B BITB ,U BNE LD86B	IS 1793 READY FOR A BYTE? (DRQ SET IN STATUS BYTE) BRANCH IF SO
3290	D85F 31 3F	LEAY -1,Y	DECREMENT WAIT TIMER
3291 3292	D861 26 F8 D863 7F Ø9 82	BNE LD85B LD863 CLR NMIFLG	KEEP WAITING FOR THE 1793 DRQ RESET NMI FLAG
3293	D866 1C AF	ANDCC #\$AF	ENABLE FIRQ, IRQ
3294 3295	D868 7E D7 DF	JMP LD7DF	FORCE INTERRUPT, SET DRIVE NOT READY ERROR
3295		* WRITE A SECTOR	
3297	D86B E6 8Ø	LD86B LDB ,X+	GET A BYTE FROM RAM
3298 3299	D86D F7 FF 4B D870 B7 FF 40	STB FDCREG+3 STA DSKREG	SEND IT TO 1793 DATA REGISTER REPROGRAM FDC CONTROL REGISTER
3300	D873 20 F6	BRA LD86B	SEND MORE DATA
33Ø1 33Ø2	D875 C6 Ø2	* WAIT FOR THE 17933 TO ACKNOWLEDGE READY LD875 LDB #\$02	TO READ DATA DRQ MASK BIT
33Ø3	D877 E5 C4	LD877 BITB ,U	DOES THE 1793 HAVE A BYTE? (DRQ SET IN STATUS BYTE)
33Ø4 33Ø5	D879 26 Ø6 D87B 31 3F	BNE LD881 LEAY -1.Y	YES, GO READ A SECTOR DECREMENT WAIT TIMER
3306	D87D 26 F8	BNE LD877	KEEP WAITING FOR 1793 DRQ
33Ø7 33Ø8	D87F 2Ø E2	BRA LD863	GENERATE DRIVE NOT READY ERROR
33Ø9		* READ A SECTOR	
331Ø 3311	D881 F6 FF 4B D884 E7 80	LD881 LDB FDCREG+3 STB ,X+	GET DATA BYTE FROM 1793 DATA REGISTER PUT IT IN RAM
3312	D886 B7 FF 4Ø	STA DSKREG	REPROGRAM FDC CONTROL REGISTER
3313 3314	D889 20 F6	BRA LD881 * BRANCH HERE ON COMPLETION OF SECTOR READ	KEEP GETTING DATA
3315	D88B 1C AF	LD88B ANDCC #\$AF	ENABLE IRQ, FIRO
3316 3317	D88D B6 FF 48 D89Ø 84 7C	LDA FDCREG ANDA #\$7C	* GET STATUS & KEEP WRITE PROTECT, RECORD TYPE/WRITE * FAULT, RECORD NOT FOUND, CRC ERROR OR LOST DATA
3318	D892 97 FØ	STA DCSTA	SAVE IN DSKCON STATUS
3319 3320	D894 39	RTS	
3321		* DSKCON OPERATION CODE JUMP VECTORS	
3322 3323	D895 D7 B8 D897 D7 DØ	LD895 FDB LD7B8 FDB LD7DØ	RESTORE HEAD TO TRACK ZERO NO OP – RETURN
3324	D899 D7 F8	FDB LD7F8	READ SECTOR
3325 3326	D89B D7 FB	FDB LD7FB	WRITE SECTOR
3327		* DSKREG MASKS FOR DISK DRIVE SELECT	
3328 3329	D89D Ø1	LD89D FCB 1 FCB 2	DRIVE SEL Ø
3329 333Ø	D89E Ø2 D89F Ø4	FCB 2 FCB 4	DRIVE SEL 1 DRIVE SEL 2
3331 3332	D8AØ 4Ø	FCB \$4Ø	DRIVE SEL 3
3332		* NMI SERVICE	
3334	D8A1 B6 Ø9 82	DNMISV LDA NMIFLG	GET NMI FLAG
3335 3336	D8A4 27 Ø8 D8A6 BE Ø9 83	BEQ LD8AE LDX DNMIVC	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR
3337	D8A9 AF 6A	STX 10,S	STORE AT STACKED PC SLOT ON STACK
3338 3339	D8AB 7F Ø9 82 D8AE 3B	CLR NMIFLG RTI	RESET NMI FLAG
3340	***= ==		
3341 3342	D8AF B6 FF Ø3	* IRQ SERVICE DIRQSV LDA PIAØ+3	63.5 MICRO SECOND OR 60 HZ INTERRUPT?
3343	D8B2 2A FA	BPL LD8AE	RETURN IF 63.5 MICROSECOND
3344 3345	D8B4 B6 FF Ø2 D8B7 B6 Ø9 85	LDA PIAØ+2 LDA RDYTMR	RESET 60 HZ PIA INTERRUPT FLAG GET TIMER
3346	D8BA 27 11	BEQ LD8CD	BRANCH IF NOT ACTIVE
3347 3348	D8BC 4A D8BD B7 Ø9 85	DECA STA RDYTMR	DECREMENT THE TIMER SAVE IT
3349	D8CØ 26 ØB	BNE LD8CD	BRANCH IF NOT TIME TO TURN OFF DISK MOTORS
3350	D8C2 B6 Ø9 86	LDA DRGRAM	= GET DSKREG IMAGE
3351 3352	D8C5 84 BØ D8C7 B7 Ø9 86	ANDA #\$BØ STA DRGRAM	<pre>= TURN ALL MOTORS AND DRIVE SELECTS OFF = PUT IT BACK IN RAM IMAGE</pre>
3353	D8CA B7 FF 40	STA DSKREG	SEND TO CONTROL REGISTER (MOTORS OFF)
3354 3355	D8CD 7E 89 55	LD8CD JMP L8955	JUMP TO EXTENDED BASIC'S IRQ HANDLER
3356 3357		* THIS IS THE END OF DISK BASIC (EXCEPT FO * THE CODE FROM THIS POINT TO \$DF00 IS GAR	
3358		* DOSBAS 1.1 = 1686 WASTED BYTES	
3359 336Ø		** THIS IS THE CODE FOR THE DOS COMMAND	

0064					
3361					
3362			ORG	DOSBAS+\$1FØØ	
3363	DFØØ 11 3F	DOSCOM	SWI3		DO A SOFTWARE INTERRUPT (#3)
3364	DFØ2 ØF Ø3		CLR	TMPLOC	RESET SECTOR COUNTER
3365	DFØ4 CC 26 ØØ		LDD	#DOSBUF	RAM LOAD ADDRESS FOR SECTOR DATA
3366	DFØ7 34 Ø6		PSHS	B,A	SAVE RAM LOAD ADDRESS
3367	DFØ9 BE CØ Ø6	LDFØ9	LDX	DSKVAR	POINT X TO DSKCON VARIABLES
3368	DFØC ØC Ø3		INC	TMPLOC	INCREMENT SECTOR COUNTER
3369	DFØE 96 Ø3		LDA	TMPLOC	GET THE SECTOR COUNTER
3370	DF10 81 12		CMPA	#SECMAX	LOADED IN 18 SECTORS? (ONE TRACK)
3371	DF12 22 22		BHT	LDE36	YES - EXIT
3372	DF14 A7 Ø3		STA	\$03 X	NO - SAVE SECTOR NUMBER IN DSEC
3373	DF16 CC Ø2 ØØ		חחו	#\$A2AA	GET FDC OP CODE (READ) AND DRIVE NUMBER (Ø)
3374	DF19 A7 84		STA	γ	SAVE THEM IN DSKCON VARIABLES (BUG - SHOULD BE STD ,X)
3375	DF1B 86 22		217	,^ #2.4	GET TRACK NUMBER (34)
3376	DF1D A7 Ø2		CTA	#34 ##2 V	SAVE IT IN DSKCON VARIABLES TOO
3377	DF1F 35 Ø6		SIA	⊅02, Λ Λ D	GET RAM LOAD ADDRESS
			CTD	TMPLOC #DOSBUF B,A DSKVAR TMPLOC TMPLOC #SECMAX LDF36 \$03,X #\$0200 ,X #34 \$02,X A,B \$04,X #\$01 B,A [DCNVEC] \$06,X LDF99 A,B #2*20 LAC46 A,B DOSBUF+1 DOSBUF+1 BAMMST	MEI KAM LUAU AUUKESS
3378	DF21 ED Ø4		210	\$04,X	AND SAVE IT IN THE DSKCON VARIABLES
3379	DF23 8B Ø1		AUUA	#\$01	ADD 256 (ONE SECTOR) TO RAM LOAD ADDRESS (SHOULD BE INCA)
3380	DF25 34 Ø6		PSH2	В, А	SAVE NEW RAM LOAD ADDRESS
3381	DF27 AD 9F CØ Ø4		JSR	[DCNVEC]	GO READ A SECTOR
3382	DF2B 6D Ø6		TST	\$Ø6,X	CHECK FOR ERRORS
3383	DF2D 27 DA		BEQ	LDFØ9	KEEP READING IF NONE
3384	DF2F 35 Ø6	LDF36	PULS	A,B	PULL LOAD ADDRESS OFF OF THE STACK
3385	DF31 C6 28		LDB	#2*20	'IO' ERROR
3386	DF33 7E AC 46		JMP	LAC46	JUMP TO ERROR SERVICING ROUTINE
3387	DF36 35 Ø6		PULS	A,B	PULL RAM LOAD ADDRESS OFF OF THE STACK
3388	DF38 FC 26 ØØ		LDD	DOSBUF	GET FIRST TWO BYTES OF RAM DATA
3389	DF3B 1Ø 83 4F 53		CMPD	#'0S'	LOOK FOR 'OS' (OS9) AT START OF BUFFER
3390	DF3F 10 27 46 BF		LBE0	DOSBUF+2	IF 'OS' THEN BRANCH TO DATA LOADED IN RAM
3391	DF43 7F 26 ØØ		CLR	DOSBUF	* OTHERWISE CLEAR THE FIRST TWO
3392					
3393	DF46 7F 26 Ø1	DOSINI	CLR	DOSBUF+1	* BYTES OF RAM DATA
3394	DF49 7E AØ E8	5001111	JMP	BAWMST	JUMP TO BASIC'S WARM START
3395	DI 45 7E NO EG		0111	DAMI 13 1	OUT TO BASIC S WART START
3396	DF4C CC 3B 3B		LDD	#\$3B3B	TWO RTI INSTRUCTIONS
3397	DF4F FD Ø1 ØØ			SW3VEC	*
3398					
	DF52 FD Ø1 Ø2			SW3VEC+2	* LOAD THE SWI2 AND SWI3 JUMP
3399	DF55 FD Ø1 Ø4			SW2VEC+1	* VECTORS WITH RTIS
3400	DF58 39	END	RTS		
3401					
3402				DOS AND DOSINI COMMANDS - THE REST	
34Ø3				OF THE DISK ROM (\$DFFF) IS GARBAGE	ŧ.
34Ø4		* DOSBAS	1.1	= 167 WASTED BYTES	
34Ø5					

<sup>\*</sup> DOSBAS 1.1 = 167 WASTED BYTES

```
aaa1
                 ØØ E1
                                           DHITOK
                                                      EOU
                                                                                                              HIGHEST 1.0 DISK TOKEN
                 00 32
0002
                                           CYEAR
                                                       EQU
аааз
0004
aaas
0006
                                           **** FILE ALLOCATION TABLE FORMAT
aaaz
0008
ØØØ9
                                          * THE FILE ALLOCATION TABLE (FAT) CONTAINS THE STATUS OF THE GRANULES ON A DISKETTE.
* THE FAT CONTAINS 6 CONTROL BYTES FOLLOWED BY 68 DATA BYTES (ONE PER GRANULE). ONLY THE
aala
ØØ11
                                           * FIRST TWO OF THE SIX CONTROL BYTES ARE USED. A VALUE OF $FF IS SAVED IN UNALLOCATED * GRANULES. IF BITS 6 & 7 OF THE DATA BYTE ARE SET, THE GRANULE IS THE LAST GRANULE
ØØ12
0013
0014
                                           * IN A FILE AND BITS 0-5 ARE THE NUMBER OF USED SECTORS IN THAT GRANULE. IF BITS 6 & 7 * ARE NOT SET, THE DATA BYTE CONTAINS THE NUMBER OF THE NEXT GRANULE IN THE FILE.
0015
ØØ16
                                           * OFFSETS TO FAT CONTROL BYTES
0017
                                                                                                              ACTIVE FILE COUNTER : DISK TO RAM FAT IMAGE DISABLE VALID DATA FLAG: Ø=DISK DATA VALID, <\!\!> Ø = NEW FAT DATA - DISK DATA INVALID
                 ØØ ØØ
ØØ18
0019
                 00 01
                                           FAT1
                                                      EOU
0020
0021
                                                               2 TO 5
                                                                                                               NOT USED
                 ØØ Ø6
                                           FATCON
                                                      EQU
                                                                                                              OFFSET TO START OF FAT DATA (68 BYTES)
0022
0023
ØØ24
0025
                                           **** DIRECTORY ENTRY FORMAT
0026
ØØ27
                                           * THE DIRECTORY IS USED TO KEEP TRACK OF HOW MANY FILES ARE STORED ON A DISKETTE
* AND WHERE THE FILE IS STORED ON THE DISK. THE FIRST GRANULE USED BY THE FILE WILL
* ALLOW THE FAT TO TRACK DOWN ALL OF THE GRANULES USED BY THE FILE. IF THE FIRST
0028
ØØ29
0030
ØØ31
                                           * BYTE OF THE DIRECTORY ENTRY IS ZERO, THE FILE HAS BEEN KILLED;
0032
                                           * IF THE FIRST BYTE IS $FF THEN THE DIRECTORY ENTRY HAS NEVER BEEN USED.
ØØ33
                                                                                                              DESCRIPTION
ØØ34
                                                               RYTE
ØØ35
                                                                                                              FILE NAME
FILE EXTENSION
0036
                 00 00
                                           DIRNAM
                                                      EOU
                                                               Ø
                 ØØ Ø8
                                           DIREXT
0037
                                                       EOU
ØØ38
                 ØØ ØB
                                           DIRTYP
                                                               11
                                                                                                               FILE TYPE
                                                       EQU
0039
                 ØØ ØC
                                           DIRASC
                                                       EOU
                                                               12
                                                                                                               ASCII FLAG
0040
                                                                                                               FIRST GRANULE IN FILE
                                           DIRGRN
                                                                                                               NUMBER OF BYTES IN LAST SECTOR
0041
                 ØØ ØE
                                           DIRLST
                                                      EOU
                                                               14
0042
                                                               16 TO 31
0043
                                           **** FILE CONTROL BLOCK FORMAT
0045
0046
αα47
                                             THE FILE STRUCTURE OF COLOR TRS DOS IS CONTROLLED BY A FILE CONTROL BLOCK (FCB)
0048
                                          * THE FILE SINCUINE DE COURT HS DUS IS CONTROLLED BY A FILE CONTROL BLOCK FOR THE FOR CONTROL BYTES CONTROL BYTES AND A SECTOR LONG (256 BYTES) DATA BUFFER.

* THE CONTROL BYTES CONTROL THE ORDERLY FLOW OF DATA FROM THE COMPUTER'S RAM TO

* THE DISKETTE AND VICE VERSA. THE OPEN COMMAND INITIALIZES THE FCB; THE INPUT,

* OUTPUT, WRITE, PRINT, GET AND PUT COMMANDS TRANSFER DATA THROUGH THE FCB AND

* THE CLOSE COMMAND TURNS OFF THE FCB.
ØØ49
0050
0051
0052
ØØ53
0054
                                           * TABLES OF OFFSETS TO FCB CONTROL BYTES
0056
ØØ57
                                           ***** RANDOM FILE
0058
                                                               BYTE
                                                                                                              DESCRIPTION
                                           FCBTYP
                                                                                                               FILE TYPE: $40=RANDOM/DIRECT, Ø=CLOSED
0059
                 00 00
                                                       EOU
9969
                 00 01
                                           FCBDRV
                                                       EQU
                                                                                                               DRIVE NUMBER
                                                                                                               FIRST GRANULE IN FILE
                 00 02
                                           FCBFGR
0061
                                                       EOU
                                                                                                               CURRENT GRANULE BEING USED
ØØ62
0063
                 00 04
                                           FCBSEC
                                                      EOU
                                                                                                               CURRENT SECTOR BEING USED (1-9)
                                           ECRPOS
                                                                                                              CURRENT PRINT POSITION - ALWAYS ZERO IN RANDOM FILES
9965
                 98 96
                                                       FOII
                                                                                                               CURRENT RECORD NUMBER
ØØ66
                 ØØ Ø7
                                           FCBREC
                                                                                                               RANDOM FILE RECORD LENGTH
0067
                 ØØ Ø9
                                           FCBRLN
                                                       E0U
                                                                                                              PRINTER TO START OF THIS FILE'S RANDOM ACCESS BUFFER SECTOR OFFSET TO CURRENT POSITION IN RECORD GET/PUT FLAG: Ø=PUT, 1=PUT
                                           FCBBUF
ØØ68
                 ØØ ØB
                                                       EQU
                                                               11
aasa
                 aa ab
                                           FCRSOF
                                                       EOU
                                                               13
                 ØØ ØF
                                           FCBFLG
0070
                                                       EQU
                                                               15
                                                                                                              OF THE CURRENT RECORD OF THE CURRENT RECORD
                                                               16,17
0071
                                           FCBDIR
                 00 12
                                                      EQU
0072
                                                               18
0073
                                           FCBLST
0074
                 00 15
                                           FCBGET
                                                      EOU
                                                               21
ØØ75
                                                                                                               'PUT' RECORD COUNTER: POINTER TO WHERE IN THE RECORD THE NEXT
0076
                 ØØ 17
                                           FCBPUT
                                                      EQU
                                                              23
                                                                                                              BYTE WILL BE 'PUT'
OFFSET TO START OF FCB DATA BUFFER (256 BYTES)
0077
0078
                 ิติต 19
                                           FCBCON
                                                      EQU
                                                              25
ØØ79
ดดลด
                                           **** SEQUENTIAL FILE
                                                                                                              DESCRIPTION
0081
                                                               BYTE
0082
                 аа аа
                                           FCRTYP
                                                       FOII
                                                                                                              FILE TYPE: $10=INPUT, $20=OUTPUT, 0=CLOSED DRIVE NUMBER
                 00 01
                                           FCBDRV
0083
                                                       EOU
ØØ84
                                           FCBFGR
                                                                                                               FIRST GRANULE IN FILE
                                                                                                              CURRENT GRANULE BEING USED
0085
                 00 03
                                           FCBCGR
                                                       EOU
                                                               3
                                                                                                               CURRENT SECTOR BEING USED (1-9)
ØØ86
                                           FCBSEC
                                                       EQU
                                                                                                               INPUT FILE: CHARACTER POINTER - POINTS TO NEXT CHARACTER IN
ØØ87
                                           ECRCPT
                                                       FOII
                                                               5
                                                                                                               FILE TO BE PROCESSED.
ØØ88
                                                                                                              OUTPUT FILE: FULL SECTOR FLAG - IF IT IS 1 WHEN THE FILE IS CLOSED IT MEANS 256 BYTES OF THE LAST SECTOR HAVE BEEN USED.
ดดลจ
0090
0091
                 ØØ Ø6
ØØ Ø7
                                           FCBPOS
FCBREC
                                                      FOII
                                                                                                              CURRENT PRINT POSITION
CURRENT RECORD NUMBER: HOW MANY WHOLE SECTORS HAVE BEEN
0092
                                                       EQU
ØØ93
                                                                                                               INPUT OR OUTPUT TO A FILE.
                                                                                                              UNUSED
                                                               9 TO 15
0094
                                           FCBRLN
                                                                                                              CACHE FLAG: ØØ=CACHE EMPTY, $FF=CACHE FULL CACHE DATA BYTE
ØØ95
                 00 10
                                                       EQU
0096
                 ØØ 11
                                           FCBCDT
                                                      EOU
                                                              17
```

0097	00 12	FCBDIR	EQU	18	DIRECTORY ENTRY NUMBER (0-71)
0098	00 13	FCBLST *	EQU	19	NUMBER OF BYTES IN LAST SECTOR OF FILE
ØØ99 Ø1ØØ	ØØ 17	FCBDFL	EQU	21,22 23	UNUSED INPUT FILE ONLY: DATA LEFT FLAG: Ø=DATA LEFT, \$FF=NO DATA (EMPTY)
0101	00 18	FCBLFT	EQU	24	NUMBER OF CHARACTERS LEFT IN BUFFER (INPUT FILE)
0102	gg 10	*	F011	05	NUMBER OF CHARS STORED IN BUFFER (OUTPUT FILE)
Ø1Ø3 Ø1Ø4	ØØ 19	FCBCON	EQU	25	OFFSET TO FCB DATA BUFFER (256 BYTES)
0105			ORG	\$CØØØ	
0106					
Ø1Ø7 Ø1Ø8	CØØØ 44 4B CØØ2 2Ø Ø4	DOSBAS LCØØ2	FCC BRA	'DK' LC008	
0109	0002 20 01	20002	5101	20000	
0110	CØØ4 D6 6C	DCNVEC	FDB	DSKCON	DSKCON POINTER
Ø111 Ø112	CØØ6 ØØ EA	DSKVAR	FDB	\$00EA	ADDRESS OF DSKCON VARIABLES
Ø113		**** ZEF	RO OUT	THE RAM USED BY DISK BASIC	
0114	CØØ8 8E Ø6 ØØ	LCØØ8	LDX	#DBUFØ	POINT X TO START OF DISK RAM
Ø115 Ø116	CØØB 6F 8Ø CØØD 8C Ø9 89	LCØØB	CLR	,X+ #DFLBUF	CLEAR A BYTE END OF DISK'S RAM?
Ø117	CØ1Ø 26 F9			LCØØB	NO - KEEP CLEARING
0118	CØ12 8E CØ F6			#LCØF6	POINT X TO ROM IMAGE OF COMMAND INTERPRETATION TABLE
Ø119 Ø12Ø	CØ15 CE Ø1 34 CØ18 C6 ØA		LDU	#COMVEC+2Ø	POINT U TO RAM ADDRESS OF SAME 10 BYTES PER TABLE
Ø121	CØ1A BD A5 9A			LA59A	MOVE (B) BYTES FROM (X) TO (U)
0122	CØ1D CC B2 77			#LB277	SYNTAX ERROR ADDRESS
Ø123 Ø124	CØ2Ø ED 43 CØ22 ED 48			\$Ø3,U \$Ø8,U	* SET JUMP TABLE ADDRESSES OF THE USER COMMAND * INTERPRETATION TABLE TO POINT TO SYNTAX ERROR
Ø125	CØ24 6F C4		CLR	,U	CLEAR BYTE Ø OF USER TABLE (DOESN'T EXIST FLAG)
Ø126	CØ26 6F 45			\$Ø5,U	SET NUMBER OF SECONDARY USER TOKENS TO ZERO
Ø127 Ø128	CØ28 CC CE 2E CØ2B FD Ø1 2D			#DXCVEC COMVEC+13	* SAVE NEW * POINTERS TO EXBAS
Ø129	CØ2E CC CE 56			#DXIVEC	* COMMAND AND SECONDARY
0130	CØ31 FD Ø1 32			COMVEC+18	* COMMAND INTERPRETATION ROUTINES
Ø131 Ø132	CØ34 CE Ø1 5E	^^^^ MU\	LDU	NEW RAM VECTORS FROM ROM TO RAM #RVECØ	POINT U TO 1ST RAM VECTOR
Ø133	CØ37 86 7E	LCØ37		#\$7E	OP CODE OF JMP INSTRUCTION
Ø134	CØ39 B7 Ø1 AØ		STA	RVEC22	SET 1ST BYTE OF 'GET'/'PUT' RAM VECTOR TO 'JMP'
Ø135 Ø136	CØ3C A7 CØ CØ3E EC 81		STA LDD	,U+ ,X++	SET 1ST BYTE OF RAM VECTOR TO 'JMP' GET RAM VECTOR FROM ROM
Ø137	CØ4Ø ED C1		STD	,U++	STORE IT IN RAM
Ø138	CØ42 8C C1 26			#LC126	COMPARE TO END OF ROM VALUES
Ø139 Ø14Ø	CØ45 26 FØ CØ47 8E C2 9A			LCØ37 #DVEC22	BRANCH IF NOT ALL VECTORS MOVED GET ROM VALUE OF 'GET'/'PUT' RAM VECTOR
0141	CØ4A BF Ø1 A1		STX	RVEC22+1	SAVE IT IN RAM
Ø142 Ø143	COAD OF OO EF	**** IN]		ZE DISK BASIC'S USR VECTORS #DUSRVC	DOINT V TO CTART OF RICK RACIC HCR VECTORS
Ø143	CØ4D 8E Ø9 5F CØ5Ø 9F BØ		STX	USRADR	POINT X TO START OF DISK BASIC USR VECTORS SAVE START ADDRESS IN USRADR
Ø145	CØ52 CE B4 4A			#LB44A	POINT U TO ADDRESS OF 'FUNCTION CALL' ERROR
Ø146 Ø147	CØ55 C6 ØA CØ57 EF 81	LCØ57	LDB STU	#10 V++	10 USER VECTORS TO INITIALIZE SET USR VECTOR TO 'FC' ERROR
Ø148	CØ59 5A	LC#37	DECB	,X++	DECREMENT USR VECTOR COUNTER
0149	CØ5A 26 FB			LCØ57	BRANCH IN NOT DONE WITH ALL 10 VECTORS
Ø15Ø Ø151	CØ5C 8E D7 AE CØ5F BF Ø1 ØA			#DNMISV NMIVEC+1	GET ADDRESS OF NMI SERVICING ROUTINE SAVE IT IN NMI VECTOR
Ø152	CØ62 86 7E			#\$7E	OP CODE OF JMP
Ø153	CØ64 B7 Ø1 Ø9			NMIVEC	MAKE THE NMI VECTOR A JMP
Ø154 Ø155	CØ67 8E D7 BC CØ6A BF Ø1 ØD			#DIRQSV IRQVEC+1	GET ADDRESS OF DISK BASIC IRQ SERVICING ROUTINE SAVE IT IN IROVEC
Ø156	CØ6D 86 13		LDA		= INITIALIZE WRITE FAT
0157	CØ6F B7 Ø9 7A			WFATVL	= TO DISK TRIGGER VALUE
Ø158 Ø159	CØ72 7F Ø8 ØØ CØ75 7F Ø8 4A			FATBLØ FATBL1	* INITIALIZE THE ACTIVE FILE COUNTER OF
0160	CØ78 7F Ø8 94		CLR	FATBL2	* EACH FAT TO ZERO. THIS WILL CAUSE THE FATS
Ø161	CØ7B 7F Ø8 DE			FATBL3	* TO THINK THERE ARE NO ACTIVE FILES
Ø162 Ø163	CØ7E 8E Ø9 89 CØ81 BF Ø9 48			#DFLBUF RNBFAD	= GET THE STARTING ADDRESS OF THE = RANDOM FILE BUFFER FREE AREA AND DAVE IT AS THE
Ø164	CØ84 3Ø 89 Ø1 ØØ		LEAX	256,X	= START ADDRESS OF FREE RAM FOR RANDOM FILE BUFFERS
Ø165 Ø166	CØ88 BF Ø9 4A	*	STX	FCBADR	SAVE 256 BYTES FOR RANDOM FILE BUFFERS INITIALLY SAVE START ADDRESS OF FCBS
Ø167	CØ8B 3Ø Ø1		LEAX	\$Ø1,X	* ADD ONE AND SAVE THE STARTING
Ø168	CØ8D BF Ø9 28		STX	FCBV1	* ADDRESS OF FCB1
Ø169 Ø17Ø	CØ9Ø 6F ØØ CØ92 3Ø 89 Ø1 19			FCBTYP,X FCBLEN,X	CLEAR THE FIRST BYTE OF FCB 1 (CLOSE FCB) POINT X TO FCB 2
Ø170 Ø171	CØ96 BF Ø9 2A			FCBV1+2	SAVE ITS STARTING ADDRESS IN FCB VECTOR TABLE
Ø172	CØ99 6F ØØ			FCBTYP,X	CLEAR THE FIRST BYTE OF FCB 2 (CLOSE FCB)
Ø173 Ø174	CØ9B 3Ø 89 Ø1 19	*	LEAX	FCBLEN,X	* POINT X TO SYSTEM FCB - THIS FCB WILL ONLY * BE USED TO COPY, LOAD, SAVE, MERGE, ETC
Ø174 Ø175	CØ9F BF Ø9 2C		STX	FCBV1+4	SAVE ITS ADDRESS IN THE FCB VECTOR TABLE
0176	CØA2 6F ØØ			FCBTYP,X	CLEAR THE FIRST BYTE OF SYSTEM FCB (CLOSE FCB)
Ø177 Ø178	CØA4 86 Ø2 CØA6 B7 Ø9 5B			#\$Ø2 FCBACT	* SET THE NUMBER OF ACTIVE RESERVED  * FILE BUFFERS TO 2 (1,2)
Ø178	CØA9 3Ø 89 Ø1 19			FCBLEN, X	POINT X TO ONE PAST THE END OF SYSTEM FCB
0180	CØAD 1F 1Ø		TFR	X,D	SAVE THE ADDRESS IN ACCD
Ø181 Ø182	CØAF 5D CØBØ 27 Ø1		TSTB BFO	LCØB3	ON AN EVEN 256 BYTE BOUNDARY? YES
Ø183	CØB2 4C		INCA		NO - ADD 256 TO ADDRESS
Ø184	CØB3 1F 89	LCØB3	TFR		COPY ACCA TO ACCB
Ø185 Ø186	CØB5 CB 18 CØB7 D7 19		ADDB STB	#24 TXTTAB	SAVE ENOUGH ROOM FOR 4 GRAPHICS PAGES (PCLEAR 4) SAVE NEW START OF BASIC ADDRESS
Ø187	CØB9 BD 96 EC		JSR	L96EC	INITIALIZE EXBAS VARIABLES & DO A NEW
Ø188	CØBC 96 BA			BEGGRP	GET THE START OF CURRENT GRAPHICS PAGE
Ø189 Ø19Ø	CØBE 8B Ø6 CØCØ 97 B7			#\$Ø6 ENDGRP	ADD 1.5K (6 X 256 = ONE GRAPHICS PAGE) SAVE NEW END OF GRAPHICS PAGE
Ø191	CØC2 8D 19		BSR	LCØDD	GO INITIALIZE THE FLOPPY DISK CONTROLLER
Ø192	CØC4 1C AF		ANDC	C #\$AF	TURN ON IRQ AND FIRQ

Ø196	CØC6 8E C1 25 CØC9 BD B9 9C CØCC 8E CØ D4 CØCF 9F 72 CØD1 7E AØ E2	JSR LDX	#LC125 STRINOUT #DKWMST RSTVEC LAØE2	POINT X TO DISK BASIC COPYRIGHT MESSAGE PRINT COPYRIGHT MESSAGE TO SCREEN GET DISK BASIC WARM START ADDRESS SAVE IT IN RESET VECTOR JUMP BACK TO BASIC
0199 0200 0201 0202 0203 0204 0205 0206 0207 0208 0209 0210 0211	CØD5 8D 06 CØD7 BD D1 E5 CØDA 7E 80 CØ CØDD 7F 09 82 CØEØ 7F 09 85 CØE3 7F 09 86	JSR JMP LCØDD CLR CLR CLR	RDYTMR DRGRAM	WARM START INDICATOR INITIALIZE THE FLOPPY DISK CONTROLLER CLOSE FILES AND DO MORE INITIALIZATION JUMP TO EXBAS' WARM START RESET MIN FLAG RESET DRIVE NOT READY TIMER RESET RAM IMAGE OF DSKREG (MOTORS OFF) RESET DISK CONTROL REGISTER FORCE INTERRUPT COMMAND OF 1793 SEND IT TO 1793 * DELAY * DELAY SOME MORE GET 1793 STATUS (CLEAR REGISTER)
0214 0215 0216 0217 0218 0219		LCØF6 FCB FDB FCB FCB		19 DISK BASIC 1.0 COMMANDS DISK BASIC'S COMMAND DICTIONARY COMMAND JUMP TABLE 6 DISK BASIC SECONDARY FUNCTIONS SECONDARY FUNCTION TABLE SECONDARY FUNCTION JUMP TABLE
0224 0225 0226 0227 0228 0229	C100 C4 26 C8 38 C8 43 C106 CB 4A C5 8F C8 18 C10C C8 1B CA 3B CA 4B C112 8E 90 CC 5B C8 59 C118 C6 B7 CA 36 C8 60 C11E CD F6 C6 B7 C2 4D C124 C9 90	IC100 EDD	FOR DISK BASIC DVECØ,DVEC1,DVEC2 DVECØ,DVEC4,DVEC5 DVECØ,DVEC7,DVEC8 XVEC9,DVEC10,DVEC11 DVEC12,DVEC13,DVEC14 DVEC15,DVEC12,DVEC17 DVEC18	
Ø233 Ø234 Ø235	C126 44 49 53 4B 20 45 C12C 58 54 45 4E 44 45 C132 44 20 43 4F 4C 4F C138 52 20 42 41 53 49		COPYRIGHT MESSAGE 'DISK EXTENDED COLOR BASIC 1.0'	
Ø237 Ø238 Ø239	C13E 43 2Ø 31 2E 3Ø C143 ØD C144 43 4F 5Ø 59 52 49 C14A 47 48 54 2Ø 28 43 C15Ø 29 2Ø 31 39 38	FCB FDB		
Ø242 Ø243	C155 32 C156 20 42 59 20 54 41 C15C 4E 44 59	FCC	CYEAR ' BY TANDY'	
0245 0246 0247 0248 0249	C15F ØD C16Ø 55 4E 44 45 52 2Ø C166 4C 49 43 45 4E 53 C16C 45 2Ø 46 52 4F 4D C172 2Ø 4D 49 43 52 4F C178 53 4F 46 54		CR 'UNDER LICENSE FROM MICROSOFT'	
0251	C17C ØD ØD ØØ		CR,CR,Ø	
Ø252 Ø253		*	COMMAND DICTIONARY TABLE	TOKEN #
0255 0256 0257 0258 0259 0260 0261 0262 0263 0264 0265 0266 0266 0267 0268 0269 0270 0271	C182 44 52 49 56 C5 C187 46 49 45 4C C4 C18C 46 49 4C 45 D3 C191 4B 49 4C CC C195 4C 4F 41 C4 C199 4C 53 45 D4 C190 4D 45 52 47 C5 C1A2 52 45 4E 41 4D C5 C1A8 52 53 45 D4 C1AC 53 41 56 C5 C1B0 57 52 49 54 C5 C1B0 57 52 49 54 C5 C1B5 56 45 52 49 46 D9 C1B8 55 4E 4C 4F 41 C4 C1C1 44 53 4B 49 4E C9 C1C7 42 41 43 4B 55 D9 C1C7 42 41 43 4B 55 D9 C1D1 44 53 4B 49 A4 C1D6 44 53 4B 49 A4	FCC FCC FCC FCC FCC FCC FCC FCC FCC FCC	'DI', \$80+'R' 'DRIV', \$80+'E' 'FIEL', \$80+'D' 'FILE', \$80+'C' 'LOA', \$80+'C' 'LOA', \$80+'T' 'LSE', \$\$0+'T' 'MERG', \$80+'E' 'RENAM', \$80+'E' 'RENAM', \$80+'E' 'RSE', \$80+'E' 'WRIT', \$80+'E' 'WRIT', \$80+'E' 'UNLOA', \$80+'D' 'DSKIN', \$80+'T' 'BACKU', \$80+'T' 'SACKU', \$80+'Y' 'DSKO', \$80+'Y' 'DSKO', \$80+'Y'	CE CF DØ D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DB DC DD DB DC DD DE
Ø274 Ø275		*	COMMAND JUMP TABLE	COMMAND / TOKEN #
0276 0277 0278 0279 0280 0281 0282 0283 0284 0285 0286 0287	C1DB CC A9 C1DD CE C5 C1DF DØ BC C1E1 D1 5C C1E3 C6 EF C1E5 CA 48 C1E7 D1 Ø2 C1E9 CA 39 C1EB DØ 1B C1ED D1 Ø1 C1EF C9 EØ C1F1 DØ 66 C1F3 D7 4E		DIR DRIVE FIELD FILES KILL LOAD LSET MERGE RENAME RSET SAVE WRITE VERIFY	DIR / CE DRIVE / CF FIELD / DØ FILES / DI KILL / D2 LOAD / D3 LSET / D4 MERGE / D5 RENAME / D6 RSET / D7 SAVE / D8 WRITE / D9 VERIFY / DA

Ø292 Ø293 Ø294 Ø295	C1F5 D2 33 C1F7 D5 99 C1F9 D2 62 C1FB D3 B9 C1FD D4 ED C1FF D5 62	FDB UNLOAD FDB DSKINI FDB BACKUP FDB COPY FDB DSKI FDB DSKO	UNLOAD / DB DSKINI /DC BACKUP / DD COPY / DE DSKI\$ / DF DSKO\$ / EØ
0299 0300 0301 0302	C201 43 56 CE C204 46 52 45 C5 C208 4C 4F C3 C208 4C 4F C6 C20E 4D 4B 4E A4 C212 41 D3	* SECONDARY FUNCTION DICTIONARY TABLE  * LC201	TOKEN # A2 A3 A4 A5 A6 A7
0306 0307 0308 0309 0310 0311	C214 CD F4 C216 CE 9C C218 CE 10 C21A CE 37 C21C CE 02 C21E B2 77	* LC214 FDB CVN FDB FREE FDB LOC FDB LOF FDB MKN\$	FUNCTION / TOKEN # CVN / A2 FREE / A3 LOC / A4 LOF / A5 MKN\$ / A6 AS / A7
Ø314 Ø315 Ø316 Ø317 Ø318 Ø319 Ø320	C220 81 E0 C222 22 08 C224 8E C1 DB C227 80 CE C229 7E AD D4 C22C 81 E0 C22E 10 23 F0 45 C232 6E 9F 01 41	LC22Ø CMPA #DHITOK BHI LC22C LDX #LC1DB SUBA #\$CE JMP LADD4 LC22C CMPA #DHITOK LBLS LB277 JMP [COMVEC+33]	*COMPARE TO HIGHEST DISK BASIC TOKEN *AND BRANCH IF HIGHER POINT X TO DISK BASIC COMMAND JUMP TABLE SUBTRACT OUT LOWEST DISK BASIC COMMAND TOKEN JUMP TO BASIC'S COMMAND HANDLER COMPARE TO HIGHEST DISK BASIC TOKEN 'SYNTAX' ERROR IF < DISK BASIC COMMAND TOKEN PROCESS A USER COMMAND TOKEN
0322 0323 0324 0325 0326 0327 0328 0329 0330 0331	C236 C1 4E C238 23 04 C23A 6E 9F 01 46 C23E C0 44 C240 34 04 C242 BD B2 62 C245 35 04 C247 8E C2 14 C24A 7E B2 CE	*DISK BASIC SECONDARY COMMAND INTERPRETATION F LC236 CMPB %(\$A7-\$80)*2 BLS LC23E JMP [COMVEC-38] LC23E SUBB %(\$A2-\$80)*2 PSHS B JSR LB262 PULS B LDX #LC214 JMP LB2CE	**COMPARE TO HIGHEST DISK BASIC TOKEN  *AND BRANCH IF HIGHER POINT X TO DISK BASIC COMMAND JUMP TABLE SUBTRACT OUT LOWEST DISK BASIC COMMAND TOKEN JUMP TO BASIC'S COMMAND HANDLER COMPARE TO HIGHEST DISK BASIC TOKEN 'SYNTAX' ERROR IF < DISK BASIC COMMAND TOKEN PROCESS A USER COMMAND TOKEN HANDLER **COMPARE MODIFIED SECONDARY TOKEN TO *HIGHEST DISK BASIC TOKEN & BRANCH IF HIGHER JUMP TO USER SECONDARY COMMAND HANDLER **SUBTRACT OUT THE SMALLEST SECONDARY **DISK TOKEN & SAVE MODIFIED TOKEN ON THE STACK SYNTAX CHECK FOR '(' AND EVALUATE EXPRESSION RESTORE MODIFIED TOKEN POINT X TO SECONDARY COMMAND JUMP TABLE JUMP TO BASIC'S SECONDARY COMMAND HANDLER  PUT THE RETURN ADDRESS INTO Y RESET THE CONT FLAG. ETC
0336 0337 0338 0339 0340 0341 0342 0343 0344 0345 0346 0347 0348	C24D 35 20 C24F BD AD 33 C252 BD D1 E5 C255 34 24 C257 BD CA 3B C25A 35 04 C25C C1 36 C25E 10 25 C6 8E C262 32 62 C264 BD A7 E9 C267 BD A9 74 C26A 0F 6F C26C BD B9 5C C26F BD B9 AF C272 8E C2 42 C275 7E AC 60	* ERROR DRIVER RAM VECTOR DVEC17 PULS Y  JSR LAD33 JSR LD1E5 PSHS Y,B JSR DVEC7 PULS B  CMPB #2*27 LBCS XVEC17 LEAS \$02,S JSR LA7E9 JSSR LA7E9 JSSR LA974 CLR DEWNUM JSR LB95C JSSR LB9AF LDX #LC278-2*27 JMP LAC60	PUT THE RETURN ADDRESS INTO Y RESET THE CONT FLAG, ETC INITIALIZE SOME DISK VARIABLES AND CLOSE FILES PUT RETURN ADDRESS AND ERROR NUMBER ON THE STACK CLOSE ALL FILES GET THE ERROR NUMBER BACK COMPARE TO THE LOWEST DISK ERROR NUMBER BRANCH TO EXBAS ERROR HANDLER IF NOT DISK ERROR NUMBER PURGE RETURN ADDRESS OFF THE STACK TURN OFF THE CASSETTE MOTOR DISABLE THE ANALOG MULTIPLEXER SET DEVICE NUMBER TO THE SCREEN SEND A CR TO THE SCREEN SEND A '?' TO THE SCREEN POINT X TO DISK BASIC'S ERROR TABLE JUMP TO BASIC'S ERROR HANDLER
0356 0357 0358 0359 0360 0361 0362	C278 42 52 C27A 44 46 C27C 4F 42 C27E 57 50 C280 46 4E C282 46 53 C284 41 45 C286 46 4F C288 53 45 C28A 56 46 C28A 56 46 C28C 45 52	* DISK BASIC ERROR MESSAGES LC290 FCC 'BR' FCC 'DF' FCC 'OB' FCC 'WP' FCC 'FS' FCC 'FC' FCC 'FC' FCC 'SE' FCC 'SE' FCC 'SE' FCC 'ER'	27 BAD RECORD 28 DISK FULL 29 OUT OF BUFFER SPACE 30 WRITE PROTECTED 31 BAD FILE NAME 32 BAD FILE STRUCTURE 33 FILE ALREADY EXISTS 34 FIELD OVERFLOW 35 SET TO NON-FIELDED STRING 36 VERIFICATION ERROR 37 WRITE OR INPUT PAST END OF RECORD
Ø367 Ø368	C28E 42 41 53 C291 20 20 20 C294 44 41 54 C297 42 49 4E	* DISK FILE EXTENSIONS BASEXT FCC 'BAS' DEFEXT FCC 'DAT' BINEXT FCC 'BIN'	BASIC FILE EXTENSION BLANK (DEFAULT) FILE EXTENSION DATA FILE EXTENSION BINARY FILE EXTENSION
Ø373 Ø374 Ø375 Ø376 Ø377	C29A 34 11 C29C AE 63 C29E 8C 97 5F C2A1 26 04 C2A3 81 23 C2A5 27 02 C2A7 35 91	* CLS RAM VECTOR DVEC22 PSHS X,CC LDX \$03,S CMPX #L975F BNE LC2A7 CMPA #'#' BEQ LC2A9 LC2A7 PULS CC,X,PC	SAVE X REG AND STATUS LOAD X WITH CALLING ADDRESS COMING FROM EXBAS' GET/PUT? NO NUMBER SIGN (GET#, PUT#)? BRANCH IF GET OR PUT TO RANDOM FILE RESTORE X REG, STATUS AND RETURN
Ø381 Ø382 Ø383	C2A9 32 65 C2AB BD C7 FE C2AE 9F F1 C2BØ 6F 88 15 C2B3 6F 88 16	* GET/PUT TO A DIRECT/RANDOM FILE LC2A9 LEAS \$05,S JSR LC7FE STX FCBTMP CLR FCBGET,X CLR FCBGET+1,X	PURGE RETURN ADDRESS AND REGISTERS OFF OF THE STACK EVALUATE DEVICE NUMBER & SET FCB POINTER SAVE FCB POINTER * RESET THE GET * DATA POINTER

Ø385 Ø386					
Ø386	C2B6 6F 88 17		CLD	FCBPUT, X	= RESET THE PUT
	C2B9 6F 88 18			FCBPUT+1,X	= DATA POINTER
				FCBPOS,X	
Ø387 Ø388	C2BC 6F Ø6			FCBDRV,X	RESET PRINT POSITION COUNTER
Ø389	C2BE A6 Ø1				*GET THE FCB DRIVE NUMBER AND
	C2CØ 97 EB			DCDRV	*SAVE IT IN DSKCON VARIABLE
0390	C2C2 9D A5		JSR	GETCCH	GET CURRENT INPUT CHARACTER FROM BASIC
Ø391	C2C4 27 ØA			LC2DØ	BRANCH IF END OF LINE
Ø392	C2C6 BD B2 6D			SYNCOMMA	SYNTAX CHECK FOR COMMA
Ø393	C2C9 BD B3 E6		JSR	LB3E6	EVALUATE EXPRESSION - RETURN IN ACCD
Ø394	C2CC 9E F1	LC2CC	LDX	FCBTMP	POINT X TO FCB
Ø395	C2CE ED Ø7		STD	FCBREC, X	SAVE RECORD NUMBER IN FCB
Ø396	C2DØ BD C6 58	LC2DØ		LC658	INCREMENT RECORD NUMBER
Ø397	C2D3 EC Ø9		LDD	FCBRLN,X	* GET RANDOM FILE RECORD LENGTH AND RANDOM FILE
Ø398	C2D5 AE ØB				* BUFFER POINTER AND SAVE THEM ON THE STACK -
				FCBBUF,X	
Ø399	C2D7 34 16		L2H2	Х,В,А	* THESE ARE THE INITIAL VALUES OF A TEMPORARY
0400		*			* RECORD LENGTH COUNTER AND RANDOM BUFFER
0401		*			* POINTER WHICH ARE MAINTAINED ON THE STACK
0402	C2D9 3Ø 5E		LEAX		POINT X TO (RECORD NUMBER -1)
0403	C2DB BD 9F B5		JSR	L9FB5	MULT (UNSIGNED) RECORD LENGTH X (RECORD NUMBER -1)
0404	C2DE 34 60		PSHS	U,Y	SAVE PRODUCT ON THE STACK
0405	C2EØ A6 EØ		LDA		CHECK MS BYTE OF PRODUCT
0406	C2E2 26 Ø9			LC2ED	'BR' ERROR IF NOT ZERO (RECORD NUMBER TOO BIG)
0407	C2E4 35 1Ø		PULS		* PULL THE BOTTOM 3 PRODUCT BYTES OFF THE STACK;
Ø4Ø8	C2E6 35 Ø4		PULS		
	C2E0 35 04	*	FULS	В	* TOP TWO IN X, BOTTOM IN ACCB; ACCB POINTS TO
0409					* THE FIRST BYTE OF THE SECTOR USED BY THIS RECORD,
0410		*			* (X) CONTAINS THE SECTOR OFFSET (IN WHICH SECTOR
Ø411		*			* FROM THE START THE BYTE IS LOCATED)
0412	C2E8 8C Ø2 64	LC2E8	CMPX	#(TRKMAX-1)	612 SECTORS MAX IN A RANDOM FILE
Ø413	C2EB 25 Ø5		BLO	LC2F2	BRANCH IF RECORD LENGTH O.K.
0414	C2ED C6 36	LC2ED	LDB	#2*27	'BAD RECORD' ERROR
Ø415	C2EF 7E AC 46		JMP	LAC46	JUMP TO ERROR HANDLER
Ø416	C2F2 DE F1	LC2F2		FCBTMP	POINT U TO FCB
Ø417	C2F4 AC 4D			FCBSOF,U	* COMPARE SAVED SECTOR OFFSET TO THE CURRENT SECTOR OFFSET
Ø418					* BEING PROCESSED - DO NOT PROCESS A NEW SECTOR IF THEY ARE EQUAL
	C2F6 10 27 00 B7			LC3B1	
0419	C2FA 34 14		PSHS		SAVE BYTE AND SECTOR OFFSET TO RECORD START ON STACK
0420	C2FC A6 4F			FCBFLG,U	* CHECK FCB GET/PUT FLAG AND
Ø421	C2FE 27 Ø6			LC306	* BRANCH IF IT WAS A GET
Ø422	C300 6F 4F		CLR	FCBFLG,U	FORCE GET/PUT TO 'PUT'
Ø423	C3Ø2 C6 Ø3		LDB	#\$Ø3	DSKCON WRITE OP CODE
Ø424	C3Ø4 8D 33		BSR	LC339	GO WRITE A SECTOR - SAVE 'PUT' DATA ON DISK
Ø425		* CONVER	T THE	SECTOR OFFSET TO A GRANULE AND SE	
Ø426	C3Ø6 EC 61		LDD	\$Ø1,S	* GET THE NUMBER OF SECTORS TO THE START OF
Ø427	C3Ø8 BD C7 54			LC754	* THIS RECORD NUMBER AND CONVERT THEM TO A GRANULE OFFSET
Ø428	C3ØB 34 Ø4		PSHS		SAVE GRANULE OFFSET ON THE STACK
Ø429	C3ØD BD C7 49			LC749	MULTIPLY GRANULE NUMBER X 9 - CONVERT TO NUMBER OF SECTORS
0430	C31Ø 5Ø		NEGB		* NEGATE LS BYTE OF GRANULE OFFSET AND ADD THE
Ø431	C311 EB 63		ADDB	\$Ø3,S	* LS BYTE OF SECTOR OFFSET - ACCB = SECTOR
Ø432		*			* NUMBER (Ø-8) CORRESPONDING TO THE SECTOR NUMBER WITHIN A
Ø433		*			* GRANULE OF THE LAST SECTOR OF THE SECTOR OFFSET
Ø433 Ø434	C313 5C		INCB		* GRANULE OF THE LAST SECTOR OF THE SECTOR OFFSET = ADD ONE - SECTORS SAVED IN THE FCB: START
Ø434	C313 5C C314 F7 44		INCB STR	FCRSFC II	= ADD ONE - SECTORS SAVED IN THE FCB; START
Ø434 Ø435	C314 E7 44		STB	FCBSEC,U	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB
Ø434 Ø435 Ø436	C314 E7 44 C316 E6 42		STB LDB	FCBFGR,U	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE
0434 0435 0436 0437	C314 E7 44 C316 E6 42 C318 BD C7 25		STB LDB JSR	FCBFGR,U LC725	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT 0 - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT
Ø434 Ø435 Ø436 Ø437 Ø438	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6		STB LDB JSR LEAU	FCBFGR,U LC725 FATCON,X	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA
Ø434 Ø435 Ø436 Ø437 Ø438 Ø439	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4		STB LDB JSR LEAU LDA	FCBFGR,U LC725	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD
Ø434 Ø435 Ø436 Ø437 Ø438 Ø439 Ø44Ø	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C		STB LDB JSR LEAU LDA INCA	FCBFGR,U LC725 FATCON,X ,S	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT 0 - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW)
Ø434 Ø435 Ø436 Ø437 Ø438 Ø439 Ø44Ø	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4		STB LDB JSR LEAU LDA	FCBFGR,U LC725 FATCON,X ,S	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA
Ø434 Ø435 Ø436 Ø437 Ø438 Ø439 Ø44Ø	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C	LC32Ø	STB LDB JSR LEAU LDA INCA	FCBFGR,U LC725 FATCON,X ,S	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT 0 - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW)
Ø434 Ø435 Ø436 Ø437 Ø438 Ø439 Ø44Ø	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4	LC32Ø	STB LDB JSR LEAU LDA INCA LEAX	FCBFGR,U LC725 FATCON,X ,S	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA
Ø434 Ø435 Ø436 Ø437 Ø438 Ø439 Ø44Ø Ø441	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A	LC32Ø	STB LDB JSR LEAU LDA INCA LEAX ABX DECA	FCBFGR,U LC725 FATCON,X ,S	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT 0 - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 96 C310 A6 E4 C317 4C C320 30 C4 C322 3A C323 4A C324 27 37	LC32Ø	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ	FCBFGR,U LC725 FATCON,X ,S ,U	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C323 4A C324 27 37 C326 E7 E4	LC32Ø	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB	FCBFGR,U LC725 FATCON,X ,S ,U	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 Ø6 C31D A6 E4 C327 AC C328 30 C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84	LC32Ø	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB	FCBFGR,U LC725 FATCON,X ,S ,U	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT 0 - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0446	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C324 C1 CØ	LC32Ø	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB CMPB	FCBFGR,U LC725 FATCON,X ,S ,U LC35D ,S ,X	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT 0 - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0446 0447	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 Ø6 C31D A6 E4 C327 AC C328 30 C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84	LC32Ø	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB	FCBFGR,U LC725 FATCON,X ,S ,U	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT 0 - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0445 0446 0447	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C324 C1 CØ	LC32Ø	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB CMPB BLO	FCBFGR,U LC725 FATCON,X ,S ,U LC35D ,S ,X ,X #\$CØ LC32Ø	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING
0434 0435 0436 0437 0438 0439 0449 0441 0442 0443 0444 0445 0447 0445 0449	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 Ø6 C310 A6 E4 C317 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C324 C1 CØ C324 C5 F2	LC32Ø * THE GR	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB CMPB BLO	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #SCØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT 0 - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  BERANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING
0434 0435 0436 0437 0438 0449 0441 0442 0443 0444 0445 0447 0448 0447 0448	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C31F 4C C320 30 C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C324 C1 C0 C320 25 F2 C32E E6 E4	LC32Ø * THE GR	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB CMPB BLO ANULE LDB	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT,S	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LLY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0449 0450 0451	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 Ø6 C31D A6 E4 C327 A6 C322 3A C323 3A C324 27 37 C326 E7 E4 C328 E6 84 C324 C1 CØ C326 E5 F2  C326 E6 E4 C337 ØD D8	LC32Ø * THE GR	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB CMPB BLO ANULE LDB TST	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT ,S VDB	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT 0 - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0450 0451	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 33 Ø6 C310 A6 E4 C329 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 E4 C330 ØD D8 C332 E6 E4 C330 ØD D8 C332 E6 14	LC32Ø * THE GRA	STB LDB JSR LEAU LDA INCA LEAX DECA BEQ STB LDB CMPB BLO ANULE LDB TST BNE	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VD8 LC348	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT
0434 0435 0436 0437 0449 0449 0441 0442 0443 0444 0445 0446 0447 0451 0452 0453	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C32Ø E6 84 C32A C1 CØ C32C 25 F2 C32E E6 E4 C33Ø ØD D8 C332 26 14 C334 C6 2E	LC32Ø * THE GR	STB LDB JSR LEAU LDA INCA LEAX DECA BEQ STB LDB CMPB BLO ANULE LDB TST BNE LDB	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC348 #2*23	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LLY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT  'INPUT PAST END OF FILE' ERROR
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0451 0452 0453 0453	C314 E7 44 C316 E6 42 C318 B0 C7 25 C318 B3 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø Ø0 D8 C332 26 14 C334 C6 2E C336 7E AC 46	LC32Ø  * THE GRA	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB CMPB BLO ANULE LDB TST BNE LDB JMP	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC348 #2*23 LAC46	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LY DEFINED IN THIS RANDOM FILE  & CHECK GET/PUT FLAG  * AND BRANCH IF PUT  'INPUT PAST END OF FILE' ERROR  JUMP TO ERROR HANDLER
0434 0435 0436 0437 0449 0449 0441 0442 0443 0444 0445 0446 0447 0451 0452 0453	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C32Ø E6 84 C32A C1 CØ C32C 25 F2 C32E E6 E4 C33Ø ØD D8 C332 26 14 C334 C6 2E	LC32Ø  * THE GRA	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB CMPB BLO ANULE LDB TST BNE LDB JMP	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC348 #2*23	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LLY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT  'INPUT PAST END OF FILE' ERROR
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0451 0452 0453 0453	C314 E7 44 C316 E6 42 C318 B0 C7 25 C318 B3 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø Ø0 D8 C332 26 14 C334 C6 2E C336 7E AC 46	LC32Ø  * THE GRA	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ STB LDB CMPB BLO ANULE LDB TST BNE LDB JMP	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC348 #2*23 LAC46	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LY DEFINED IN THIS RANDOM FILE  & CHECK GET/PUT FLAG  * AND BRANCH IF PUT  'INPUT PAST END OF FILE' ERROR  JUMP TO ERROR HANDLER
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0449 0451 0453 0453 0454	C314 E7 44 C316 E6 42 C318 B0 C7 25 C318 B3 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø Ø0 D8 C332 26 14 C334 C6 2E C336 7E AC 46	* THE GRALLC334	STB LDB JSR LEAU LDA INCA LEAX ABX DECA BEQ CMPB BLO ANULE LDB TST BNE LDB TST BNE LDB LDB LDB LDB LDB LDB LDB LDB LDB LDB	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC348 #2*23 LAC46	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT 0 - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  "LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT  "INPUT PAST END OF FILE' ERROR  JUMP TO ERROR HANDLER  POINT X TO FCB DATA BUFFER
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0446 0447 0451 0452 0453 0455 0456 0457	C314 E7 44 C316 E6 42 C318 B0 C7 25 C318 B3 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø Ø0 D8 C332 26 14 C334 C6 2E C336 7E AC 46	* THE GRALC334 LC339 * READ/WI	STB LDB JSR LEAU LEAU LEAU ABX ABX ABX BEQ STB LDB LDB LDB LDB LDB LDB LDB LDB LDB LD	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC348 #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT  'INPUT PAST END OF FILE' ERROR  JUMP TO ERROR HANDLER  POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0449 0450 0451 0453 0454 0456 0457	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 B4 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD DB C33C 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA	* THE GR/ LC334 LC339 * READ/WI	STB LDB JSR LEAU LDA INCA BEQ STB LDB CMPB BLO ANULE LDB STB LDB STB LDB CMPB BLO ANULE LDB JMP LEAX	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #8C0 LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VD8 LC348 #2*23 LAC46 FCBCON,U  A SECTOR. ENTER WITH OP CODE IN AC	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  ICB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE
0434 0435 0436 0437 0449 0449 0441 0442 0443 0444 0445 0446 0447 0451 0452 0453 0454 0455 0456 0457 0458	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C322 3A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C33C 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33C D7 EA C33C D7 EA	* THE GRALLC334 LC339 * READ/WILC33C	STB LDB JSR LEAU LDA INCA ABX DECA ABX DECA BEQ STB LDB LDB LDB TST BNE LDB LDB LDB LEAX RITE / X STB STX	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT,S VDB HC34B #2*23 LAC46 FCBCON,U A CCOPC DCBPT	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  "LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT  "INPUT PAST END OF FILE' ERROR  JUMP TO ERROR HANDLER  POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X  SAVE DSKCON OPERATION CODE VARIABLE  SAVE DSKCON LOAD BUFFER VARIABLE
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0446 0447 0451 0452 0453 0454 0455 0456 0457 0458 0458 0459	C314 E7 44 C316 E6 42 C318 B0 C7 25 C318 B3 Ø6 C31D A6 E4 C32F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C332 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33E P EE C33E P EE C33E P EE C33B P EE C34B 3Ø C4	* THE GRALLC334 LC339 * READ/WILC33C	STB LDB JSR LEAU LDA ANDA LEAX DECA BEQ LDB LDB LDB STB STB LDB LDB STB LDB LDB STB STB LDB STB STB LDB STB STB LDB STB STB LDB STB STB LDB STB STB LDB STB STB STB LDB STB STB LDB STB STB STB STB STB STB STB STB STB ST	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC348 #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT  'INPUT PAST END OF FILE' ERROR  JUMP TO ERROR HANDLER  CCB, BUFFER PTR IN X  SAVE DSKCON OPERATION CODE VARIABLE  SAVE DSKCON LOAD BUFFER VARIABLE  POINT X TO FCB  POINT X TO FCB
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0446 0446 0450 0451 0453 0454 0456 0457 0456 0457 0456 0457	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 B4 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD DB C33C 25 F2  C32E E6 E4 C33Ø ØD DB C33C 25 F2  C32E E6 C4 C33Ø ØD DB C33C 25 F2  C32E E6 C4 C33Ø ØD DB C33C D7 EA C33E PF EE C34Ø 3Ø C4 C34Z BD C7 33	* THE GRALLC334 LC339 * READ/WILC33C	STB LDB LEAX DECA STB LDA STB LEAV LEAX ABX DECA BEQ LDB LDB LDB LDB LDB LDB STS LDB LDB STS LDB LDB LEAX STB LDB LDB LEAX STB LDB LEAX STB STX LEAX JSR	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #8500 LC320  BEING SEARCHED FOR IS NOT PRESENT,S VDB LC348 #2*23 LAC46 FCBCON,U  A SECTOR. ENTER WITH OP CODE IN ACC DCOPC DCBPT ,U LC733	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON LOAD BUFFER VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES
0434 0435 0436 0437 0449 0449 0441 0442 0443 0444 0445 0446 0447 0451 0452 0451 0452 0455 0456 0457 0458 0460 0461	C314 E7 44 C316 E6 42 C318 B0 C7 25 C318 B3 Ø6 C31D A6 E4 C32F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C332 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33E P EE C33E P EE C33E P EE C33B P EE C34B 3Ø C4	* THE GRALLC334 LC339 * READ/WILC33C	STB LDB LEAX DECA STB LDA STB LEAV LEAX ABX DECA BEQ LDB LDB LDB LDB LDB LDB STS LDB LDB STS LDB LDB LEAX STB LDB LDB LEAX STB LDB LEAX STB STX LEAX JSR	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC320  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC348 #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT  'INPUT PAST END OF FILE' ERROR  JUMP TO ERROR HANDLER  CCB, BUFFER PTR IN X  SAVE DSKCON OPERATION CODE VARIABLE  SAVE DSKCON LOAD BUFFER VARIABLE  POINT X TO FCB  POINT X TO FCB
0434 0435 0436 0437 0438 0449 0441 0442 0443 0444 0445 0445 0445 045 045 045 045 045	C314 E7 44 C316 E6 42 C318 BD C7 25 C31B 33 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 B4 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD DB C33C 25 F2  C32E E6 E4 C33Ø ØD DB C33C 25 F2  C32E E6 C4 C33Ø ØD DB C33C 25 F2  C32E E6 C4 C33Ø ØD DB C33C D7 EA C33E PF EE C34Ø 3Ø C4 C34Z BD C7 33	* THE GRALC334 LC339 * READ/WILC33C	STB LDB LDB LEAU LDA LEAX ABX DECA STB LDB BLO STB LDB BLO ANULE LDB TST BNE LDB JMP LEAX STB STB LBB LDB JMP LEAX JSR JMP	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC34B #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  NO - KEEP LOOKING  "LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  ** CHECK GET/PUT FLAG  ** AND BRANCH IF PUT  'INPUT PAST END OF FILE' ERROR  JUMP TO GRROR HANDLER  POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X  SAVE DSKCON LOAD BUFFER VARIABLE  SAVE DSKCON OPERATION CODE VARIABLE  SAVE DSKCON OLOAD BUFFER VARIABLE  POINT X TO FCB  CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES  READ/WRITE A TRACK OR SECTOR
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0446 0447 0445 0450 0451 0452 0453 0454 0459 0451 0452 0453 0454 0456 0457	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C32Ø E6 E4 C32Ø E6 E4 C33Ø ØD DB C33C 25 F2 C32E E6 E4 C33Ø ØD DB C33C 25 F2 C33E E6 E4 C33Ø ØD DB C33C 25 F2 C33E E6 E4 C33Ø ØD DB C33C 25 F2 C33E E6 E4 C33Ø ØD DB C33C 25 F2 C34E E6 E4 C33Ø ØD DB C33C 25 F2 C34E E6 E4 C33Ø ØD DB C33C D7 EA C33E 9F EE C34Ø 3Ø C8 19 C33C D7 EA C33E 9F EE C34Ø 3Ø C4 C342 BD C7 33 C345 7E D5 FF	* THE GRALC334 LC339 * READ/WILC33C * 'PUT' I	STB LDB LCBAU LDA LCBAU LDA LEAX ABX ABX DECA STB LDB LDB LDB TST BNE LDB JMP LEAX RITE LFAX JSR JMP DATA  LEAX JSR JMP DATA LEAX JSR JMP	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C00 LC320  BEING SEARCHED FOR IS NOT PRESENT, S VD8 LC348 #2*23 LAC46 FCBCON,U  A SECTOR. ENTER WITH OP CODE IN ACDOPC DCBPT ,U LC733 LD5FF	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR
0434 0435 0436 0437 0449 0449 0441 0442 0443 0444 0445 0446 0447 0451 0452 0451 0452 0453 0454 0455 0466 0467 0468	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C327 A6 C322 3A C323 34 C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C33C 25 F2  C32E E6 E4 C33Ø ØD B8 C332 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33E 9F EE C34Ø 3Ø C4 C34E BD C7 33 C345 7E D5 FF	* THE GRALC334 LC339 * READ/WILC33C * 'PUT' I	STB LDB LDB LDA LEAU LDA LEAX ABX DECA BEQ STB LDB LDB LDB LDB LDB LDB LDB LDB STST LDB LDB STST LDB LDB JMP LEAX STX LEAX JMP DATA PSHS	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT, S VDB LC34B #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCOPCT ,U LC733 LD5FF INTO A GRANULE NOT PRESENTLY INCLU	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT 0 - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER CCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0446 0447 0445 0450 0451 0452 0453 0454 0459 0451 0452 0453 0454 0456 0457	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 Ø6 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C32Ø E6 E4 C32Ø E6 E4 C33Ø ØD DB C33C 25 F2 C32E E6 E4 C33Ø ØD DB C33C 25 F2 C33E E6 E4 C33Ø ØD DB C33C 25 F2 C33E E6 E4 C33Ø ØD DB C33C 25 F2 C33E E6 E4 C33Ø ØD DB C33C 25 F2 C34E E6 E4 C33Ø ØD DB C33C 25 F2 C34E E6 E4 C33Ø ØD DB C33C D7 EA C33E 9F EE C34Ø 3Ø C8 19 C33C D7 EA C33E 9F EE C34Ø 3Ø C4 C342 BD C7 33 C345 7E D5 FF	* THE GRALC334 LC334 LC336 * READ/WILC33C * 'PUT' ILC348	STB LDB LDB LEAU LEAU LDA DECA BEQ STB LDB CMPB BLO ANULE LDB	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC34B #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF LINTO A GRANULE NOT PRESENTLY INCLU X,A LC78F	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR
0434 0435 0436 0437 0449 0449 0441 0442 0443 0444 0445 0446 0447 0451 0452 0451 0452 0453 0454 0455 0466 0467 0468	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C327 A6 C322 3A C323 34 C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C33C 25 F2  C32E E6 E4 C33Ø ØD B8 C332 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33E 9F EE C34Ø 3Ø C4 C34E BD C7 33 C345 7E D5 FF	* THE GRALC334 LC334 LC336 * READ/WILC33C * 'PUT' ILC348	STB LDB LDB LDA LEAU LDA LEAX ABX DECA BEQ STB LDB LDB LDB LDB LDB LDB LDB LDB STST LDB LDB STST LDB LDB JMP LEAX STX LEAX JMP DATA PSHS	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC34B #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF LINTO A GRANULE NOT PRESENTLY INCLU X,A LC78F	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT 0 - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER CCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0446 0447 0450 0451 0452 0453 0454 0456 0457 0456 0457 0456 0457 0466 0467 0468	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 Ø6 C31D A6 E4 C327 A4 C328 BD C7 25 C318 B3 Ø6 C31D A6 E4 C329 30 C4 C322 3A C324 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD DB C332 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33E 9F EE C34B 3Ø C4 C342 BD C7 33 C345 7E D5 FF  C348 34 12 C348 BD C7 8F	* THE GRALC334 LC334 LC336 * READ/WILC33C * 'PUT' ILC348	STB LDB LDB LEAU LDA LEAX ABX DECA BEQ STB BLOB CMPB BLO ANULE LDB	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C00 LC320  BEING SEARCHED FOR IS NOT PRESENT, S VD8 LC348 #2*23 LAC46 FCBCON,U  A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF  ENTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING  LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON OPERATION CODE VARIABLE POINT X TO FCB CONVENT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  LIDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE NOBER IN ACCB
0434 0435 0436 0437 0449 0449 0441 0442 0443 0444 0445 0446 0447 0451 0452 0451 0452 0453 0454 0455 0466 0467 0466 0467	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C327 AC C328 AC C322 AA C324 A7 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C33C 25 F2  C32E E6 E4 C33Ø ØD D8 C33C 25 F2  C32E E6 E4 C33Ø ØD D8 C33C 25 F2  C32E E6 E4 C33Ø ØD D8 C33C 25 F2  C34E C5 E4 C33Ø ØD D8 C33C 25 F2  C34E C5 E6 C34C C5 F2  C34E C5 E6 C34C C5 F2  C34E C6 E4 C33Ø ØD D8 C33C C7 F2 C34E C7	* THE GRALC334 LC334 LC336 * READ/WILC33C * 'PUT' ILC348	STB LDB LEAX ABX ABX ABX ABX ABX ABX ABX ABX ABX A	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT, S VDB LC348 #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF (NTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,B A,B	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT Ø - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  "LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT  "INPUT PAST END OF FILE' ERROR  JUMP TO ERROR HANDLER  POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X  SAVE DSKCON OPERATION CODE VARIABLE  SAVE DSKCON LOAD BUFFER VARIABLE  POINT X TO FCB  CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES  READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE  SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE  FIND FIRST FREE GRANULE IN FAT  SAVE FREE GRANULE NUMBER IN ACCCB  PULL LAST GRANULE NUMBER IN ACCCB
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0445 0445 0445 0453 0454 0451 0452 0453 0454 0456 0457 0461 0462 0461 0462 0464 0467 0468 0467 0468	C314 E7 44 C316 E6 42 C318 B0 C7 25 C318 B3 Ø6 C31D A6 E4 C32F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C332 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33E 9F EE C33B 9F EE C34B 3Ø C4 C342 BD C7 33 C345 7E D5 FF  C34B 34 12 C34A BD C7 BF C34D 1F 89 C34F 35 42 C35F C C4	* THE GRALC334 LC339 * READ/WILC33C * 'PUT' ILC348	STB LDB LDB LEAU LEAU LEAU LEAU LEAU LEAU LEAU LEAU	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT, S VDB LC348 #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF (NTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,B A,B	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE BERAMCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING  "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * AND BRANCH IF PUT "INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON LOAD BUFFER VARIABLE SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OF F OF STACK SAVE REMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0445 0445 0445 0450 0451 0452 0453 0454 0456 0457 0456 0457 0466 0462 0463 0462 0463 0464 0465 0466 0467 0468 0467 0468	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C31F 4C C32Ø 3Ø C4 C322 3A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C33C 25 F2  C32E E6 E4 C33Ø ØD C8 C33C 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33E 9F EE C34Ø 3Ø C4 C32B D7 EA C33E 9F EE C34Ø 3Ø C4 C33E 9F EE C34Ø 3Ø C4 C33E 9F EE C34Ø 3Ø C4 C34C BD C7 33 C345 7E D5 FF  C348 34 12 C34A BD C7 8F C34B 15 89 C34F 35 42 C351 E7 C4 C3553 4A	* THE GRALLC334 LC339 * READ/WILC33C * 'PUT' ILC348	STB LDB LEAX DECA ANDLE LEAX DECA ANDLE LEAX STB BLO ANDLE LDB LDB LST BNE LEAX STX LEAX JMP DATA 1 JSR	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C00 LC320  BEING SEARCHED FOR IS NOT PRESENT, S VD8 LC348 #2*23 LAC46 FCBCON,U  A SECTOR. ENTER WITH OP CODE IN ACC DCOPC DCBPT ,U LC733 LD5FF  ENTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,U ,U	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING  "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON OPERATION CODE VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  DIDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE COUNTER AND COUNTER OFF OF STACK SAVE DSKCON GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE DRIVE FORD GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE COUNTER
0434 0435 0436 0437 0449 0449 0441 0442 0443 0444 0445 0446 0447 0451 0452 0453 0454 0455 0456 0467 0468 0467 0466 0467 0466 0467 0469 0470 0470	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C327 AC C328 AC C322 AA C324 A7 C326 E7 E4 C328 E6 E4 C328 E6 E4 C328 C1 C0 C320 C25 F2  C326 E6 E4 C330 ØD D8 C332 26 14 C334 C6 2E C336 7E AC 46 C339 30 C8 19  C33C D7 EA C32C D7 EA	* THE GRALC334 LC339 * READ/WILC33C * 'PUT' ILC348	STB LDB LDA LEAX ABX ABX ABX ABX ABX ABX ABX ABX ABX A	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT, S VDB LC348 #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF INTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,U ,U LC348	= ADD ONE - SECTORS SAVED IN THE FCB; START  = AT 1 NOT 0 - SAVE IT IN THE FCB  GET FIRST GRANULE IN FILE  POINT X TO FAT  POINT U TO FAT DATA  GET NUMBER OF GRANULES OFFSET TO RECORD  ADD ONE (COMPENSATE FOR DECA BELOW)  POINT X TO FAT DATA  POINT X TO FAT DATA  POINT X TO CORRECT GRANULE  DECREMENT GRANULE COUNTER  BRANCH IF CORRECT GRANULE FOUND  SAVE GRANULE ADDRESS ON STACK  GET NEXT GRANULE IN FILE  LAST GRANULE IN FILE?  NO - KEEP LOOKING  "LY DEFINED IN THIS RANDOM FILE  GET OFFSET TO LAST GRANULE  * CHECK GET/PUT FLAG  * AND BRANCH IF PUT  "INPUT PAST END OF FILE' ERROR  JUMP TO ERROR HANDLER  POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X  SAVE DSKCON OPERATION CODE VARIABLE  SAVE DSKCON LOAD BUFFER VARIABLE  POINT X TO FCB  CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES  READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE  SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE  FIND FIRST FREE GRANULE IN FAT  SAVE FREE GRANULE NUMBER IN ACCB  PULL LAST GRANULE POINTER AND COUNTER OF F OF STACK  SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE  DECREMENT GRANULE COUNTER  GET ANOTHER GRANULE IN FOUND DONE
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0445 0453 0454 0453 0454 0456 0457 0456 0457 0461 0462 0463 0461 0465 0467 0468 0467 0468	C314 E7 44 C316 E6 42 C318 B0 C7 25 C318 B3 Ø6 C31D A6 E4 C32F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C332 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33E 9F EE C34B 3Ø C4 C34B BD C7 33 C345 7E D5 FF  C34B 34 12 C34A BD C7 BF C34D 1F 89 C34F 35 42 C351 E7 C4 C353 4A C354 26 F2 C356 34 14	* THE GR/ LC334 LC339 * READ/WI LC33C * 'PUT' I	STB LDB LDB LEAX ABXX DECA LEAX ABXX BEQ STB LDB LDB LDB LDB LDB LDB LDB LDB LDB LD	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT ,S VD8 LC34B #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF INTO A GRANULE NOT PRESENTLY INCLU X ,A LC78F A,B A,U ,U LC348 X,B	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT UTO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON LOAD BUFFER VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE COUNTER GET ANOTHER GRANULE FOND DONE SAVE POINTER TO LAST GRANULE DECREMENT GRANULE OUNTER GET ANOTHER GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE FOND DONE SAVE POINTER TO LAST GRANULE AND OFFSET
0434 0435 0436 0437 0439 0440 0441 0442 0443 0444 0445 0445 0445 0445 0450 0451 0453 0454 0450 0451 0456 0457 0456 0457 0468 0461 0462 0463 0464 0466 0467 0468 0468 0469 0471 0472 0472	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C327 A6 C322 AA C324 27 37 C326 E7 E4 C328 E6 E4 C330 ØD D8 C320 25 F2 C326 E6 E4 C339 ØC B1 C330 ØC B1 C300 ØC B1 C30	* THE GRALLC334 LC339 * READ/WILC33C * 'PUT' ILC348	STB LDB LEAX A BECA ANULE LDB BLO ANULE BBLO ANULE LDB LDB LDB LDB STB LEAX A STB LEAX JMP DATA STR TRITE A STB LEAX JMP DATA STR TRITE A STB LEAX A STB LEAX JMP DATA STR TRITE A STB LEAX JMP DATA STR TRITE A STB LEAX JMP DATA STR TRITE A STB LEAX JMP PSHS STB LEAX JMP PSHS STB LEAX JSR DATA STR TRITE A STB LEAX JSR	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C00 LC320  BEING SEARCHED FOR IS NOT PRESENT, S VD8 LC348 #2*23 LAC46 FCBCON,U  A SECTOR. ENTER WITH OP CODE IN ACC DCOPC DCBPT ,U LC733 LD5FF  ENTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,U ,U LC348 X,B LC66F1	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON OPERATION CODE VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE COUNTER AND COUNTER OFF OF STACK SAVE DSKCON USER GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE IF NOT DONE SAVE NEMLY FOUND GRANULE AND OFFSET WRITE FAT TO DISK
0434 0435 0436 0437 0438 0449 0441 0442 0443 0444 0445 0445 0445 0451 0452 0453 0454 0456 0456 0456 0456 0461 0465 0461 0465 0466 0467 0469 0469 0470 0472 0473	C314 E7 44 C316 E6 42 C318 B0 C7 25 C318 B3 Ø6 C31D A6 E4 C32F 4C C32Ø 3Ø C4 C322 3A C323 4A C324 27 37 C326 E7 E4 C328 E6 84 C32A C1 CØ C32C 25 F2  C32E E6 E4 C33Ø ØD D8 C332 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C33E 9F EE C34B 3Ø C4 C34B BD C7 33 C345 7E D5 FF  C34B 34 12 C34A BD C7 BF C34D 1F 89 C34F 35 42 C351 E7 C4 C353 4A C354 26 F2 C356 34 14	* THE GRALLC334 LC339 * READ/WILC33C * 'PUT' ILC348	STB LDB LDB LEAX ABXX DECA LEAX ABXX BEQ STB LDB LDB LDB LDB LDB LDB LDB LDB LDB LD	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C00 LC320  BEING SEARCHED FOR IS NOT PRESENT, S VD8 LC348 #2*23 LAC46 FCBCON,U  A SECTOR. ENTER WITH OP CODE IN ACC DCOPC DCBPT ,U LC733 LD5FF  ENTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,U ,U LC348 X,B LC66F1	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT UTO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON LOAD BUFFER VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE COUNTER GET ANOTHER GRANULE FOND DONE SAVE POINTER TO LAST GRANULE DECREMENT GRANULE OUNTER GET ANOTHER GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE FOND DONE SAVE POINTER TO LAST GRANULE AND OFFSET
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0445 0450 0451 0453 0454 0450 0451 0456 0457 0456 0457 0456 0457 0456 0467 0463 0461 0462 0463 0461 0467 0468 0467 0468 0467 0468	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C327 A6 C322 AA C324 27 37 C326 E7 E4 C328 E6 E4 C330 ØD D8 C320 25 F2 C326 E6 E4 C339 ØC B1 C330 ØC B1 C300 ØC B1 C30	* THE GRALC334 LC339 * READ/WILC33C * 'PUT' ILC348	STB LDB LDA ANULE LDA BEQ LDB BLO BBLO BLOB BLO BLOB BLO BLOB BLO BLO	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT,S VDB LC348 #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF LNTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,U ,U LC348 X,B LC66f1 B,X	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  LCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  LDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE FOR NOT DONE SAVE POINTER TO LAST GRANULE AND OFFSET WRITE FAT TO DISK RESTORE POINTER AND OFFSET
0434 0435 0436 0437 0438 0449 0441 0442 0443 0444 0445 0445 0445 0451 0452 0453 0454 0456 0456 0456 0456 0461 0465 0461 0465 0466 0467 0469 0469 0470 0472 0473	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C327 A6 C322 AA C324 27 37 C326 E7 E4 C328 E6 E4 C330 ØD D8 C320 25 F2 C326 E6 E4 C339 ØC B1 C330 ØC B1 C300 ØC B1 C30	* THE GRALC334 LC339 * READ/WILC33C * 'PUT' ILC348	STB LDB LDA ANULE LDA BEQ LDB BLO BBLO BLOB BLO BLOB BLO BLOB BLO BLO	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C00 LC320  BEING SEARCHED FOR IS NOT PRESENT, S VD8 LC348 #2*23 LAC46 FCBCON,U  A SECTOR. ENTER WITH OP CODE IN ACC DCOPC DCBPT ,U LC733 LD5FF  ENTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,U ,U LC348 X,B LC66F1	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  LCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  LDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE FOR NOT DONE SAVE POINTER TO LAST GRANULE AND OFFSET WRITE FAT TO DISK RESTORE POINTER AND OFFSET
0434 0435 0436 0437 0438 0439 0440 0441 0442 0443 0444 0445 0446 0447 0448 0445 0450 0451 0453 0454 0450 0451 0456 0457 0456 0457 0456 0457 0456 0467 0463 0461 0462 0463 0461 0467 0468 0467 0468 0467 0468	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C327 A6 C322 AA C324 27 37 C326 E7 E4 C328 E6 E4 C330 ØD D8 C320 25 F2 C326 E6 E4 C339 ØC B1 C330 ØC B1 C300 ØC B1 C30	* THE GRALC334 LC339 * READ/WILC33C * 'PUT' ILC348	STB LDB LDA ANULE LDB	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$C0 LC320  BEING SEARCHED FOR IS NOT PRESENT,S VDB LC348 #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF LNTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,U ,U LC348 X,B LC66f1 B,X	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE? NO - KEEP LOOKING "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  LCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  LDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE FOR NOT DONE SAVE POINTER TO LAST GRANULE AND OFFSET WRITE FAT TO DISK RESTORE POINTER AND OFFSET
0434 0435 0436 0437 0438 0449 0441 0442 0443 0444 0445 0445 0445 0451 0452 0453 0454 0455 0456 0457 0461 0462 0461 0462 0467 0468 0467 0468 0467 0468 0467 0468 0467 0468 0470 0471 0472 0473	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 Ø6 C31D A6 E4 C327 AC C328 AC C322 AA C324 27 37 C326 E7 E4 C328 E6 E4 C328 E6 E4 C328 C1 CØ C320 Z5 F2  C326 E7 E4 C330 ØD DB C332 26 14 C334 C6 2E C336 7E AC 46 C339 3Ø C8 19  C33C D7 EA C34C D7 EA C35C D7 E	* THE GRALC334 LC339 * READ/WILC33C * 'PUT' ILC348	STB LDB LEAX ABX ABX BEQ STB BLO BEQ STB BLO BLO BLO BLO BLO BLO BLO BLO BLO BL	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT ,S VDB LC34B #2*23 LAC46 FCBCON,U A ASECTOR. ENTER WITH OP CODE IN AC DCOPC DCBPT ,U LC733 LD5FF INTO A GRANULE NOT PRESENTLY INCLU X,A LC78F A,B A,U ,U ,U LC348 X,B LC66F1 B,X I GRANULE IS FOUND, FIND THE RIGHT \$01,S	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT U TO FAT DATA GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO CORRECT GRANULE DECREMENT GRANULE COUNTER BRANCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE LAST GRANULE IN FILE LAST GRANULE IN FILE GET OFFSET TO LAST GRANULE * CHECK GET/PUT FLAG * * AND BRANCH IF PUT 'INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON OPERATION CODE VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE COUNTER AND COUNTER OF F OF STACK SAVE DSK GRANULE COUNTER AND COUNTER OF F OF STACK SAVE READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND COUNTER OF F OF STACK SAVE READ/WRITE A TRACK OR SECTOR  IDED IN THIS FILE SAVE GRANULE COUNTER AND COUNTER OF F OF STACK SAVE REMALY FOUND GRANULE NUMBER IN ACCB PULL LAST GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OF F OF STACK SAVE NEMLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE IF NOT DONE SAVE POINTER TO LAST GRANULE AND OFFSET WRITE FAT TO DISK RESTORE POINTER AND OFFSET  SECTOR
0434 0435 0436 0437 0443 0449 0441 0442 0443 0444 0445 0446 0447 0452 0453 0454 0455 0456 0457 0468 0469 0462 0463 0464 0466 0467 0468 0469 0471 0472 0475 0475 0475	C314 E7 44 C316 E6 42 C318 BD C7 25 C318 B3 06 C31D A6 E4 C327 A4 C328 A7 C324 A7 C324 A7 C324 E7 E4 C328 E6 E4 C328 E6 E4 C338 ØD D8 C322 C5 F2 C328 E6 E4 C339 ØD C8 C320 C7 C	* THE GRALC334 LC339 * READ/WILC33C * 'PUT' ILC348  * WHEN CCLC35D	STB LDB JSR LEAU LDA ABX DECA BEQ LDB BBLO BBLO BBLO BBLO BBLO BBLO BBLO	FCBFGR,U LC725 FATCON,X ,S ,U  LC35D ,S ,X #\$CØ LC32Ø  BEING SEARCHED FOR IS NOT PRESENT, S VDB LC34B #2*23 LAC46 FCBCON,U A SECTOR. ENTER WITH OP CODE IN AC DCOPC DCOPC DCOPC DCSPT ,U LC733 LT733 LT734 LC735 LT735 LT734 LC735 LT735 LT736 LT735 LT736 LT737 L	= ADD ONE - SECTORS SAVED IN THE FCB; START = AT 1 NOT Ø - SAVE IT IN THE FCB GET FIRST GRANULE IN FILE POINT X TO FAT POINT X TO FAT GET NUMBER OF GRANULES OFFSET TO RECORD ADD ONE (COMPENSATE FOR DECA BELOW) POINT X TO FAT DATA POINT X TO FAT DATA POINT X TO CORRECT GRANULE BERAMCH IF CORRECT GRANULE FOUND SAVE GRANULE ADDRESS ON STACK GET NEXT GRANULE IN FILE LAST GRANULE IN FILE NO - KEEP LOOKING  "LY DEFINED IN THIS RANDOM FILE GET OFFSET TO LAST GRANULE ** CHECK GET/PUT FLAG ** AND BRANCH IF PUT "INPUT PAST END OF FILE' ERROR JUMP TO ERROR HANDLER POINT X TO FCB DATA BUFFER  CCB, BUFFER PTR IN X SAVE DSKCON OPERATION CODE VARIABLE SAVE DSKCON LOAD BUFFER VARIABLE POINT X TO FCB CONVERT FCB TRACK AND SECTOR TO DSKCON VARIABLES READ/WRITE A TRACK OR SECTOR  DIED IN THIS FILE  SAVE GRANULE COUNTER AND POINTER TO LAST USED GRANULE FIND FIRST FREE GRANULE IN FAT SAVE FREE GRANULE NUMBER IN ACCB PULL LAST GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEWLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER TO LODE SAVE REANULE FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE POINTER AND COUNTER OFF OF STACK SAVE NEWLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE FOINTER AND COUNTER OFF OF STACK SAVE NEWLY FOUND GRANULE NUMBER IN ADDRESS OF LAST GRANULE DECREMENT GRANULE FOINTER AND COUNTER GET ANOTHER GRANULE FOINTER AND OFFSET WRITE FAT TO LAST GRANULE AND OFFSET  "SECTOR REMOVE GRAN NUMBER FROM STACK

Ø481	C363 86 FF		LDA	#\$FF	*SET FCBSOF,U TO ILLEGAL SECTOR OFFSET WHICH WILL
Ø482	C365 A7 4D			FCBSOF,U	*FORCE NEW SECTOR DATA TO BE READ IN
Ø483	C367 A6 84		LDA		GET CURRENT GRANULE
Ø484	C369 81 CØ		CMPA		IS IT THE LAST GRANULE?
Ø485 Ø486	C36B 25 27 C36D 84 3F		ANDA	LC394	NO
Ø487	C36F A1 44			FCBSEC,U	MASK OFF LAST GRANULE FLAG BITS * COMPARE CALCULATED SECTOR TO CURRENT SECTOR IN FCB
Ø488	C371 24 21			LC394	* AND BRANCH IF CALCULATED SECTOR IS > LAST SECTOR IN FILE
Ø489	C373 96 D8		LDA		= CHECK GET/PUT FLAG: IF 'GET' THEN 'INPUT
Ø49Ø	C375 27 BD			LC334	= PAST END OF FILE' ERROR
Ø491	C377 A6 44		LDA	FCBSEC,U	* GET CURRENT SECTOR NUMBER FROM FCB,
Ø492	C379 8A CØ		ORA		* OR IN THE LAST GRANULE FLAG BITS
Ø493	C37B A7 84		STA		* AND SAVE IN FAT
Ø494	C37D BD C5 7C			LC57C	WRITE FAT TO DISK IF NECESSARY
Ø495 Ø496	C38Ø AE 49 C382 8C Ø1 ØØ			FCBRLN,U #SECLEN	* GET RECORD LENGTH AND CHECK TO  * SEE IF IT IS SECLEN (EXACTLY ONE SECTOR)
Ø497	C385 26 Ø8			LC38F	BRANCH IF IT IS NOT EXACTLY ONE SECTOR
Ø498	C387 AC C8 13			FCBLST,U	=BRANCH IF THE NUMBER OF BYTES IN THE LAST SECTOR
Ø499	C38A 27 Ø8			LC394	=IS SET TO ONE SECTOR (SECLEN)
0500	C38C 86 81		LDA	#\$81	*SET THE PRESAVED FLAG (BIT15) AND FORCE
0501	C38E 21 4F	LC38E		LC3DF	*THE NUMBER OF BYTES IN LAST SECTOR TO 256
	C38F 4F	LC38F	CLRA		SET THE NUMBER OF BYTES IN LAST SECTOR TO ZERO
Ø5Ø3	C39Ø 5F		CLRB	FCDL CT II	CLEAR LS BYTE OF ACCD
Ø5Ø4 Ø5Ø5	C391 ED C8 13 C394 C6 Ø2	LC394	LDB	FCBLST,U	SAVE THE NUMBER OF BYTES IN LAST SECTOR DSKCON READ OP CODE
Ø5Ø6	C396 AE 49	20334		FCBRLN,U	* GET RECORD LENGTH AND COMPARE
Ø5Ø7	C398 8C Ø1 ØØ			#SECLEN	* IT TO SECLEN - EXACTLY ONE SECTOR
0508	C39B 26 ØD			LC3AA	BRANCH IF NOT EXACTLY ONE SECTOR LONG
0509	C39D 32 67			\$07,S	CLEAN UP STACK
	C39F AE 4B			FCBBUF,U	POINT X TO START OF RANDOM FILE BUFFER
Ø511 Ø512	C3A1 96 D8 C3A3 27 Ø2		LDA REO	VD8 LC3A7	* CHECK GET/PUT FLAG AND * BRANCH IF GET
	C3A3 27 02 C3A5 C6 03		LDB		DSKCON WRITE OP CODE
	> C3A7 7E C3 3C	LC3A7		LC33C	READ/WRITE A SECTOR
	> C3AA BD C3 39	LC3AA		LC339	READ A SECTOR INTO FCB DATA BUFFER
Ø516	C3AD 35 14		PULS		* GET BACK THE BYTE OFFSET TO RECORD: X = NUMBER OF
Ø517		*			* SECTORS; ACCB = BYTE POINTER IN SECTOR
	C3AF AF 4D			FCBSOF,U	SAVE SECTOR OFFSET IN FCB
Ø519 Ø52Ø	C3B1 34 Ø4	LC3B1	PSHS	LC725	SAVE BYTE OFFSET ON STACK
Ø520	C3B3 BD C7 25 C3B6 3Ø Ø6			FATCON, X	POINT X TO FILE ALLOCATION TABLE MOVE X TO FAT DATA
Ø522	C3B8 E6 43			FCBCGR, U	GET CURRENT GRANULE NUMBER
Ø523	C3BA 3A		ABX		POINT X TO PROPER GRANULE IN FAT
Ø524	C3BB A6 84		LDA		* GET CURRENT GRANULE AND CHECK TO
Ø525	C3BD 81 CØ		CMPA		* SEE IF IT IS LAST GRANULE
Ø526 Ø527	C3BF 25 24		ANDA	LC3E5	BRANCH IF THIS GRANULE IS < LAST GRANULE
Ø527	C3C1 84 3F C3C3 A1 44			FCBSEC,U	MASK OFF LAST GRANULE FLAG BITS * COMPARE LAST SECTOR USED IN GRANULE TO
Ø529	C3C5 26 1E			LC3E5	* CALCULATED SECTOR; BRANCH IF NOT EQUAL
Ø53Ø	C3C7 EC C8 13			FCBLST,U	GET NUMBER OF BYTES IN LAST SECTOR
Ø531	C3CA 84 7F		ANDA		MASK OFF PRESAVED FLAG (BIT 15)
Ø532	C3CC 34 Ø6		PSHS	B,A	SAVE NUMBER OF BYTES IN LAST SECTOR ON STACK
Ø533	C3CE 4F		CLRA		* LOAD ACCB WITH THE BYTE OFFSET TO CURRENT
Ø534 Ø535	C3CF E6 62 C3D1 E3 63			\$Ø2,S	* RECORD AND ADD THE REMAINING RECORD LENGTH
Ø536	C3D3 10 A3 E1		CMPD	\$Ø3,S S++	* TO IT - ACCD = END OF RECORD OFFSET =COMPARE THE END OF RECORD OFFSET TO THE NUMBER OF
Ø537	C3D6 23 ØD			LC3E5	=BYTES USED IN THE LAST SECTOR
Ø538	C3D8 ØD D8		TST		* CHECK GET/PUT FLAG AND BRANCH IF 'GET'
Ø539	C3DA 10 27 FF 56			LC334	* TO 'INPUT PAST END OF FILE' ERROR
Ø54Ø					
Ø541				SECTOR, CALCULATE HOW MANY BYTES	
Ø542	C2DE 04 81	* IF DAI		BEING 'PUT' PASTH THE CURRENT END	OF FILE
Ø543 Ø544	C3DE 84 Ø1 C3EØ 8A 8Ø		ANDA ORA		* SET PRE-SAVED FLAG BIT - ALL PUT RECORDS ARE
Ø545	SOLD ON OD	*	UNA		* WRITTEN TO DISK BEFORE LEAVING 'PUT'
Ø546	C3E2 ED C8 13		STD	FCBLST,U	SAVE NUMBER OF BYTES USED IN LAST SECTOR
Ø547	C3E5 35 Ø4	LC3E5	PULS	В	PULL BYTE OFFSET OFF OF THE STACK
Ø548	C3E7 30 C8 19			FCBCON,U	POINT X TO FCB DATA BUFFER
Ø549	C3EA 3A		ABX	##O C	MOVE X TO START OF RECORD
Ø55Ø Ø551	C3EB EE 62 C3ED 34 Ø4		LDU PSHS	\$Ø2,S	POINT U TO CURRENT POSITION IN RANDOM FILE BUFFER SAVE BYTE OFFSET ON STACK
Ø551	C3EF 86 FF		LDA		* CONVERT ACCD INTO A NEGATIVE 2 BYTE NUMBER
Ø553		*	LUN		* REPRESENTING THE REMAINING UNUSED BYTES IN THE SECTOR
Ø554	C3F1 E3 61		ADDD	\$Ø1,S	* ADD TEMPORARY RECORD LENGTH COUNTER (SUBTRACT
Ø555		*		-	* REMAINING BYTES FROM TEMPORARY RECORD LENGTH)
Ø556	C3F3 24 Ø7			LC3FC	BRANCH IF THERE ARE ENOUGH UNUSED BYTES TO FINISH THE RECORD
Ø557	C3F5 ED 61			\$Ø1,S	SAVE NEW TEMPORARY RECORD LENGTH COUNTER
Ø558	C3F7 35 Ø4		PULS	В	RESTORE BYTE COUNTER  * NECATE IT - ACCR - THE NUMBER OF RYTES
Ø559 Ø56Ø	C3F9 5Ø	*	NEGB		* NEGATE IT - ACCB = THE NUMBER OF BYTES  * AVAILABLE TO A RECORD IN THIS SECTOR
Ø561	C3FA 20 08		BRA	LC404	MOVE THE DATA
Ø562	== ==				·
Ø563		* BRANCH	HERE	IF REMAINING RECORD LENGTH WILL F	TIT IN
Ø564				OF THE CURRENTLY SELECTED SECTOR	
Ø565	C3FC E6 62	LC3FC		\$Ø2,S	GET REMAINING RECORD LENGTH
Ø566	C3FE 6F 61			\$Ø1,S	* CLEAR THE TEMPORARY RECORD LENGTH
Ø567 Ø568	C400 6F 62 C402 32 61			\$02,S \$01,S	* COUNTER ON THE STACK PURGE BYTE OFFSET FROM STACK
Ø569	C402 32 61 C404 96 D8	LC4Ø4	LDA		* CHECK GET/PUT FLAG AND
Ø57Ø	C4Ø6 27 Ø2	20 107		LC4ØA	* BRANCH IF GET
Ø571	C4Ø8 1E 13		EXG	X,U	SWAP SOURCE AND DESTINATION POINTERS
Ø572	C4ØA BD A5 9A	LC4ØA		LA59A	TRANSFER DATA FROM SOURCE TO DESTINATION BUFFERS
Ø573	C4ØD EF 62		STU	\$Ø2,S	SAVE NEW TEMP RECORD POINTER ON THE STACK (GET)
Ø574 Ø575		* MU/L D	ΔTΛ Γ	ROM FCB DATA BUFFER TO THE RANDOM	FILE RUFFER IF 'GFT'
				OOM FILE BUFFER TO FCB DATA BUFFER	
Ø576					

Ø581 Ø582 Ø583	C40F DE F1 C411 96 D8 C413 27 04 C415 A7 4F C417 AF 62 C419 AE 4D C418 30 01 C41B 30 01 C41D 5F C41E EE E4 C420 10 26 FE C4 C424 35 96	LDU FCBTMP LDA VDB BEQ LC419 STA FCBFLG,U STX \$02,S LC419 LDX FCBSOF,U LEAX \$01,X CLRB LDU ,S LBNE LC2EB PULS A,B,X,PC	POINT U TO FCB  * CHECK GET/PUT FLAG AND  * BRANCH IF GET  SAVE 'PUT' FLAG IN THE FCB  SAVE NEW TEMPORARY RECORD POINTER ON STACK (PUT)  * GET SECTOR OFFSET COUNTER AND  * ADD ONE TO IT  SET BYTE OFFSET = Ø  * CHECK THE LENGTH OF THE TEMPORARY RECORD LENGTH  * COUNTER AND KEEP MOVING DATA IF   * PULL TEMPORARY RECORD LENGTH AND  * BUFFER ADDRESS OFF STACK AND RETURN
0590 0591 0592 0593 0594 0596 0597 0598 0699 0600 0601 0602 0603 0604 0605 0606 0607 0608 0609 0610 0611 0611 0611 0613	C426 32 62 C428 BD B1 56 C428 BD B6 A4 C42E 34 Ø4 C42E 34 Ø5 C43Ø BD A5 A2 C433 5D C434 1Ø 2F E1 CB C438 35 Ø2 C43A 34 Ø6 C43C ØF 6F C44B BD B2 6D C441 8E C2 94 C444 BD C8 8A C447 CC Ø1 FF C44A FD Ø9 57 C44D 8E Ø1 ØØ C450 9D A5 C452 27 Ø8 C454 BD B2 6D C457 BD B3 E6 C457 BD B3 E6 C458 9E 52 C458 9E 52 C458 FØ 97 C C45F 1Ø 27 EF E7 C466 35 Ø6	* OPEN RAM HOOK DVECØ LEAS \$02,S	PULL RETURN ADDRESS OFF OF THE STACK EVALUATE AN EXPRESSION *GET MODE(I,O,R) - FIRST BYTE OF STRING EXPRESSION *AND SAVE IT ON STACK GET DEVICE NUMBER SET FLAGS
0616 0617 0618 0619 0620 0621 0622 0623 0624 0625 0626	C468 34 Ø2 C46A BD C7 19 C46D 1Ø 26 E1 AB C471 9F F1 C473 BD C7 6D C476 BD C6 5F C479 35 Ø4 C47B 86 1Ø C47D 34 Ø2 C47F C1 49 C481 26 1F	* OPEN DISK FILE FOR READ OR WRITE LC468 PSHS A JSR LC719 LENE LA61C STX FCBTMP JSR LC76D JSR LC65F PULS B LDA #INPFIL PSHS A CMPB #'1' BNE LC4A2	SAVE MODE ON STACK POINT X TO FCB FOR THIS FILE 'FILE ALREADY OPEN' ERROR IF FILE OPEN SAVE FILE BUFFER POINTER MAKE SURE FILE ALLOC TABLE IS VALID SCAN DIRECTORY FOR 'FILENAME.EXT' GET MODE INPUT TYPE FILE SAVE FILE TYPE ON STACK INPUT MODE? BRANCH IF NOT
0630 0631 0632 0633 0634 0635 0636 0637 0638 0640 0641 0642 0643 0644	C483 BD C6 BB C486 BD C7 D7 C489 BE Ø9 74 C48C EC ØB C481 BD C7 D7 C491 BD C5 FA C493 BD C5 FA C496 BD C7 25 C499 6C ØØ C498 9E F1 C490 35 Ø2 C497 A7 ØØ C441 39 C444 C1 4F C446 26 1A	* OPEN A SEQUENTIAL FILE FOR INPUT  JSR LC6B8  JSR LC707  LDX V974  LDD DIRTYP,X  STD DFLTYP  BSR LC5600  JSR LC5FA  LC496 JSR LC725  INC FAT0,X  LDX FCBTMP  PULS A  STA FCBTYP,X  RTS  LC442 ASL ,S  CMPB #'0'  BNE LC4C2	CHECK TO SEE IF DIRECTORY MATCH IS FOUND CHECK TO SEE IF FILE ALREADY OPEN GET RAM DIRECTORY BUFFER GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE INITIALIZE FILE BUFFER CONTROL BLOCK GO FILL DATA BUFFER POINT X TO PROPER FILE ALLOCATION TABLE ADD ONE TO FAT ACTIVE FILE COUNTER GET FILE BUFFER POINTER GET FILE TYPE SAVE IT IN FCB  SET FILE TYPE TO OUTPUT FILE MODE = OUTPUT? BRANCH IF NOT
0647 0648 0649 0650 0651 0652 0653 0654	C4A8 7D 09 73 C4AB 27 0F C4AD BD C6 CF C4B0 B6 09 73 C4B3 B7 09 77 C4B6 BE 09 74 C4B9 BF 09 78	* OPEN A SEQUENTIAL FILE FOR OUTPUT TST V973 BEQ LC48C JSR LC6CF LDA V973 STA V977 LDX V974 STX V978	DOES FILE EXIST ON DIRECTORY? BRANCH IF NOT KILL THE OLD FILE * GET DIRECTORY SECTOR NUMBER OF OLD FILE AND * SAVE IT AS FIRST FREE DIRECTORY ENTRY =GET RAM DIRECTORY IMAGE OF OLD FILE AND =SAVE IT AS FIRST FREE DIRECTORY ENTRY
Ø658 Ø659 Ø66Ø Ø661 Ø662	C4BC 8D 7C C4BE 8D 4B C4CØ 2Ø D4 C4CØ C1 52 C4C4 27 Ø6 C4C6 C1 44 C4C8 1Ø 26 E1 4A	LC4BC BSR LC53A BSR LC5BB BRA LC496 LC4C2 CMPB #'R' BEQ LC4CC CMPB #'D' LBNE LA616	SET UP NEW DIRECTORY ENTRY ON DISK INITIALIZE FILE BUFFER FLAG AND MAP FCB AS BEING USED FILE MODE = R (RANDOM)? BRANCH IF SO FILE MODE = D (DIRECT)? 'BAD FILE MODE' ERROR IF NOT
0665 0666 0667 0668 0669 0670 0671	C4CC 68 E4 C4CE FC 09 48 C4D1 34 06 C4D3 F3 09 7C C4D6 25 06 C4D8 10 B3 09 4A C4DC 23 05	* OPEN A RANDOM/DIRECT FILE LC4CC ASL ,S LDD RNBFAD PSHS B,A ADDD DFFLEN BLO LC4DE CMPD FCBADR BLS LC4E3	SET FILE TYPE TO DIRECT  * GET ADDRESS OF RANDOM FILE BUFFER AREA  * AND SAVE IT ON THE STACK ADD THE RECORD LENGTH  'OB' ERROR IF SUM > \$FFFF IS IT > THAN FCB DATA AREA? BRANCH IF NOT

Ø673	C4DE C6 3A	LC4DE	LDB #	#2*29	'OUT OF BUFFER SPACE' ERROR
Ø674	C4EØ 7E AC 46		JMP I	LAC46	JUMP TO ERROR HANDLER
Ø675	C4E3 34 Ø6	LC4E3	PSHS I	В,А	SAVE END OF RANDOM BUFFER ON STACK
Ø676	C4E5 7D Ø9 73		TST		DID THIS FILE EXIST
Ø677	C4E8 26 Ø2		BNE	LC4EC	BRANCH IF SO
Ø678	C4EA 8D 4E		BSR		SET UP NEW FILE IN DIRECTORY
Ø679	C4EC 8D 12	LC4EC	BSR	LC500	INITIALIZE FILE BUFFER
Ø68Ø	C4EE 63 ØD		COM	FCBSOF,X	* SET FCBSOF,X TO \$FF (ILLEGAL SECTOR OFFSET) WHICH WILL
Ø681		*			* FORCE NEW SECTOR DATA TO BE READ IN DURING GET/PUT
Ø682	C4FØ 6C Ø8		INC		INITIALIZE RECORD NUMBER = 1
Ø683	C4F2 35 46		PULS	A.B.U	U = START OF RANDOM FILE BUFFER AREA, ACCD = END
Ø684	C4F4 FD Ø9 48				SAVE NEW START OF RANDOM FILE BUFFER AREA
Ø685	C4F7 EF ØB				SAVE BUFFER START IN FCB
	C4F9 FE Ø9 7C		LDU I		* GET RANDOM FILE RECORD LENGTH
Ø687	C4FC EF Ø9			FCBRLN,X	* AND SAVE IT IN FCB
Ø688	C4FE 2Ø 96		BRA		SET FAT FLAG, SAVE FILE TYPE IN FCB
Ø689					*** ····· · = ·**, *··· = · · · · = · · · · *-
Ø69Ø		* INITIA	LIZE FO	CB DATA FOR INPUT	
Ø691	C500 8D 09	LC5ØØ			INITIALIZE FCB
Ø692	C502 FE 09 74		LDU		GET RAM DIRECTORY IMAGE
Ø693	C5Ø5 EE 4E			DIRLST,U	*GET NUMBER OF BYTES IN LAST SECTOR OF FILE
Ø694	C507 EF 88 13			FCBLST, X	*SAVE IT IN FCB
Ø695	C5ØA 39		RTS	,	
Ø696		* INITIA		ILE CONTROL BLOCK	
Ø697	C5ØB 9E F1	LC5ØB			GET CURRENT FILE BUFFER
Ø698	C5ØD C6 19			#FCBCON	CLEAR FCB CONTROL BYTES
Ø699	C5ØF 6F 8Ø	LC5ØF	CLR		CLEAR A BYTE
0700	C511 5A		DECB		DECREMENT COUNTER
	C512 26 FB			LC5ØF	BRANCH IF NOT DONE
0702	C514 9E F1			FCBTMP	GET CURRENT FILE BUFFER ADDRESS BACK
0703	C516 96 EB		LDA I		*GET CURRENT DRIVE NUMBER AND
0704	C518 A7 Ø1				*SAVE IT IN FCB
	C51A B6 Ø9 76		LDA 1		=GET FIRST GRANULE -
	C51D A7 Ø2				=SAVE IT AS THE STARTING GRANULE NUMBER AND
	C51F A7 Ø3				=SAVE IT AS CURRENT GRANULE NUMBER
	C521 F6 Ø9 73		LDB 1		GET DIRECTORY SECTOR NUMBER
0709	C524 CØ Ø3		SUBB #		SUBTRACT 3 - DIRECTORY SECTORS START AT 3
0710	C526 58		ASLB		* MULTIPLY SECTORS
	C527 58		ASLB		* BY 8 (8 DIRECTORY
	C528 58		ASLB		* ENTRIES PER SECTOR)
	C529 34 Ø4		PSHS I	B	SAVE SECTOR OFFSET
	C52B FC Ø9 74		LDD 1		GET RAM DIRECTORY IMAGE
	C52E 83 Ø6 ØØ				SUBTRACT RAM OFFSET
Ø716	C531 86 Ø8		LDA #		8 DIRECTORY ENTRIES/SECTOR
Ø717	C533 3D		MUL	7400	NOW ACCA CONTAINS Ø-7
Ø718	C534 AB EØ		ADDA	+2	ACCA CONTAINS DIRECTORY ENTRY (0-71)
Ø719	C536 A7 88 12			FCBDIR,X	SAVE DIRECTORY ENTRY NUMBER
0720	C539 39			. 055111,11	ONTE BINESTON ENTRY NONDEN
			RTS		
0721		* SFT IIP		TORY AND UPDATE FILE ALLOCATION T	ARIF FATRY IN FIRST HANGED SECTOR
Ø721 Ø722	C53A C6 38	* SET UP	DIRECT		ABLE ENTRY IN FIRST UNUSED SECTOR
Ø721 Ø722 Ø723	C53A C6 38	* SET UP	DIRECT	#28*2	'DISK FULL' ERROR
0721 0722 0723 0724	C53C B6 Ø9 77		DIRECT	#28*2 V977	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY
0721 0722 0723 0724 0725	C53C B6 Ø9 77 C53F 1Ø 27 E7 Ø3		DIRECT LDB # LDA !	#28*2 V977 LAC46	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRIES
0721 0722 0723 0724 0725 0726	C53C B6 Ø9 77 C53F 1Ø 27 E7 Ø3 C543 B7 Ø9 73		DIRECT LDB # LDA ! LBEQ !	#28*2 V977 LAC46 V973	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRIES SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY
0721 0722 0723 0724 0725 0726 0727	C53C B6 Ø9 77 C53F 1Ø 27 E7 Ø3 C543 B7 Ø9 73 C546 97 ED		DIRECT LDB # LDA ! LBEQ ! STA !	#28*2 V977 LAC46 V973 DSEC	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRIES SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER
0721 0722 0723 0724 0725 0726 0727 0728	C53C B6 Ø9 77 C53F 1Ø 27 E7 Ø3 C543 B7 Ø9 73 C546 97 ED C548 C6 Ø2		DIRECT LDB # LDA ! LBEQ ! STA ! STA ! LDB #	#28*2 1977 LAC46 1973 DSEC #\$02	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRIES SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE
0721 0722 0723 0724 0725 0726 0727 0728 0729	C53C B6 09 77 C53F 10 27 E7 03 C543 B7 09 73 C546 97 ED C548 C6 02 C54A D7 EA		DIRECT LDB # LDA ! LBEQ   STA ! STA ! LDB #	#28*2 V977 LAC46 V973 DSEC #\$Ø2 DCOPC	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRIES SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730	C53C B6 Ø9 77 C53F 1Ø 27 E7 Ø3 C543 B7 Ø9 73 C546 97 ED C548 C6 Ø2 C54A D7 EA C54C BD D5 FF		DIRECT LDB # LDA ! LBEQ   STA ! STA ! LDB # STB !	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRIES SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731	C53C B6 09 77 C53F 10 27 E7 03 C543 B7 09 73 C546 97 ED C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78		DIRECT LDB # LDA ! LBEQ   STA ! STA ! LDB # STB ! JSR ! LDX !	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRES SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR ** GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731 0732	C53C B6 89 77 C53F 10 27 E7 03 C543 B7 09 73 C546 97 ED C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C552 BF 09 74		DIRECT LDB # LDA   LBEQ   STA   STA   LDB # STB   JSR   LDX   STX	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRYS SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731 0732 0733	C53C B6 09 77 C53F 10 27 E7 03 C543 B7 09 73 C546 97 ED C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C552 BF 09 74 C555 33 84		DIRECT LDB # LDA V LBEQ STA V STA I LDB # STB I JSR LDX V STX V LEAU	#28*2 V977 V973 DSEC #\$Ø2 DCOPC LD5FF V978 V974	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731 0732 0733	C53C B6 09 77 C53F 10 27 E7 03 C543 B7 09 73 C546 97 ED C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C552 BF 09 74 C555 33 84 C557 C6 20		DIRECT LDB # LDA   LBEQ   STA   LDB # LDB   LDB	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1978 1974 7, X #DIRLEN	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY)
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731 0732 0733	C53C B6 09 77 C53F 10 27 E7 03 C543 B7 09 73 C546 97 ED C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C552 BF 09 74 C555 33 84		DIRECT LDB # LDA V LBEQ STA V STA I LDB # STB I JSR LDX V STX V LEAU	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1978 1974 7, X #DIRLEN	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731 0732 0733 0734	C53C B6 09 77 C53F 10 27 E7 03 C548 97 ED C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C552 BF 09 74 C555 33 84 C557 C6 29 C559 6F 80 C558 5A		DIRECT LDB # LDA   LBEQ   STA   LDB # STB   LDB # STB   LDS   LDS   LDS   LDX   LDX   LDB   LDB   CLR   CLR   CLR   LDB   CLR   LDB   CLR   LDB   LDB	#28*2 V977 V977 LAC46 V973 DSEC #\$02 UD5FF V978 V974 ,X #DIRLEN ,X+	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0732 0733 0734 0735 0736	C53C B6 09 77 C53F 10 27 E7 03 C548 97 ED C548 C6 02 C54A D7 EA C54C D7 EA C54C BD D5 FF C54F BE 09 78 C55S BF 09 74 C555 33 84 C557 C6 20 C558 5A C558 5A		D DIRECT LDB = 1 LDA   1 LBEQ   1 STA   1 LDB = 2 STB   1 JSR   1 LDX   1 LDX	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1978 1974 , X #DIRLEN , X+	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRYSAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR ** ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731 0732 0733 0734 0735	C53C B6 09 77 C53F 10 27 E7 03 C548 97 ED C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C552 BF 09 74 C555 33 84 C557 C6 29 C559 6F 80 C558 5A		D DIRECT LDB = 1 LDA   1 LBEQ   1 STA   1 LDB = 2 STB   1 JSR   1 LDX   1 LDX	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V974 ,X #DIRLEN ,X+ LC559 #DNAMBF	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0732 0733 0734 0735 0737	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 74 C555 33 84 C555 36 80 C559 6F 80 C558 5A C55C 26 FB C55E 8 69 4C		DIRECT LDB = LDA   LDB = LDA   LDB = LDB = LDB   LDB =	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V974 ,X #DIRLEN ,X+ LC559 #DNAMBF #11	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE
0721 0722 0723 0724 0725 0726 0727 0728 0729 0731 0732 0733 0735 0736 0737	C53C B6 09 77 C53F 10 27 E7 03 C548 97 ED C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C552 BF 09 74 C555 33 84 C557 C6 28 C559 6F 80 C558 5A C55C 26 FB C55E BE 09 4C C551 C6 08		DIRECT LDB # LDBA # LBEQ   STA   STA   LDB # STB   JSR   LDX   LDX   LDX   LDB # LDX   LDB # LDX   LDB # LDX #	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V974 ,X #DIRLEN ,X+ LC559 #DNAMBF #11	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731 0732 0733 0734 0735 0736 0737	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 89 78 C555 BF 89 74 C555 33 84 C557 C6 29 C559 6F 80 C558 5A C55C BF 89 4C C55C BF 89 4C C55C BF 89 4C C55C BF 89 4C		DIRECT LDB # LDBA # LBEQ   STA   STA   LDB # STB   JSR   LDX   LDX   LDX   LDB # LDX   LDB # LDX   LDB # LDX #	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V978 V974 ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U
0721 0722 0723 0724 0725 0726 0727 0728 0729 0730 0731 0732 0733 0734 0735 0737 0738 0737	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C555 33 84 C555 C6 20 C559 6F 80 C558 5A C55C 26 FB C55E 8B 09 4C C561 C6 08 C563 BD A5 9A		DIRECT LDB # LDB # LBEQ   STA   STA   LDB # STB   JSR   LDX   LDB # LEAU LDB # LEAU LDB # LEAU LDB # LDX # L	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V978 V974 , X #DIRLEN ,X+ LC559 #DNAMBF ##11 LA59A DFLTYP ,U	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0732 0733 0734 0735 0736 0737 0738	C53C B6 Ø9 77 C53F 10 27 E7 Ø3 C548 97 ED C548 C6 Ø2 C54A D7 EA C54C BD D5 FF C54F BE Ø9 78 C552 BF Ø9 74 C555 33 84 C557 C6 20 C559 6F 80 C558 5A C55C 26 FB C55E BE Ø9 4C C561 C6 ØB C563 BD A5 9A C566 FC Ø9 57		DIRECT LDB =	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1978 1974 ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U #33	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OF CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0732 0733 0734 0736 0737 0738 0737 0740 0741 0742	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C555 BF 09 74 C555 33 84 C556 C6 20 C559 6F 80 C558 BA C55C BF 09 4C C55C C6 FB C55C BF 09 57 C56B C6 D 40 C56B C6 21		DIRECT LDB # LDBA # LBEQ   STA   STA   LDB # LDBA # LDBA # LDBA # LDDA # LDBA #	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V974 ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0732 0733 0734 0735 0736 0737 0738 0739 0741 0742	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C555 BF 09 74 C555 33 84 C557 C6 20 C559 6F 80 C558 BA C55C 26 FB C55E BE 09 4C C561 C6 08 C56B BD A5 9A C566 FC 09 57 C566 EC 09 57 C568 EC 40 C568 EC 21 C568 EC 21 C568 BD C7 8F		DIRECT LDB =	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V974 ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC78F	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0732 0733 0734 0735 0736 0737 0738 0739 0740 0741 0742	C53C B6 09 77 C53F 10 27 E7 03 C548 97 ED C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 78 C552 BF 09 74 C555 33 84 C557 C6 20 C559 6F 80 C558 5A C55C 26 FB C55E BE 09 4C C561 C6 08 C563 B0 A5 9A C566 FC 09 57 C569 ED 40 C568 C6 21 C560 BD C7 8F		DIRECT LDB ; LDA ; LDB ; STA ; STA ; LDB ; STB ; JSR ; LDX ; LEAU LDB ; LEAU LDB ; LDX ; LEAU LDB ; LDX ; LDD ; LDB ; LDX ; LDB ; LDX ; LDB ; LDX ; LDB ; LDX ; LDB ; LD	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1978 1974 ,X #DIRLEN ,X+ #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC78F 1976 1862,U	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM
9721 9722 9723 9724 9725 9726 9727 9738 9731 9731 9732 9733 9734 9735 9736 9737 9749 9741 9742 9743 9744	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 74 C555 33 84 C557 62 20 C558 B6 B0 C558 B7 C55C BF B0 C56C BF B0 C57C BF B7 C57C BF B7 C57C BF B7 C57C B7 C77C B7 C57C B7 C77C B7 C77		DIRECT LDB = LDA = LDB =	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V974 ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC78F V976 \$902,U	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE OF DIRECTORY TRACK
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0732 0733 0735 0736 0737 0738 0739 0741 0742 0744 0745 0744	C53C B6 Ø9 77 C53F 10 27 E7 Ø3 C546 97 E0 C548 C6 Ø2 C54A D7 EA C54C BD D5 FF C54F BE Ø9 78 C555 BF Ø9 74 C555 33 84 C557 C6 20 C559 6F 80 C558 BA C55C 26 FB C55E 8B Ø9 4C C561 C6 Ø8 C56B C6 20 C56B C6 21 C56B C6 21 C56B C6 21 C56B B0 78 C56B B0 79 76 C56B B0 79 76 C573 A7 42 C575 C6 Ø3		DIRECT LDB ; LDA ; LDB ; LDB ; STA   LDB ; STA   JSR   JSR   LDX ; LEAU LDB ; LEAU LDB ; LDD ; LDB ; L	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V974 ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC78F V976 \$02,U	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR ** CHAPTER OF THE SECTION OF THE SECTION OF THE SECTORY  ** ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM MAGE OF DIRECTORY TRACK  ** GET WRITE OPERATION CODE AND SAVE  ** IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0732 0733 0734 0735 0736 0737 0738 0740 0741 0742 0743 0744 0744 0744 0744 0744	C53C B6 Ø9 77 C53F 10 27 E7 Ø3 C546 97 ED C548 C6 Ø2 C54A D7 EA C54C BD D5 FF C54F BE Ø9 78 C555 33 84 C557 C6 20 C559 6F 80 C55B 5A C55C 26 FB C55E 8E Ø9 4C C561 C6 Ø8 C563 B0 A5 9A C566 FC Ø9 57 C569 ED 40 C568 BC 21 C560 BC C21 C560 BC C7 C560 BC 70 8F C563 BC A5 C564 C6 Ø8 C565 BC C6 70 C566 BC C7 C560 BC 70 C567 B7 Ø9 76 C573 A7 42 C575 C6 Ø3 C577 D7 EA		DIRECT LDB ; LDA ; LDA ; LDB ; STA   STA   JSR   JSR   LDX ; LEAU   LDB ; LEAU   LDB ; LDD ; LDB	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V978 V974 , X #DIRLEN , X #DIRLEN , X+ LC559 #DNAMBF #11 LA59A DFLTYP , U #33 LC78F V976 V976 V976 V976 V976 V976 V976 V976	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OF CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM MAGE OF DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER
9721 9722 9723 9724 9725 9726 9727 9730 9731 9732 9733 9734 9735 9736 9737 9749 9741 9742 9743 9744 9744 9745	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C548 C7 E7 E9 C546 B0 D5 FF C552 BF 09 74 C555 33 84 C555 B6 80 C558 B7 C556 B7 C556 B7 C556 B7 C566 C6 09 C568 C6 21 C568 C6 21 C567 C6 03 C577 D7 EA C578 C6 03 C579 B7 09 76	LC559	DIRECT LDB	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V978 V974 , X #DIRLEN , X #DIRLEN , X+ LC559 #DNAMBF #11 LA59A DFLTYP , U #33 LC78F V976 V976 V976 V976 V976 V976 V976 V976	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR ** CHAPTER OF THE SECTION OF THE SECTION OF THE SECTORY  ** ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM MAGE OF DIRECTORY TRACK  ** GET WRITE OPERATION CODE AND SAVE  ** IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0732 0733 0734 0735 0736 0737 0738 0741 0742 0744 0745 0744 0745 0747	C53C B6 Ø9 77 C53F 10 27 E7 Ø3 C546 97 ED C548 C6 Ø2 C54A D7 EA C54C BD D5 FF C54F BE Ø9 78 C555 BF Ø9 74 C555 33 84 C557 C6 20 C559 6F 80 C558 BA C55C 26 FB C55E 8B Ø9 4C C561 C6 Ø8 C55B BA 94 C566 FC Ø9 57 C569 ED 40 C566 FC Ø9 57 C569 BD C7 8F C573 A7 42 C575 D7 EA C579 DD D5 FF C577 D7 EA C579 BD D5 FF	LC559	DIRECT LDB ; LDA ; LDA ; LDB ; STA   STA   JSR   JSR   LDX ; LEAU   LDB ; LEAU   LDB ; LDD ; LDB	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1978 1974  ##01RLEN ,X ##DIRLEN ,X+ LC559 ##0NAMBF ##11 LA59A DFLTYP ,U ##33 LC78F 1976 1982,U ##000PC LD5FF U,X,B,A LC725	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM IMAGE OF DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE SAVE ENTERS
9721 9722 9723 9724 9725 9726 9727 9728 9730 9731 9732 9733 9734 9735 9736 9737 9748 9749 9749 9744 9744 9744 9744 974	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C B0 D5 FF C54F BE 89 78 C5552 BF 89 74 C555 33 84 C555 66 88 C55B 5A C55C BF 80 C55B 8E 89 4C C55C BF 80 C55B 8E 89 4C C56G E6 80 C56B C6 88 C56B C6 88 C56B C6 88 C56B C7 8F C56B C7 8F C578 B7 89 76 C573 A7 42 C573 A7 42 C575 C6 83 C577 D7 EA C576 BD C7 25	LC559	DIRECT LDB	#28*2 1937 LAC46 1937 DSEC #\$02 DCOPC LD5FF 1937 #01RLEN ,X #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X, #DIRLE	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X, U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM MAGE FIRST GRANULE TO CHECK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE
9721 9722 9723 9724 9726 9727 9728 9730 9731 9731 9733 9736 9736 9736 9737 9749 9741 9742 9743 9744 9744 9745 9746 9747 9748 9750	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C548 C7 E7 E9 C546 B0 D5 FF C555 BE 09 74 C555 33 84 C556 B6 B0 C558 B7 C556 B6 B0 C556 B7 C566 C6 09 57 C568 B7 C570 B7 07 EA C577 D7 EA C577 B7 C577 B7 C577 B7 C577 B7 C577 B7 C577 B7 C578 B0 D5 C578 B7 C579 B0 D5 C578 B7 C579 B0 D5 C578 C6 C579 B0 D5 C576 C7 C579 B0 D5 C576 C7 C577 C7 C	LC559	DIRECT LDB ; LDA ; LDA ; LDA ; STA ; STA ; JSR ; JSR ; LDX ; LEAU ; LDB	#28*2 1937 LAC46 1937 DSEC #\$02 DCOPC LD5FF 1937 #01RLEN ,X #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X, #DIRLE	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM SAVE IN RAM IMAGE OF DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS FOINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO
0721 0722 0723 0724 0725 0726 0727 0728 0730 0731 0733 0734 0735 0736 0737 0738 0741 0742 0744 0745 0747 0748 0747 0748 0747 0748 0751	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 74 C555 33 84 C557 C6 20 C558 5A C55C 26 FB C55E 8B 09 4C C561 C6 08 C56B C6 21 C573 A7 42 C575 D7 EA C579 DD D5 FF C577 D7 EA C579 DD FF C577 D7 EA C579 BD D5 FF C576 BB C7 25 C581 6C 01 C583 A6 01	LC559	DIRECT LDB ; LDA ; LDA ; LDA ; STA ; STA ; JSR ; JSR ; LDX ; LEAU ; LDB	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V978 V974 , X #DIRLEN , X #DIRLEN , X+ LC559 #DNAMBF #11 LA59A DFLTYP , U #33 LC72F V976 V976 V976 V976 V976 V976 V976 V976	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM SAVE IN RAM IMAGE OF DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS FOINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO
9721 9722 9723 9724 9725 9726 9727 9728 9730 9731 9732 9733 9734 9735 9736 9749 9740 9741 9742 9743 9744 9745 9746 9747 9748 9749 9750 9751 9752 9753	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 09 74 C555 33 84 C557 C6 20 C558 5A C55C 26 FB C55E 8B 09 4C C561 C6 08 C56B C6 21 C573 A7 42 C575 D7 EA C579 DD D5 FF C577 D7 EA C579 DD FF C577 D7 EA C579 BD D5 FF C576 BB C7 25 C581 6C 01 C583 A6 01	LC559	DIRECT LDB : LDA : LDA : STA : STA : STA : LDB : JSR : LDX : STX : LEAU LDB : LDX : STX : LEAU LDB : LDB : JSR : LDB : L	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1974 ,X #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X, #D	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM MAGE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE ROSCANDERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA FLAG  SOUR MATE HAS ALLOCATION TABLE INDICATE NEW DATA FLAG  SOUR SECTION OF TABLE  FIND THE SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA FLAG  GET NEW DATA FLAG
0721 0722 0723 0724 0726 0726 0727 0728 0730 0731 0733 0734 0735 0736 0737 0738 0741 0742 0744 0745 0747 0748 0747 0748 0747 0751 0753 0754	C53C B6 89 77 C53F 10 27 E7 03 C546 97 ED C548 C6 02 C548 D7 EA C54C BD D5 FF C55F BE 09 74 C555 33 84 C557 C6 20 C558 5A C55C 26 FB C55E B 09 4C C561 C6 08 C56B C6 21 C56B C6 21 C56B C6 21 C56B C6 21 C56B C7 87 C573 A7 42 C573 A7 42 C575 D7 EA C577 D7 EA C579 BD D5 FF C571 BD C7 25 C581 6C 01 C583 A6 01 C588 25 03	LC559	DIRECT LDB : LDB : LDB : LDB : STA   STA   JSR   JSR   LDX : LEAU LDB :	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V978 V974 ,X #DIRLEN ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC72F V976 V976 V976 V976 V976 V976 V9776 V9777 V97	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OF CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG  * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE
9721 9722 9723 9724 9725 9726 9727 9738 9730 9731 9732 9733 9734 9735 9749 9741 9742 9743 9744 9745 9746 9747 9748 9759 9759 9759 9759	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C B0 D5 FF C54F BE 99 78 C555 BF 09 74 C555 33 84 C556 B6 B0 C55B BA C556 B7 B9 C556 B6 B0 C556 B7 C566 B7 C566 B7 C567 B7 C568 B7 C568 B7 C568 B7 C569 B7 C569 B7 C569 B7 C569 B7 C569 B7 C569 B7 C570 B7 C571 D7 C572 B7 C573 B7 C574 B7 C575 B7 C576 B7 C577 B7 C578 B7 C588 B8 C6 F1	LC559 LC57C	DIRECT LDB : LDA : LDB : LDA : LDB : LDB : STA   LDB : STA   LDB : JSR   LDD : LDB :	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1978 1974  ##01RLEN ,X ##DIRLEN ,X+  LC559 ##0NAMBF ##11 LA59A DFLTYP ,U ##33 LC78F 19976 100PC LC78F 1976 1982,U ##503 100PC LC75F FAT1, X FAT1,	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK * SET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK
0721 0722 0723 0724 0726 0726 0727 0728 0730 0731 0731 0735 0736 0737 0738 0736 0737 0744 0741 0742 0743 0744 0745 0747 0748 0750 0751 0755 0755 0756 0757	C53C B6 89 77 C53F 10 27 E7 03 C546 97 ED C548 C6 02 C548 D7 EA C54C BD D5 FF C55F BE 09 74 C555 33 84 C557 C6 20 C558 5A C55C 26 FB C55E B 09 4C C561 C6 08 C56B C6 21 C56B C6 21 C56B C6 21 C56B C6 21 C56B C7 87 C573 A7 42 C573 A7 42 C575 D7 EA C577 D7 EA C579 BD D5 FF C571 BD C7 25 C581 6C 01 C583 A6 01 C588 25 03	LC559	DIRECT LDB : LDA : LDB : LDA : LDB : LDB : STA   LDB : STA   LDB : JSR   LDD : LDB :	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V978 V974 ,X #DIRLEN ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC72F V976 V976 V976 V976 V976 V976 V9776 V9777 V97	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OF CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG  * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE
0721 0722 0723 0724 0726 0726 0727 0728 0730 0731 0733 0734 0735 0736 0737 0738 0741 0742 0744 0745 0744 0747 0748 0747 0751 0753 0754 0755 0756 0757	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C B0 D5 FF C54F BE 99 78 C555 BF 09 74 C555 33 84 C556 B6 B0 C55B BA C556 B7 B9 C556 B6 B0 C556 B7 C566 B7 C566 B7 C567 B7 C568 B7 C568 B7 C568 B7 C569 B7 C569 B7 C569 B7 C569 B7 C569 B7 C569 B7 C570 B7 C571 D7 C572 B7 C573 B7 C574 B7 C575 B7 C576 B7 C577 B7 C578 B7 C588 B8 C6 F1	LC559  LC57C  * LC58D	DIRECT LDB : LDA : LDB : LDB : STA   STA   JSR   JSR   LDX : LDB : JSR   LDD : LDB : JSR   LDD : LDB : JSR   LDB : JSR   LDB : JSR   LDB : JSR   JSR   LDB : JSR   JSR   LDB : JSR	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V978 V974 , X ##DIRLEN ,X+ LC559 ##DNAMBF ##11 LA59A DFLTYP ,U #33 LC78F V976 V976 V976 V976 V976 V977  KPATYL LC55B LC6F1 A,B,X,U,PC	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK * SET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK
9721 9722 9723 9724 9725 9726 9727 9738 9730 9730 9733 9735 9736 9749 9741 9742 9743 9744 9745 9745 9746 9747 9759 9759 9759 9759 9759 9759 9759	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C BD D5 FF C54F BE 99 78 C555 BF 09 74 C555 33 84 C556 B6 B0 C55B BA C556 B7 B9 C556 B6 B0 C558 B7 C556 B7 C566 B0 C57 D7 C566 B0 C57 B7 C569 B0 C57 B7 C569 B0 C57 B7 C569 B0 C57 B7 C57 B7 C57 B8 C57 B7 C57 B8 C57 B7 C57 B8 C57 B8 C57 B7 C57 B8 C58 B	LC559  LC57C  * LC58D  * CONSOL	DIRECT LDB : LDA : LDA : LDB : STA : LDB : STA : LDB : JSR : LDD : STX : LEAU LDB : LDX : LDB :	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1978 1974 ,X #DIRLEN ,X+ #DIRLEN ,X+  LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC78F 1976 \$02,U #\$03 DCOPC LD5FF U,X,B,A LC725 FAT1,X FAT1,X FAT1,X MFATVL LC58D LC6F1 A,B,X,U,PC	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK * SET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RESTORE REGISTERS
0721 0722 0723 0724 0726 0726 0727 0728 0730 0731 0732 0733 0735 0736 0737 0738 0736 0737 0740 0741 0742 0743 0744 0745 0747 0745 0750 0751 0755 0756 0757	C53C B6 Ø9 77 C53F 10 27 E7 Ø3 C546 97 ED C548 C6 Ø2 C548 D7 EA C54C BD D5 FF C54F BE Ø9 74 C555 33 84 C557 C6 20 C558 5A C55C BF B0 40 C56B C6 E21 C56B C7 B7 C573 A7 42 C573 A7 42 C575 C6 Ø3 C577 D7 EA C579 BD D5 FF C57C C34 56 C578 BD C7 25 C581 6C Ø1 C583 A6 Ø1 C583 A6 Ø1 C588 A6 Ø1 C588 B D C6 C588 C6 F1 C580 B D C7 C580 B D C7 C581 C6 Ø1 C583 A6 Ø1 C583 A6 Ø1 C584 BD C6 C586 C6 F1 C588 B D C6 C588 B D C6 C588 B D C6 C588 B D C6 C588 B D C7 C588 B D C6 C588 B C6	LC559  LC57C  * LC58D  * CONSOL	DIRECT LDB : LDA : LDA : LDA : LDB : LDA : LDB :	#28*2 1937 1AC46 1937 1DSEC #\$02 DCOPC LD5FF 1937 ##01RLEN ##01RLE	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR ** GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY ** ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM IMAGE ** GET WRITE OPERATION CODE AND SAVE ** IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA FLAG ** HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO ** CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RESTORE REGISTERS  GET DEVICE NUMBER
0721 0722 0723 0724 0726 0726 0727 0728 0730 0731 0733 0734 0733 0734 0736 0737 0738 0741 0742 0743 0744 0745 0747 0748 0747 0751 0753 0754 0755 0757 0758 0759 0759 0760 0761	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C548 D7 EA C54C BD D5 FF C55F BE 09 74 C555 33 84 C55C 26 FB C55B 5A C55C 26 FB C55B 5A C55C 26 FB C55B BA C55C 26 GB C55B BA C55C 26 GB C55B BC C56B C6 21 C56B BC C56B C6 21 C56B BC C56B C6 21 C56B BC C573 A7 42 C578 BC C577 BC C573 A7 42 C578 BC C578 BC C578 BC C578 BC C579 BD D5 FF C570 A3 56 C576 BC C578 BC C578 BC C578 BC C579 BD D5 FF C570 A3 56 C576 BC C578 BC C578 BC C579 BD C7 25 C581 6C 01 C583 A6 01 C588 B1 09 7A C588 25 03 C587 BC C588 BC	LC559  LC57C  * LC58D  * CONSOL	DIRECT LDB :	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V974 ,X #DIRLEN ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC78F V976 S02,U #360 LC78F V976 FAT1,X FAT1,X FAT1,X WFATVL LC58D LC6F1 A,B,X,U,PC AMM VECTOR DEVNUM EVECA	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OF CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE SAVE IN RAM IMAGE FIRST BRANULE TO CHECK * SOFT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY TRACK * SET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG  * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RESTORE REGISTERS  GET DEVICE NUMBER BRANCH IF NOT DISK FILE
9721 9722 9723 9724 9726 9727 9728 9730 9730 9731 9732 9733 9736 9736 9741 9742 9744 9744 9744 9745 9749 9750 9750 9750 9750 9750 9755 9756 9757 9758 9759 9759 9759 9759 9759 9759	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C548 C7 E7 E9 78 C546 B0 D5 FF C552 BF 09 74 C555 33 84 C556 B6 B0 C58B BA C556 B6 B0 C58B BA C566 FC 09 57 C569 ED 40 C570 B7 09 76 C570 C7 B7 C570 B7 09 57 C570 B7 09 57 C570 B7 09 76 C570 B7 09 57 C570 B7 09 57 C570 B7 09 76 C573 A7 42 C575 C6 03 C579 BD D5 FF C570 B7 09 76 C570 B7 09 76 C573 A7 42 C575 C6 03 C570 D7 EA C570 B7 09 76 C570 B7 09 76 C571 A9 56 C572 B7 09 76 C573 A7 42 C575 C6 03 C576 D7 EA C579 BD D5 FF C570 B7 09 76 C571 A9 56 C572 B7 C7 C7 C588 B1 09 7A	LC559  LC57C  *  LC58D  * CONSOL  DVEC4	DIRECT LDB : LDA : LDA : LDB : STA : STA : LDB : JSR : LDX : LDX : LDB : JSR : LDX : LDB : JSR : LDD : LDB : JSR : LDB : LDB : JSR : LDB : LDB : JSR : LDB :	#28*2 1977 LAC46 1973 DSEC #\$02 DCOPC LD5FF 1978 1974 ,X #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #11 LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC78F 1976 \$02,U #\$03 DCOPC LD5FF U,X,B,A LC725 FAT1,X FAT1,X FAT1,X FAT1,X FAT1,X MFATVL LC58D LC6F1 A,B,X,U,PC AM VECTOR DEVNUM XVEC4 \$02,S	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRYS SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM IMAGE OF DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RESTORE REGISTERS  GET DEVICE NUMBER BRANCH IF NOT DISK FILE GET RID OF RETURN ADDRESS
0721 0722 0723 0724 0726 0726 0727 0728 0730 0731 0733 0733 0733 0736 0737 0738 0739 0741 0742 0743 0744 0745 0747 0748 0751 0755 0756 0757 0758 0757 0758 0759 0761 0762 0762	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C548 D7 EA C54C BD D5 FF C54F BE 09 74 C555 33 84 C557 C6 20 C558 5A C55C 26 FB C558 E0 94 C561 C6 08 C568 B7 C568 B7 C57 B7 C58 B7	LC559  LC57C  * LC58D  * CONSOL	D DIRECT LDB : LDA : LDA : LDB : STA : STA : LDB : JSR : LDX : LDB : JSR : LDX : LDB : JSR : LDB : JSR : LDB : JSR : LDB	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V978 V974 , X #DIRLEN , X #DIRLEN , X #DIRLEN , X+ LC559 #DNAMBF #11 LA59A DFLTYP , U #33 LC72F S92, U #\$03 DCOPC LD5FF U, X, B, A LC72F FAT1, X FAT1, X WFATVL LC58D LC6FI A, B, X, U, PC AM VECTOR DEVNUM XVEC4 \$\$02, S X, B	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM IMAGE OF DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA FLAG ET NEW DATA ETTENTORY ET TO THE TOTAL ENTRY ETTENT OF THE TOTAL ENTRY ETTENTY ETTENTY
9721 9722 9723 9724 9725 9726 9727 9730 9730 9731 9732 9733 9734 9735 9736 9749 9741 9741 9742 9743 9744 9745 9746 9747 9755 9756 9757 9758 9759 9759 9769 9769 9769 9769 9769 9769	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C54A D7 EA C54C B0 D5 FF C54F BE 99 78 C555 33 84 C555 33 84 C556 B6 80 C55B 5A C55C B7 80 C55B 8E 99 4C C561 C6 08 C563 B0 A5 9A C566 FC 09 C570 A7 C570 A7 C570 A7 C570 A7 C570 A7 C571 A7 C571 A7 C571 A7 C571 A7 C572 A7 C572 A7 C573 A7 C573 A7 C574 A7 C575 A8 C576 B0 C7 C577 D7 C576 B0 C7 C577 D7 C577 D7 C577 D7 C577 D7 C578 B0 D5 FF C570 A7 C578 B0 C7 C578 B0 C7 C578 B0 C7 C578 B0 C7 C579 B0 D5 C578 B1 C579 B1 C577 D7 C7 C577 D7 C7	LC559  LC57C  *  LC58D  * CONSOL  DVEC4	DIRECT LDB :	#28*2 V977 LAC46 V973 DSEC #\$02 DCOPC LD5FF V978 V974 ,X #DIRLEN ,X #DIRLEN ,X+ LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC78F V976 \$02,U #503 LC78F V976 \$02,U LC78F LC725 FAT1,X FAT1,X FAT1,X WFATVL LC58D LC6F1 A,B,X,U,PC AMV VECTOR DEVNUM EVECTOR	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OF CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM MAGE OF DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG  * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RESTORE REGISTERS  GET DEVICE NUMBER BRANCH IF NON TED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RESTORE REGISTERS  GET DEVICE NUMBER BRANCH IF NOT DISK FILE GET RID OF RETURN ADDRESS SAVE REGISTERS  CLEAR BUFFER NOT EMPTY FLAG
9721 9722 9723 9724 9725 9726 9727 9730 9730 9731 9732 9733 9736 9736 9736 9741 9742 9743 9744 9744 9745 9747 9750 9750 9750 9750 9750 9750 9755 9756 9757 9758 9759 9769 9761 9761 9763 9764 9764 9764 9766	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C548 C7 E7 E9 C546 B0 D5 FF C552 BF 09 74 C555 33 84 C556 B6 67 C556 B0 D5 FF C556 B6 B0 C7 B6 C556 B6 C21 C568 B7 B7 B7 C570 B7 B7 C570 B7 B7 C570 B7 B7 C571 B7 C572 B7 C573 A7 C573 A7 C574 B7 C575 C6 C7 C573 A7 C576 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C579 B7 C570 B7 C571 B7 C571 B7 C572 B7 C573 A7 C573 A7 C574 B7 C575 B7 C576 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C570 B7 C570 B7 C571 B7 C571 B7 C572 B7 C573 B7 C574 B7 C575 B7 C577 B7 C576 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C579 B7 C579 B7 C579 B7 C570 B7 C577 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C570 B7 C577 B7 C577 B7 C577 B7 C578 B7 C579 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C570	LC559  LC57C  *  LC58D  * CONSOL  DVEC4	DIRECT LDB : LDA : LDA : LDB : STA : LDB : STA : LDB : JSR : LDD : LDB : LDB : LDB : LDB : JSR : LDB :	#28*2 1937 1AC46 1937 1DSEC #\$02 DCOPC LD5FF 19378 19374 ,X #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #H11 1A59A DFLTYP ,U #33 LC78F 1403 LC78F 1576 1582,U #\$03 DCOPC LD5FF ,U,X,B,A LC725 FAT1,X KFAT1,X K	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRYS SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM IMAGE OF DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA FLAG * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NON DEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RETURN IF NON DISK FILE GET RICH ALLOCATION SECTOR TO DISK RETURN IF NON DISK FILE GET RICH ALLOCATION SECTOR TO DISK RETURN IF NON DISK FILE GET RICH OF RETURN ADDRESS SAVE REGISTERS CLEAR BUFFER NOT EMPTY FLAG POINT TO FILE BUFFER VECTOR TABLE
0721 0722 0723 0724 0726 0726 0727 0728 0730 0731 0733 0733 0733 0736 0737 0738 0739 0741 0742 0743 0744 0745 0751 0755 0756 0757 0758 0759 0759 0761 0762 0763	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C548 C7 E7 E7 E7 C546 B7 E0 C546 B0 D5 FF C547 BE 09 74 C555 33 84 C557 C6 20 C558 5A C556 B7 80 94 C556 B7 80 97 C568 C6 21 C568 C6 21 C560 BD C7 8F C573 A7 42 C573 A7 42 C573 B7 09 76 C573 A7 42 C575 C6 03 C577 A7 42 C578 B1 09 7A C588 B1 09 7A C588 B1 09 7A C588 C7 B7 C579 BD C7 BF C570 A7 C7 C581 6C 01 C581 6C 01 C583 A6 01 C583 A6 01 C583 A6 01 C584 B0 C7 25 C581 6C 01 C583 A6 01 C583 A6 01 C583 A6 01 C584 B0 C7 25 C585 B1 09 7A C588 B2 03 C587 A1 C588 B3 C587 B3 C587 B0 C7 5C C587 3A 14 C599 07 70 C598 B2 09 26 C597 3A 14 C599 07 70 C598 B2 09 26 C598 D6 6F	LC559  LC57C  *  LC58D  * CONSOL  DVEC4	D DIRECT LDB : LDA : LDA : LDA : LDB : LDA : LDB	#28*2 1937 1AC46 1937 1DSEC #\$02 DCOPC LD5FF 19378 19374 ,X #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #H11 1A59A DFLTYP ,U #33 LC78F 1403 LC78F 1576 1582,U #\$03 DCOPC LD5FF ,U,X,B,A LC725 FAT1,X KFAT1,X K	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TTR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM MAGE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA IN FILE ALLOC TABLE GET NEW DATA FLAG * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NO NEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RESTORE REGISTERS  GET DEVICE NUMBER BRANCH IF NOT DISK FILE GET RID OF RETURN ADDRESS SAVE REGISTERS  CLEAR BUFFER NOT EMPTY FLAG POINT TO FILE BUFFER VECTOR TABLE GET ACTURE DISK FILE GET RID OF RETURN ADDRESS SAVE REGISTERS  CLEAR BUFFER NOT EMPTY FLAG POINT TO FILE BUFFER VECTOR TABLE GET ACTURE DISK FILE GET RID OFRETURN BUFFER VECTOR TABLE GET ACTURE DISK FILE NUMBER
9721 9722 9723 9724 9725 9726 9727 9730 9730 9731 9732 9733 9736 9736 9736 9741 9742 9743 9744 9744 9745 9747 9750 9750 9750 9750 9750 9750 9755 9756 9757 9758 9759 9769 9761 9761 9763 9764 9764 9764 9766	C53C B6 89 77 C53F 10 27 E7 03 C546 97 E0 C548 C6 02 C548 C7 E7 E9 C546 B0 D5 FF C552 BF 09 74 C555 33 84 C556 B6 67 C556 B0 D5 FF C556 B6 B0 C7 B6 C556 B6 C21 C568 B7 B7 B7 C570 B7 B7 C570 B7 B7 C570 B7 B7 C571 B7 C572 B7 C573 A7 C573 A7 C574 B7 C575 C6 C7 C573 A7 C576 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C579 B7 C570 B7 C571 B7 C571 B7 C572 B7 C573 A7 C573 A7 C574 B7 C575 B7 C576 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C570 B7 C570 B7 C571 B7 C571 B7 C572 B7 C573 B7 C574 B7 C575 B7 C577 B7 C576 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C579 B7 C579 B7 C579 B7 C570 B7 C577 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C570 B7 C577 B7 C577 B7 C577 B7 C578 B7 C579 B7 C577 B7 C577 B7 C578 B7 C579 B7 C579 B7 C570	LC559  LC57C  *  LC58D  * CONSOL  DVEC4	DIRECT LDB : LDA : LDA : LDB : STA : LDB : STA : LDB : JSR : LDD : LDB : LDB : LDB : LDB : JSR : LDB :	#28*2 1937 LAC46 1937 DSEC #\$02 DCOPC LD5FF 19378 W974 ,X #DIRLEN ,X+ #DIRLEN ,X+ #DIRLEN ,X+ #11 LC559 #DNAMBF #11 LA59A DFLTYP ,U #33 LC78F 1936 LC78F 1976 \$02,U #\$03 DCOPC LD5FF U,X,B,A LC725 FAT1,X FAT1,X WFATVL LC58D LC6F1 A,B,X,U,PC AM VECTOR DEVNUM XVEC4 \$02,S X,B CC1NBFL #FCEV1-2	'DISK FULL' ERROR GET SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY 'DISK FULL' ERROR IF NO EMPTY DIRECTORY ENTRYS SAVE SECTOR NUMBER OF FIRST EMPTY DIRECTORY ENTRY SAVE SECTOR NUMBER IN DSKCON REGISTER READ OP CODE SAVE IN DSKCON REGISTER READ SECTOR * GET ADDRESS OF RAM IMAGE OF UNUSED DIRECTORY * ENTRY AND SAVE AS CURRENT USED RAM IMAGE (TFR X,U) POINT U TO DIRECTORY RAM IMAGE SET COUNTER TO CLEAR 32 BYTES (DIRECTORY ENTRY) CLEAR BYTE DECREMENT COUNTER CONTINUE IF NOT DONE POINT TO FILENAME AND EXTENSION RAM IMAGE 11 BYTES IN FILENAME AND EXTENSION MOVE B BYTES FROM X TO U GET FILE TYPE AND ASCII FLAG SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM IMAGE FIRST GRANULE TO CHECK FIND THE FIRST FREE GRANULE SAVE IN RAM SAVE IN RAM IMAGE OF DIRECTORY TRACK * GET WRITE OPERATION CODE AND SAVE * IT IN DSKCON REGISTER GO WRITE A SECTOR IN DIRECTORY SAVE REGISTERS POINT X TO FILE ALLOCATION TABLE INDICATE NEW DATA FLAG * HAVE ENOUGH GRANULES BEEN REMOVED FROM THE FAT TO * CAUSE THE FAT TO BE WRITTEN TO THE DISK RETURN IF NON DEED TO WRITE OUT ALLOCATION TABLE WRITE FILE ALLOCATION SECTOR TO DISK RETURN IF NON DISK FILE GET RICH ALLOCATION SECTOR TO DISK RETURN IF NON DISK FILE GET RICH ALLOCATION SECTOR TO DISK RETURN IF NON DISK FILE GET RICH OF RETURN ADDRESS SAVE REGISTERS CLEAR BUFFER NOT EMPTY FLAG POINT TO FILE BUFFER VECTOR TABLE

Ø769	C5A1 AE 85	LDX B,X	NOW X POINTS TO FILE BUFFER
Ø77Ø Ø771	C5A3 E6 84 C5A5 C1 40	LDB FCBTYP,X CMPB #RANFIL	GET FILE TYPE IS THIS A RANDOM (DIRECT) FILE?
Ø772	C5A7 26 16	BNE LC5BF	BRANCH IF NOT
Ø773			
0774	0510 50 00 15	* GET A BYTE FROM A RANDOM FILE - RETURN CHAR	
Ø775 Ø776	C5A9 EC 88 15 C5AC 10 A3 09	LDD FCBGET,X CMPD FCBRLN,X	GET THE RECORD COUNTER *COMPARE TO RECORD LENGTH AND
Ø777	C5AC 10 A3 09 C5AF 24 20	RHS I CED1	*BRANCH TO BUFFER EMPTY IF >= RECORD LENGTH
	C5B1 C3 ØØ Ø1	ADDD #\$0001	= ADD ONE TO RECORD POINTER AND
	C5B4 ED 88 15	ADDD #\$0001 STD FCBGET,X LDX FCBBUF,X	= SAVE IT IN FCB
	C5B7 AE ØB	LDX FCBBUF,X	* POINT X TO START OF RANDOM FILE BUFFER AND
	C5B9 3Ø 8B C5BB A6 1F	LEAX D,X LDA -1,X	* ADD THE RECORD COUNTER TO IT GET A CHARACTER FROM THE BUFFER
Ø783		PULS B,X,PC	RESTORE REGISTERS AND RETURN
0701		* GET A BYTE FROM A SEQUENTIAL FILE	
Ø785		LC5BF LDB FCBCFL,X	* TEST THE CACHE FLAG AND BRANCH IF AN
Ø786 Ø787	C5C2 27 W6	BEQ LC5CC LDA FCBCDT,X	* EXTRA CHARACTER HAS NOT BEEN READ FROM FILE GET THE CACHE CHARACTER
Ø788	C5C7 6F 88 1Ø	CLR FCBCFL,X	CLEAR THE CACHE FLAG
Ø789	C5CA 35 94	PULS B,X,PC	RESTORE REGISTERS AND RETURN
0790			
Ø791		LC5CC LDB FCBDFL,X	IS ANY DATA LEFT?
Ø792 Ø793	C5CF 27 Ø4 C5D1 Ø3 7Ø	BEQ LC5D5 LC5D1 COM CINBFL	BRANCH IF SO SET FLAG TO BUFFER EMPTY
Ø794	C5D3 35 94	PULS B,X,PC	SET TENG TO BOTTER EMITT
Ø795			
Ø796	C5D5 E6 Ø5	LC5D5 LDB FCBCPT,X	GET CHARACTER POINTER
Ø797 Ø798	C5D7 6C Ø5 C5D9 6A 88 18	INC FCBCPT,X DEC FCBLFT,X	ADD ONE TO CHARACTER POINTER DECREMENT NUMBER OF CHARACTERS LEFT IN FILE BUFFER
	C5DC 27 Ø6		IF LAST CHARACTER, GO GET SOME MORE
	C5DE 3A		ADD CHARACTER COUNTER TO X
	C5DF A6 88 19	LDA FCBCON,X	GET DATA CHARACTER (SKIP PAST 25 FCB CONTROL BYTES
	C5E2 35 94	PULS B,X,PC	W 01145 TH 4004
Ø8Ø3 Ø8Ø4	C5E4 34 6Ø	* GET A CHARACTER FROM FCB DATA BUFFER - RETUR LC5E4 PSHS U,Y	N CHAR IN ACCA SAVE REGISTERS
Ø8Ø5	C5E6 4F	CLRA	*
Ø8Ø6	C5E7 33 8B	LEAU D,X LDA FCBCON,U	* POINT U TO CORRECT CHARACTER
Ø8Ø7	C5E9 A6 C8 19		=GET DATA CHAR (SKIP PAST 25 CONTROL BYTES)
	C5EC 34 Ø2	PSHS A	=AND SAVE DATA CHARACTER ON STACK
Ø81Ø	C5EE 6F Ø5 C5FØ A6 Ø1	IDA FORDRY X	RESET CHAR POINTER TO START OF BUFFER GET DRIVE NUMBER AND SAVE IT IN
Ø811	C5EE 6F Ø5 C5FØ A6 Ø1 C5F2 97 EB	PSHS A CLR FCBCPT,X LDA FCBDRV,X STA DCDRV BSR IC5FA	DSKCON VARIABLE
Ø812	C5F4 8D Ø4	2017	GO READ A SECTOR - FILL THE BUFFER
Ø813	C5F6 35 62	PULS A,Y,U	RESTORE REGISTERS AND DATA CHARACTER
Ø814 Ø815	C5F8 35 94	PULS B,X,PC  * REFILL THE FCB INPUT DATA BUFFER FOR SEQUENT	RESTORE REGISTERS AND RETURN
	C5FA A6 Ø4	LC5FA LDA FCBSEC,X	GET CURRENT SECTOR NUMBER
	C5FC 4C	LC5FC INCA	ADD ONE
	C5FD 34 Ø2	PSHS A	SAVE NEW SECTOR NUMBER ON THE STACK
	C5FF 81 Ø9 C6Ø1 23 Ø1	CMPA #\$Ø9 BLS LC6Ø4	NINE SECTORS PER GRANULE BRANCH IF <= 9
	C6Ø3 4F	CLRA	SET TO SECTOR ZERO
		LC604 STA FCBSEC,X	SAVE SECTOR NUMBER
	C606 E6 03	LDB FCBCGR, X	GET GRANULE NUMBET TO FAT POINTER
	C608 33 84 C60A BD C7 25	LEAU ,X JSR LC725	POINT U TO FCB (TFR X,U) POINT X TO PROPER FILE ALLOCATION TABLE
	C6ØD 3A	ABX	ADD OLD GRANULE NUMBER TO FAT POINTER
		LDB FATCON,X	GET GRANULE NUMBER (6 CONTROL BYTES AT FRONT OF FAT)
Ø828	C60E E6 06 C610 30 C4 C612 C1 C0 C614 24 0A C616 35 02 C618 80 0A C61A 26 15	LEAX ,U	POINT X TO FCB
Ø829	C612 C1 CØ	CMPB #\$CØ BHS LC62Ø	IS CURRENT GRANULE LAST ONE IN FILE? YES
Ø831	C616 35 Ø2	PULS A	GET SECTOR NUMBER
Ø832	C618 80 ØA	SUBA #10	WAS IT 10? - OVERFLOW TO NEXT GRANULE IF SO
Ø833	C61A 26 15	BNE LC631	BRANCH IF NOT
B054	C61C E7 Ø3 C61E 2Ø DC	STB FCBCGR,X BRA LC5FC	SAVE NEW GRANULE NUMBER SET VARIABLES FOR NEW GRANULE
		LC62Ø ANDB #\$3F	GET NUMBER OF SECTORS USED IN THIS GRANULE
Ø837	C622 C1 Ø9	CMPB #\$Ø9	9 SECTORS / GRANULE
	C624 23 Ø5	BLS LC62B	BRANCH IF OK
	C626 C6 4Ø C628 7E AC 46	LC626 LDB #2*32 JMP LAC46	'BAD FILE STRUCTURE' ERROR ERROR DRIVER
		LC62B SUBB ,S+	SUBTRACT CURRENT SECTOR NUMBER AND PULS A
	C62D 25 21	BLO LC65Ø	BRANCH IF PAST LAST SECTOR
	C62F 1F 98	TFR B,A	SECTOR NUMBER TO ACCA
		LC631 PSHS A BSR LC658	SAVE SECTOR NUMBER DIFFERENCE
	C633 8D 23 C635 86 Ø2	LDA #\$02	INCREMENT RECORD NUMBER *GET READ OPERATION CODE
Ø847	C637 97 EA	STA DCOPC	*AND SAVE IT IN DSKCON VARIABLE
Ø848	C639 BD C7 33	JSR LC733	GET PROPER TRACK AND SECTOR TO DSKCON VARIABLES
DO42	0030 33 00 19	LEAU FCBCON,X	* POINT U TO START OF FCB DATA BUFFER
	C63F DF EE C641 BD D5 FF	STU DCBPT JSR LD5FF	* AND SAVE IT IN DSKCON VARIABLE GO READ A SECTOR INTO FCB BUFFER
	C644 6F 88 18	CLR FCBLFT,X	NUMBER OF CHARS LEFT IN BUFFER = 256
Ø853	C647 E6 EØ		GET SECTOR NUMBER OFF STACK
	C649 26 ØC	LDB ,S+ BNE LC657 LDD FCBLST,X BNE LC654	RETURN IF DATA LEFT; FALL THRU IF LAST SECTOR
	C64B EC 88 13 C64E 26 Ø4	LDD FCBLST,X BNE LC654	GET NUMBER OF BYTES IN THE LAST SECTOR BRANCH IF SOME BYTES IN LAST SECTOR
		LC65Ø CLRB	SET NUMBER OF REMAINING BYTES = 256
Ø858	C651 63 88 17	COM FCBDFL,X	SET DATA LEFT FLAG TO \$FF
	C654 E7 88 18	LC654 STB FCBLFT,X	SAVE THE NUMBER OF CHARS LEFT IN BUFFER
Ø86Ø Ø861	C657 39	LC657 RTS	
	C658 EE Ø7	LC658 LDU FCBREC,X	GET CURRENT RECORD NUMBER
	C65A 33 41	LEAU \$01,U	BUMP IT
	C65C EF Ø7	STU FCBREC, X	PUT IT BACK

Ø865	C65E	39			RTS		
Ø866 Ø867				* CUN I	DIBECTO	RY FOR FILENAME.EXT FOUND IN DNAM	NDE TE ETLENAME EQUIND
Ø868						SECTOR NUMBER IN V973, GRANULE IN	
Ø869				* CONTA	INING D	IRECTORY DATA IN V974. IF DISK IS	FULL THEN V973,
Ø87Ø	0.655	aa	- 70			E FIRST UNUSED SECTOR RETURNED IN	
Ø871 Ø872	C65F C662			LC65F	CLR CLR	V9/3 V977	CLEAR SECTOR NUMBER
Ø873	C665					#\$1102	CLEAR TEMP SECTOR COUNTER TRACK 17 (DIRECTORY), READ OPERATION CODE
Ø874	C668					DCTRK	SAVE TRACK NUMBER
Ø875	C66A				STB	DCOPC	SAVE OPERATION CODE (READ)
Ø876	C66C					#\$Ø3	READ SECTOR 3 (FIRST DIRECTORY SECTOR)
Ø877	C66E			LC66E		DSEC #DBUFØ	SAVE SECTOR NUMBER IN DSKCON VARIABLE
	C67Ø C673					DCBPT	*BUFFER AREA NUMBER Ø AS DATA BUFFER - SAVE *IN DSKCON VARIABLE
	C675				JSR		GO READ A SECTOR
Ø881	C678	FF Ø9	74	LC678	STU	V974	SAVE RAM DIRECTORY BUFFER ADDRESS
	C67B				LEAY		POINT Y TO DIRECTORY BUFFER
Ø883	C67D .					,U	GET A BYTE FROM BUFFER
Ø884 Ø885	C67F C681				BNE BSR		BRANCH IF NOT ZERO - FILE IS ACTIVE SET UNUSED FILE POINTERS IF ENTRY HAS BEEN KILLED
	C683			LC683		#DNAMBF	POINT TO DISK FILE NAME BUFFER
	C686			LC686		, X+	*COMPARE THE FILENAME AND EXTENSION
	C688				CMPA		*STORED IN RAM AT DNAMBF TO THE DIRECTORY
Ø889	C68A				BNE		*ENTRY STORED AT ,U (BRANCH IF MISMATCH)
Ø89Ø Ø891	C68C C68F				BNE	#DNAMBF+11	AT END OF FILE NAME BUFFER? BRANCH IF NOT DONE CHECKING FILENAME
	C691					V973	SAVE SECTOR NUMBER IN DSKCON VARIABLE
Ø893	C694					FCBFGR,U	*GET NUMBER OF FIRST GRANULE IN FILE
Ø894	C696		76			V976	*AND SAVE IT IN V976
Ø895	C699	39			RTS		
Ø896 Ø897	C69A	33 V 8	201	LC69A	LEAH	DIRLEN, Y	GET NEXT DIRECTORY ENTRY (DIRLEN BYTES PER ENTRY)
Ø898	C69D					#DBUFØ+SECLEN	AT END OF BUFFER?
Ø899	C6A1				BNE		CHECK NEXT ENTRY IF NOT AT END
0900	C6A3				INCB		NEXT SECTOR
0901	C6A4				CMPB		11 SECTORS MAX IN DIRECTORY
Ø9Ø2 Ø9Ø3	C6A6 C6A8		•		BLS RTS	LC66E	BRANCH IF MORE SECTORS
Ø9Ø4	COAO	33			KIS		
	C6A9	43		LC6A9	COMA		COMPLEMENT FIRST BYTE IN DIRECTORY EMTRY
0906	C6AA	26 D7			BNE	LC683	BRANCH IF FILE IS ACTIVE - FALL THRU IF NOT USED
Ø9Ø7				+ CET D		FOR FIRST UNUSER RIBECTORY ENTRY	
Ø9Ø8 Ø9Ø9	C6AC	R6 Ø0	77	LC6AC		FOR FIRST UNUSED DIRECTORY ENTRY V977	UNUSED ENTRY ALREADY FOUND?
Ø91Ø	C6AF			LCOAC		DVEC12	RETURN IF UNUSED ENTRY ALREADY FOUND
Ø911	C6B1					V977	SECTOR CONTAINING THIS DIRECTORY ENTRY
Ø912	C6B4		78			V978	POINTS TO RAM AREA WHERE DIRECTORY DATA IS STORED
Ø913	C6B7	39		DVEC12	RTS		
Ø914 Ø915	C6B8	re 21		LC6B8	LDB	#2*26	'NE' ERROR
Ø916	C6BA			LCOBO		V973	WAS A DIRECTORY MATCH FOUND?
Ø917	C6BD					DVEC12	RETURN IF FOUND
0918	C6BF				JMP	LAC46	JUMP TO ERROR HANDLER IF NOT FOUND
Ø919							
Ø92Ø Ø921	C6C2	ם רם	07	* KILL (	JSR		GET FILENAME.EXT FROM BASIC
	C6C5			KILL	JSR		'SYNTAX' ERROR IF MORE CHARACTERS ON LINE
Ø923	C6C8					LC76D	GET VALID FAT DATA
Ø924	C6CB					LC65F	TEST FOR FILE NAME MATCH IN DIRECTORY
Ø925	C6CD					LC6B8	MAKE SURE THE FILE EXISTED
Ø926 Ø927	C6CF	86 FF		LC6CF	LDA	#\$FF	* MATCH FILE TYPE = \$FF; THIS WILL CAUSE AN 'AO' * ERROR TO BE GENERATED IF ANY FILE TYPE IS OPEN
Ø928	C6D1	BD C7	D7		JSR	LC7D7	CHECK TO MAKE SURE FILE IS NOT OPEN
Ø929	C6D4					V974	*GET RAM IMAGE OF DIRECTORY
	C6D7					DIRNAM, X	*AND ZERO FIRST BYTE - KILL FILE
	C6D9				LDB		=WRITE OPERATION CODE - SAVE
	C6DB C6DD				STB JSR		=IT IN DSKCON VARIABLE WRITE A SECTOR
	C6EØ					DIRGRN,X	GET NUMBER OF FIRST GRANULE IN FILE
Ø935	C6E2	8D 41		LC6E2	BSR	LC725	POINT X TO PROPER FILE ALLOCATION TABLE
	C6E4		i			FATCON, X	SKIP 6 CONTROL BYTES
	C6E6				ABX	w.	POINT TO CORRECT ENTRY
	C6E7				LDB LDA	,X #¢FF	GET NEXT GRANULE *GET FREE GRANULE FLAG AND
	C6EB				STA		*MARK GRANULE AS FREE
	C6ED				CMPB		WAS THIS THE LAST GRANULE?
	C6EF				BLO		* KEEP FREEING GRANULES IF NOT LAST ONE
Ø943				*			* WRITE FILE ALLOCATION SECTOR TO DIRECTORY - DO NOT WRITE
Ø944	CC	rr ~-		*	1.00	#DDIIE4	* THE SIX CONTROL BYTES AT THE START OF THE FAT TO THE DISK
	C6F1 C6F4			LC6F1		#DBUFØ DCBPT	=POINT U TO DISK BUFFER Ø AND =SAVE IT AS DSKCON VARIABLE
	C6F6					#\$11Ø3	* WRITE DIRECTORY TRACK - SAVE
	C6F9					DCTRK	* TRACK AND WRITE OPERATION CODE IN
Ø949	C6FB	D7 EA	1		STB	DCOPC	* DSKCON VARIABLES
	C6FD				LDB		= GET FILE ALLOCATION SECTOR AND
	C6FF				STB		= SAVE IN DSKCON VARIABLE
	C7Ø1 C7Ø3				BSR	LC725 FAT1,X	POINT X TO PROPER FILE ALLOCATION TABLE RESET FLAG INDICATING VALID FAT DATA HAS BEEN STORED ON DISK
	C7Ø5					FATCON, X	MOVE (X) TO START OF GRANULE DATA
	C7Ø7					#GRANMX	68 BYTES IN FAT
Ø956	C7Ø9				JSR		MOVE ACCB BYTES FROM FAT RAM IMAGE TO DBUFØ
Ø957					OUT	OF THE DUTES IN THE T-	WITCH DO NOT CONTAIN THE CRANNES THE
Ø958 Ø959						OF THE BYTES IN THE FAT SECTOR W REMAINDER OF THE SECTOR BUFFER	IHICH DO NOT CONTAIN THE GRANULE DATA
	C7ØC	6F 80	i		CLR		THIS IS A BUG; SHOULD BE CLR ,U+
							· · · · · · · · · · · · · · · · · · ·

```
Ø961
         C70E 8C 07 00
                                                  CMPX #DBUFØ+SECLEN
                                                                                                     MORE OF THE SAME BUG; SHOULD BE CMPU
                                       * BNE LC70C THIS INSTRUCTION HAS BEEN LEFT OUT
Ø962
Ø963
         C711 7E D5 FF
                                                  JMP
                                                         LD5FF
                                                                                                     WRITE A SECTOR
Ø964
Ø965
                                       * ENTER WITH ACCB CONTAINING FILE NUMBER (1-15); EXIT WITH X POINTING
                                       * TO CORRECT FILE BUFFER; FLAGS SET ACCORDING TO FILE TYPE.
Ø966
Ø967
         C714 34 Ø4
                                                                                                     SAVE FILE NUMBER ON STACK
Ø968
                                       LC714
                                                  PSHS B
                                                        DEVNUM
         C716 D6 6F
                                                                                                     GET DEVICE NUMBER (FILE NUMBER)
Ø969
                                       LC718
Ø97Ø
         C718 8C 34 Ø4
                                                  CMPX #$3404
                                                                                                     SKIP TWO BYTES
                                                                                                      SAVE FILE NUMBER ON STACK
Ø971
         C719 34 Ø4
                                       LC719
                                                  PSHS
                                                                                                     X2: 2 BYTES PER POINTER
POINT X TO START OF FCB POINTERS
0972
         C71B 58
                                                  ASIR
Ø973
         C71C 8E Ø9 26
                                                         #FCBV1-2
                                                  LDX
Ø974
         C71F AE 85
                                                  LDX
                                                         B,X
FCBTYP,X
                                                                                                     POINT X TO PROPER FCB
         C721 E6 ØØ
                                                                                                     SET FLAGS ACCORDING TO FILE TYPE
Ø975
                                                  LDB
Ø976
         C723 35 84
                                                                                                     RESTORE FILE NUMBER
Ø977
Ø978
                                       * POINT X TO DRIVE ALLOCATION TABLE
Ø979
         C725 34 Ø6
C727 96 EB
                                                  PSHS B,A
                                                                                                     SAVE ACCD ON STACK
Ø98Ø
                                       LC725
                                                         DCDRV
Ø981
                                                  LDA
                                                                                                     GET DRIVE NUMBER
Ø982
         C729 C6 4A
                                                  LDB
                                                         #FATLEN
                                                                                                     GET LENGTH OF FILE ALLOCATION TABLE
Ø983
         C72B 3D
                                                  MUL
                                                                                                     MULTIPLY BY DRIVE NUMBER TO GET OFFSET START OF FILE ALLOCATION TABLE
         C72C 8E Ø8 ØØ
                                                  LDX
                                                         #FATBLØ
Ø984
Ø985
         C72F 3Ø 8B
                                                  LEAX D,X
                                                                                                     POINT TO RIGHT TABLE
                                                  PULS A,B,PC
         C731 35 86
                                                                                                     RESTORE ACCD
Ø986
Ø987
                                       * CONVERT GRANULE NUMBER TO TRACK & SECTOR NUMBER - X MUST BE POINTING TO CORRECT 
* FCB; THE TRACK AND SECTOR NUMBER WILL BE STORED IN DSKCON REGISTERS 
LC733 LDB FCBCGR,X GET GRANULE NUMBER
0988
Ø989
         C733 E6 Ø3
0990
         C735 54
                                                  LSRB
                                                                                                     DIVIDE BY 2 - 2 GRANULES / TRACK
Ø991
Ø992
         C736 D7 EC
                                                  STB DCTRK
                                                                                                     TRACK NUMBER
                                                                                                     TRACK 17 = DIRECTORY TRACK
BRANCH IF < DIRECTORY TRACK
INCR TRACK NUMBER IF > DIRECTORY TRACK
         C738 C1 11
Ø993
                                                  CMPB
                                                         LC73F
Ø994
         C73A 25 Ø2
                                                  BL0
         C73C ØC EC
Ø995
                                                  INC
                                                         DCTRK
                                                                                                     MULTIPLY TRACK NUMBER BY 2
NEGATE GRANULE NUMBER
Ø996
         C73E 58
                                       LC73E
                                                  ASIR
         C73F 5Ø
Ø997
                                                  NEGB
Ø998
         C74Ø EB Ø3
                                                  ADDB
                                                        FCBCGR,X
                                                                                                     B=Ø IF EVEN GRANULE; 1 IF ODD
                                                                                                     RETURN B=Ø FOR EVEN GRANULE NUMBER, B=9 FOR ODD GRANULE NUMBER
Ø999
         C742 8D Ø5
                                                  BSR
                                                         LC749
         C744 EB Ø4
                                                  ADDB
                                                         FCBSEC,X
                                                                                                     ADD SECTOR NUMBER
         C746 D7 ED
1001
                                                  STB
                                                        DSEC
                                                                                                     SAVE SECTOR NUMBER
1002
         C748 39
                                       * MULTIPLY ACCD BY 9
1003
         C749 34 Ø6
                                                  PSHS B,A
1004
                                       LC749
                                                                                                     TEMP STORE ACCD ON STACK
1005
         C74B 58
                                                  ASLB
1006
         C74C 49
                                                   ROLA
                                                                                                     * MULTIPLY BY 2
1007
         C74D 58
                                                  ASLB
         C74E 49
                                                                                                     = MULTIPLY BY FOUR
1008
                                                  ROLA
1009
         C74F 58
                                                  ASLB
         C75Ø 49
                                                                                                      * MULTIPLY BY EIGHT
1010
                                                  ROLA
                                                         ,S++
                                                                                                     ADD ONE = MULTIPLY BY NINE
1012
         C753 39
1013
                                       * CONVERT ACCD INTO A GRANULE NUMBER - RETURN RESULT IN ACCB:
1014
                                       * ENTER WITH ACCD CONTAINING A NUMBER OF SECTORS. RETURN IN ACCB

* THE NUMBER (Ø-67) CORRESPONDING TO THE NUMBER OF COMPLETE

* GRANULES CONTAINED IN THAT MANY SECTORS.
1015
1016
1017
                                       * DIVIDE BY 90, MULTIPLY BY 10 IS FASTER THAN DIVIDE BY 9
LC754 CLR ,-S CLEAR A TEMPORARY SLOT ON THE STACK
1018
                                       LC754
                                                  CLR ,-S
INC ,S
SUBD #9*10
1019
         C754 6F E2
1020
         C756 6C E4
C758 83 ØØ 5A
                                       LC756
                                                  INC
                                                                                                     * DIVIDE ACCD BY 90 - SAVE THE

* QUOTIENT+1 ON THE STACK - REMAINDER
1021
         C75B 2A F9
                                                        LC756
                                                                                                     * IN ACCB
1022
                                                                                                     = PUT THE OUOTIENT+1 IN ACCA AND
1023
         C75D A6 E4
                                                  LDA
                                                          ,S
                                                                                                     = SAVE REMAINDER ON STACK
1024
         C75F E7 E4
                                                                                                     * MULTIPLY (QUOTIENT+1)
* BY 10
1025
         C761 C6 ØA
                                                  I DR
                                                         #1 Ø
1026
         C763 3D
                                                  MUL
                                                                                                     PUT THE REMAINDER IN ACCA
         C764 35 Ø2
1027
                                                  PULS A
                                                                                                     * DECREMENT THE GRANULE COUNT BY ONE FOR

* EVERY NINE SECTORS (1 GRANULE) IN THE

* REMAINDER - COMPENSATE FOR THE + 1 IN QUOTIENT+1
1028
         C766 5A
                                       LC766
                                                  DECB
1029
         C767 8B Ø9
                                                  ADDA #$Ø9
         C769 2B FB
1030
                                                  BMI
                                                         LC766
1031
         C76B 4F
                                                  CLRA
                                                                                                     CLEAR MS BYTE OF ACCD
                                       LC76C
1032
         C76C 39
                                                  RTS
1033
                                       * MAKE SURE RAM FILE ALLOCATION TABLE DATA IS VALID
1034
                                                                                                     POINT X TO FAT FOR THE CORRECT DRIVE NUMBER
CHECK TO SEE IF ANY FILES ARE ACTIVE
RETURN IF ANY FILES ACTIVE IN THIS FAT
1035
         C76D 8D B6
C76F 6D ØØ
                                       LC76D
                                                  BSR
                                                        LC725
1036
                                                  TST
                                                         FATØ.X
         C771 26 F9
1037
                                                  BNE
                                                          LC76C
1038
1039
         C773 6F Ø1
C775 33 Ø6
                                                                                                     RESET FAT DATA VALID FLAG
LOAD U WITH START OF GRANULE DATA BUFFER
                                                  CLR
                                                          FAT1.X
                                                  LEAU
                                                         FATCON, X
1040
         C777 8E Ø6 ØØ
                                                  I DX
                                                         #DBUFØ
                                                                                                     BUFFER FOR DISK TRANSFER
PUT IN DSKCON PARAMETER
         C77A 9F EE
1041
                                                  STX
                                                         DCBPT
1042
         C77C CC 11 Ø2
C77F 97 EC
                                                  חחו
                                                         #$1102
                                                                                                     DIRECTORY TRACK, READ SECTOR
STORE IN DSKCON TRACK NUMBER
                                                         DCTRK
1043
                                                  STA
1044
         C781 D7 EA
                                                          DCOPC
                                                                                                     STORE IN DSKCON OP CODE
GET SECTOR NUMBER 2 (FILE ALLOCATION TABLE)
                                                  STB
1045
         C783 C6 Ø2
                                                  LDB
                                                         #$02
1046
         C785 D7 ED
                                                                                                      STORE IN DSKCON PARAMETER
                                                  STB
1047
         C787 BD D5 FF
                                                  JSR
                                                         I D5FF
                                                                                                     GO READ SECTOR
                                                                                                     TRANSFER FILE ALLOCATION TABLE TO FILE ALLOC TABLE BUFFER
1048
         C78A C6 44
                                                         #GRANMX
                                                  LDB
1049
         C78C 7E A5 9A
                                                         LA59A
                                                                                                     MOVE B BYTES FROM (X) TO (U)
1050
                                       * FIND FIRST FREE GRANULE - ENTER WITH ACCB CONTAINING
* GRANULE FROM WHICH TO START SEARCHING. THE FOUND GRANULE
* IS MARKED BY STORING A $CØ IN THE GRANULE'S DATA BYTE
* TO INDICATE THAT IT IS THE LAST GRANULE IN THE FILE.
1051
1052
1053
1054
1055
                                       * RETURN WITH FIRST FREE GRANULE FOUND IN ACCA
         C78F 8D 94
                                                                                                     POINT X TO FILE ALLOC TABLE
                                                 BSR LC725
1056
                                       LC78F
```

1057	C791 30 06		LEAX FATCON,X	SKIP CONTROL BYTES
1058 1059	C793 4F		CLRA ANDB #\$FE	USE ACCA AS GRANULE COUNTER
1059	C794 C4 FE C796 6F E2		CLR ,-S	MASK OFF BIT ZERO OF SEARCH GRANULE INITIALIZE AND SAVE A BYTE ON STACK (DIRECTION FLAG)
1061	C798 63 85	LC798	COM B,X	IS THIS GRANULE FREE? (\$FF=FREE)
1062	C79A 27 31		BEQ LC7CD	BRANCH IF IT IS
1063 1064	C79C 63 85 C79E 4C		COM B,X INCA	RESTORE GRANULE DATA ADD ONE TO GRANULE COUNTER
1065	C79F 81 44		CMPA #GRANMX	GRANMX GEANULES PER DISK
1066	C7A1 24 25		BHS LC7C8	BRANCH IF ALL GRANULES CHECKED (DISK FULL)
1067	C7A3 5C		INCB	INCR TO NEXT GRANULE
1068 1069	C7A4 C5 Ø1 C7A6 26 FØ		BITB #\$Ø1 BNE LC798	IS BIT Ø SET? BRANCH IF ODD GRANULE NUMBER (SAME TRACK)
1070	C7A8 34 Ø6		PSHS B,A	SAVE GRANULE COUNTER AND CURRENT GRANULE NUMBER
1071	C7AA CØ Ø2		SUBB #\$Ø2	SUBTRACT ONE TRACK (2 GRANULES)
1072 1073	C7AC 63 62 C7AE 26 ØC		COM \$02,S BNE LC7BC	COMPLEMENT DIRECTION FLAG BRANCH EVERY OTHER TIME
1074	C7BØ EØ EØ		SUBB ,S+	SUBTRACT THE GRANULE COUNTER FROM THE CURRENT GRANULE NUMBER
1075 1076	C7B2 2A Ø4		BPL LC7B8 LDB ,S	BRANCH IF LOWER BOUND NOT EXCEEDED RESTORE CURRENT GRANULE NUMBER IF LOWER BOUND EXCEEDED
1077	C7B4 E6 E4 C7B6 63 61	LC7B6	LDB ,S COM \$01,S	* COMPLEMENT FLAG - IF GRANULE NUMBER HAS EXCEEDED
1078		*	,	* BOUNDS ON EITHER THE HI OR LO SIDE, FORCE IT TO GO IN
1079	C7D0 22 61	*	LEAC ##1 C	* THE DIRECTION OPPOSITE THE EXCEEDED BOUND
1080 1081	C7B8 32 61 C7BA 20 DC	LC7B8	LEAS \$01,S BRA LC798	CLEAN UP STACK CHECK FOR ANOTHER FREE GRANULE
1082				
1083 1084	C7BC EB EØ C7BE C1 44	LC7BC	ADDB ,S+ CMPB #GRANMX	ADD THE GRANULE COUNTER TO THE CURRENT GRANULE NUMBER GRANMX GRANULES PER DISK
1085	C7CØ 25 F6		BLO LC7B8	BRANCH IF UPPER BOUND NOT EXCEEDED
1086	C7C2 E6 E4		LDB ,S	* RESTORE CURRENT GRANULE COUNT AND GO TWICE
1087 1088	C7C4 CØ Ø4 C7C6 2Ø EE		SUBB #\$04 BRA LC7B6	* AS FAR AS USUAL IN OPPOSITE DIRECTION IF UPPER BOUND EXCEEDED KEEP SEARCHING
1089	C7C8 C6 38	LC7C8	LDB #2*28	'DISK FULL' ERROR
1090	C7CA 7E AC 46		JMP LAC46	JUMP TO ERROR HANDLER
1091 1092		* POINT	X TO FIRST FREE GRANULE POSITI	ON IN THE FILE ALLOCATION
1093		* TABLE	AND MARK THE POSITION WITH A L	AST GRANULE IN FILE MARKER
1094	C7CD 32 61 C7CF 1F 98	LC7CD	LEAS \$01,S	CLEAR UP STACK - REMOVE DIRECTION FLAG
1095 1096	C7D1 3A		TFR B,A ABX	GRANULE NUMBER TO ACCA POINT X TO FIRST FOUND GRANULE
1097	C7D2 C6 CØ		LDB #\$CØ	LAST GRANULE FLAG
1098 1099	C7D4 E7 84 C7D6 39	LC7D6	STB ,X RTS	MARK THE FIRST FOUND GRANULE AS THE LAST GRANULE
1100	0.20 03	20750	W.C	
1101				A FILE IS NOT ALREADY OPEN - TO BE OPEN
1102 1103			E BUFFER MUST MATCH THE DRIVE N M DIRECTORY ENTRY AND THE FCB T	UMBER AND FIRST GRANULE NUMBER YPE MUST NOT MATCH THE FILE TYPE IN ACCA
1104		* AN 'A	O' ERROR WILL NOT BE GENERATED	IF A FILE IS BEING OPENED FOR
1105		* THE SA	AME MODE THAT IT HAS ALREADY BE	EN OPENED UNDER.
1106 1107	C7D7 34 Ø2	LC7D7	PSHS A	SAVE FILE TYPE ON STACK
1108	C7D9 F6 Ø9 5B		LDB FCBACT	NUMBER OF CURRENTLY OPEN FILES
1109 1110	C7DC 5C C7DD BD C7 19	LC7DD	INCB JSR LC719	ADD ONE MORE TO FILE COUNTER POINT X TO FCB OF THIS FILE
1111	C7EØ 27 17	20700	BEQ LC7F9	BRANCH IF BUFFER NOT BEING USED
1112	C7E2 96 EB		LDA DCDRV	* GET DRIVE NUMBER AND CHECK TO SEE IF IT
1113 1114	C7E4 A1 Ø1 C7E6 26 11		CMPA FCBDRV,X BNE LC7F9	* MATCHES THE DRIVE NUMBER FOR THIS BUFFER FILE EXISTS ON ANOTHER DRIVE
1115	C7E8 FE Ø9 74		LDU V974	GET RAM DIRECTORY AREA
1116	C7EB A6 4D		LDA DIRGRN,U	GET FIRST GRANULE IN FILE
1117 1118	C7ED A1 Ø2 C7EF 26 Ø8		CMPA FCBFGR,X BNE LC7F9	DOES IT MATCH THIS FILE BUFFER? NO
1119	C7F1 A6 ØØ		LDA FCBTYP,X	GET FILE TYPE OF THIS BUFFER
1120	C7F3 A1 E4		CMPA ,S	DOES IT MATCH THE ONE WE ARE LOOKING FOR?
1121 1122	C7F5 10 26 DE 23 C7F9 5A	LC7F9	LBNE LA61C DECB	'FILE ALREADY OPEN' ERROR IF NOT DECR FILE COUNTER
1123	C7FA 26 E1	23713	BNE LC7DD	BRANCH IF HAVEN'T CHECKED ALL ACTIVE FILES
1124	C7FC 35 82		PULS A,PC	RESTORE FILE TYPE AND RETURN
1125 1126	C7FE BD A5 A5	LC7FE	JSR LA5A5	EVALUATE AN EXPRESSION (DEVICE NUMBER)
1127	C8Ø1 ØF 6F		CLR DEVNUM	SET DEVICE NUMBER TO SCREEN
1128 1129	C8Ø3 5D C8Ø4 1Ø 2F EC 42		TSTB LBLE LB44A	TEST NEW DEVICE NUMBER 'FC' ERROR IF DEVICE NUMBER NOT A DISK FILE
1129	C8Ø8 BD C7 19		JSR LC719	POINT X TO FCB
1131	C8ØB A6 ØØ		LDA FCBTYP,X	TEST IF BUFFER IS IN USE
1132 1133	C8ØD 10 27 DB EA C811 81 40		LBEQ LA3FB CMPA #RANFIL	'FILE NOT OPEN' ERROR DIRECT/RANDOM FILE?
1134	C813 27 C1		BEQ LC7D6	RETURN IF RANDOM
1135	C815 7E A6 16	LC815	JMP LA616	BAD FILE MODE ERROR IF NOT RANDOM
1136 1137		* INPUT	DEVICE NUMBER CHECK RAM HOOK	
1138	C818 86 10	DVEC5	LDA #INPFIL	INPUT FILE TYPE
1139 1140	C81A 8C 86 20	LC81A	CMPX #\$862Ø	SKIP TWO BYTES
1140		* PRINT	DEVICE NUMBER CHECK RAM HOOK	
1142	C81B 86 20	DVEC6	LDA #OUTFIL	OUTPUT FILE TYPE
1143 1144	C81D ØD 6F C81F 2F B5		TST DEVNUM BLE LC7D6	* CHECK DEVICE NUMBER AND RETURN IF * NOT A DISK FILE
1145	C821 AF E4		STX ,S	= REPLACE SUBROUTINE RETURN ADDRESS WITH X REGISTER -
1146	C000 DD C7 14	*		= THIS IS THE SAME AS LEAS 2,S AND PSHS X
1147 1148	C823 BD C7 14 C826 34 Ø2		JSR LC714 PSHS A	POINT X TO FCB SAVE FILE TYPE ON STACK
1149	C828 A6 ØØ		LDA FCBTYP,X	GET FILE TYPE
1150	C82A 10 27 DB CD		LBEQ LA3FB	'FILE NOT OPEN' ERROR
1151 1152	C82E 81 40 C830 27 04		CMPA #RANFIL BEQ LC836	RANDOM FILE? BRANCH IF RANDOM FILE
-				

	C832 A1 E4	CMPA		IS THIS FCB OF THE PROPER TYPE?
	C834 26 DF C836 35 92		LC815 S A,X,PC	'FILE MODE' ERROR IF NOT RETURN
1156	0000 00 02	* DEVICE NUM	BER VALIDITY CHECK RAM HOOK	RETORN
1157	C838 2F 9C	DVEC1 BLE		RETURN IF NOT A DISK FILE
1158 1159	C83A F1 Ø9 5B C83D 1Ø 22 DD DE		3 FCBACT [ LA61F	COMPARE DEVICE NUMBER TO HIGHEST POSSIBLE 'DEVICE NUMBER' ERROR IF TOO BIG
	C841 35 9Ø		S X,PC	RETURN
1161				
1162 1163	C843 ØD 6F	* SET PRINT I DVEC2 TST	PARAMETERS RAM HOOK	*CHECK DEVICE NUMBER AND
	C845 2F 8F		LC7D6	*RETURN IF NOT DISK FILE
	C847 32 62	LEAS	S \$02,S	PURGE RETURN ADDRESS OFF OF THE STACK
	C849 34 16		5 X,B,A	SAVE REGISTERS
1167 1168	C84B ØF 6E C84D BD C7 14		PRTDEV LC714	SET PRINT DEVICE NUMBER TO NON-CASSETTE POINT X TO FCB
1109	COOM ED NO	LDB	FCBPOS,X	GET PRINT POSITION
	C852 4F	CLRA		PRINT WIDTH (256)
1171 1172	C853 8E 10 00 C856 7E A3 7C		#\$1000 LA37C	TAB FIELD WIDTH AND TAB ZONE SAVE THE PRINT PARAMETERS
1173	0000 72 710 70	0	2.107 0	ONTE THE TRAIN TANNIE ENO
1174		* BREAK CHECK		
1175 1176	C859 ØD 6F C85B 2F Ø2	DVEC11 TST	LC85F	* CHECK DEVICE NUMBER AND RETURN * IF NOT A DISK FILE
1177	C85D 32 62		5 \$02,S	= PURGE RETURN ADDRESS OFF OF THE STACK - DON'T
	C85F 39	LC85F RTS		= DO A BREAK CHECK IF DISK FILE
1179 1180		* EOF RAM HO	nν	
	C86Ø 32 62	DVEC14 LEAS		PURGE RETURN ADDRESS OFF OF THE STACK
1182	C862 96 6F	LDA	DEVNUM	* GET DEVICE NUMBER AND SAVE
	C864 34 Ø2	PSHS	S A	* IT ON THE STACK
	C866 BD A5 AE C869 BD A3 ED	JSR	LASED	STRIP DEVICE NUMBER OFF OF INPUT LINE VERIFY THAT THE FILE TYPE WAS 'INPUT'
	C86C ØD 6F	TST	DEVNUM	* CHECK DEVICE NUMBER AND
	C86E 10 2F DD 68	LBLE	E LA5DA	* BRANCH BACK TO BASIC'S EOF IF NOT DISK FILE
	C872 BD C7 14 C875 E6 ØØ	JSR	LC/14 FCRTVD V	POINT X TO FCB GET FILE TYPE
	C877 C1 4Ø	CMPB	B #RANFIL	RANDOM FILE?
	C879 27 9A	BEQ	A LASAE LASAE LASED DEWNUM E LASDA LC714 FCBTYP,X 3 #RANFIL LC815 3 FCBCFL,X LC884	'BAD FILE MODE' ERROR IF RANDOM
	C87B 5F C87C A6 88 10	CLRE	ECBCEL A	FILE NOT EMPTY FLAG - SET TO NOT EMPTY *CHECK THE CACHE FLAG - BRANCH IF
	C87F 26 Ø3	BNE	LC884	*THERE IS A CHARACTER WHICH HAS BEEN CACHED
	C881 E6 88 17		FCBDFL,X	GET SEQUENTIAL INPUT FILE STATUS
1196 1197	C884 7E A5 E4	LC884 JMP	LA5E4	LINK BACK TO BASIC'S EOF STATEMENT
1198		* GET FILENAN	ME/EXTENSION: DRIVE NUMBER FROM BAS	SIC
	C887 8E C2 91	LC887 LDX	#DEFEXT	POINT TO ' ' BLANK (DEFAULT) EXTENSION
	C88A 6F E2		,-S	CLEAR A BYTE ON STACK FOR USE AS A DRIVES FLAG
1201 1202	C88C B6 Ø9 5A C88F 97 EB		DEFDRV DCDRV	* GET DEFAULT DISK NUMBER * STORE IN DSKCON PARAMETER
	C891 CE Ø9 4C		#DNAMBF	DISK FILENAME BUFFER
	C894 CC 2Ø Ø8		#\$2008	STORE 8 BLANKS IN RAM (DEFAULT FILE NAME)
	C897 A7 CØ C899 5A	LC897 STA DECB		STORE A BLANK IN FILE NAME DECREMENT COUNTER
1207	C89A 26 FB	BNE	LC897	BRANCH IF NOT DONE
	C89C C6 Ø3		#\$Ø3	3 BYTES IN EXTENSION
	C89E BD A5 9A C8A1 BD 87 48		LA59A L8748	MOVE B BYTES FROM (X) TO (U) EVALUATE A STRING EXPRESSION
	C8A4 33 84	LEAU		POINT U TO START OF STRING
	C8A6 C1 Ø2		3 #\$Ø2	* CHECK LENGTH OF STRING AND  * BRANCH IF < 2
1213	C8A8 25 12 C8AA A6 41		LC8BC \$01,U	= GET 2ND CHARACTER IN STRING AND
1215	C8AC 81 3A	CMPA	A #':'	= CHECK FOR COLON
1216 1217	C8AE 26 ØC C8BØ A6 C4		LC8BC ,U	BRANCH IF NO DRIVE NUMBER * GET 1ST CHARACTER
1217	C8B2 81 3Ø	CMPA	, o \ #'Ø'	* IN STRING AND
1219	C8B4 25 Ø6	BLO	LC8BC	* CHECK TO SEE
	C8B6 81 33 C8B8 22 Ø2		\ #'3' LC8BC	* IF IT IS IN * THE RANGE 0-3
	C8BA 8D 33		LCSEF	* THE KANGE 0-3 GET DRIVE NUMBER
1223	C8BC 8E Ø9 4C	LC8BC LDX	#DNAMBF	POINT X TO FILE NAME BUFFER
	C8BF 5C C8CØ 5A	INCE LC8CØ DECE		COMPENSATE FOR DECB BELOW DECREMENT STRING LENGTH
	C8C1 26 ØC		LC8CF	BRANCH IF MORE CHARACTERS IN STRING
1227	C8C3 32 61	LEAS	\$ \$01,S	CLEAN UP STACK - REMOVE DRIVE FLAG
	C8C5 8C Ø9 4C		( #DNAMBF	POINTER STILL AT START OF BUFFER?
	C8C8 26 67 C8CA C6 3E		LC931 #2*31	RETURN IF NOT 'BAD FILENAME' ERROR IF NULL FILENAME
1231	C8CC 7E AC 46	JMP	LAC46	ERROR HANDLER
	C8CF A6 CØ	LC8CF LDA	,U+ \ #'.'	GET A CHARACTER FROM STRING
	C8D1 81 2E C8D3 27 2D		\	LOOK FOR PERIOD? YES
	C8D5 81 2F	•	A #'/'	SLASH?
	C8D7 27 29		LC902	YES
	C8D9 81 3A C8DB 27 Ø9		\ #':' LC8E6	COLON? YES
	C8DD 8C Ø9 54		( #DEXTBF	COMPARE POINTER TO END OF FILENAME BUFFER
1240	C8EØ 27 E8	BEQ	LC8CA	'BAD FILENAME' ERROR - FILENAME TOO LONG
	C8E2 8D 3E C8E4 2Ø DA		LC922 LC8CØ	PUT A CHARACTER IN FILENAME GET ANOTHER CHARACTER FROM STRING
	C8E6 8D DD		LC8C5	'BAD FILENAME' ERROR IF NO FILENAME YET
1244	C8E8 8D Ø5	BSR	LC8EF	GET DRIVE NUMBER
	C8EA 5D	TSTE		* CHECK LENGTH OF STRING
	C8EB 26 DD C8ED 35 82		LC8CA S A,PC	* 'BAD FILENAME' ERROR IF MORE CHARACTERS LEFT REMOVE DRIVES FLAG FROM STACK AND RETURN
1248			•	

1249		* GRAB DRIVE		
1250	C8EF 63 62	LC8EF COM		TOGGLE DRIVE FLAG
	C8F1 27 D7		LC8CA	'BAD FILENAME' ERROR IF DRIVE NUMBER DEFINED TWICE
	C8F3 A6 C1	LDA		ASCII VALUE OF DRIVE NUMBER TO ACCA
	C8F5 CØ Ø2	SUBB		DECREMENT STRING LENGTH BY 2 FOR DRIVE (:X)
	C8F7 8Ø 3Ø	SUBA BLO		SUBTRACT ASCII BIAS
	C8F9 25 CF C8FB 81 Ø3	CMPA		DRIVE NUMBER TOO LOW - 'BAD FILENAME' ERROR
				MAX OF 4 DRIVES DRIVE NUMBER TOO HIGH - 'BAD FILENAME' ERROR
	C8FD 22 CB C8FF 97 EB	BHI	DCDRV	STORE IN DSKCON DRIVE NUMBER
	C9Ø1 39	RTS	DCDRV	STORE IN DSCON DRIVE NUMBER
1260	0301 33	KIS		
1261		* GRAB EXTENS	ION	
1262	C902 8D C1	LC9Ø2 BSR	LC8C5	'BAD FILENAME' ERROR IF NO FILENAME YET
	C9Ø4 8E Ø9 57	LDX	#DNAMBF+11	POINT X TO END OF EXTENSION BUFFER
	C907 86 20		#SPACE	BLANK
	C9Ø9 A7 82	LC9Ø9 STA		*
	C9ØB 8C Ø9 54		#DEXTBF	* FILL EXTENSION WITH
	C9ØE 26 F9		LC909	* BLANKS (DEFAULT)
	C910 5A	LC91Ø DECB		DECREMENT STRING COUNTER
	C911 27 DA		LC8ED	RETURN IF ZERO
	C913 A6 CØ C915 81 3A	LDA CMPA		GET A CHARACTER FROM STRING *CHECK FOR DRIVE SEPARATOR
	C915 81 3A C917 27 CD	BEQ		* CHECK FOR DRIVE SEPARATOR
1272	C919 8C Ø9 57		#DNAMBF+11	=CHECK FOR END OF ESTENSION RAM BUFFER &
1274	C91C 27 AC	BEQ		='BAD FILENAME' ERROR IF EXTENSION TOO LONG
	C91E 8D Ø2	BSR		PUT A CHARACTER IN EXTENSION BUFFER
			LC910	GET ANOTHER EXTENSION CHARACTER
1277		5.51	-	
1278			ACTER INTO FILENAME OR EXTENSION	
	C922 A7 8Ø	LC922 STA		STORE CHARACTER IN FILENAME BUFFER
	C924 27 A4		LC8CA	'BAD FILENAME' ERROR; ZEROES ARE ILLEGAL
	C926 81 2E		#'.'	PERIOD?
	C928 27 AØ	BEQ		'BAD FILENAME' ERROR IF PERIOD
	C92A 81 2F	CMPA		SLASH?
	C92C 27 9C		LC8CA	'BAD FILENAME' ERROR IF SLASH
	C92E 4C	INCA		CHECK FOR \$FF
1286	C92F 27 99		LC8CA	'BAD FILENAME' ERROR IF \$FF
1288	C931 39	LC931 RTS		
1289		* SAVE COMMAN	D	
	C932 81 4D	SAVE CMPA		*
1291	C934 10 27 05 54	LBEQ	LCE8C	*BRANCH IF SAVEM
1292	C938 8D 4B	BSR		GO GET FILENAME, ETC. FROM BASIC
	C93A 9E 8A	LDX		ZERO OUT X REG
	C93C BF Ø9 57		DFLTYP	SET FILE TYPE AND ASCII FLAG TO ZERO
	C93F 9D A5		GETCCH	GET CURRENT INPUT CHARACTER FROM BASIC
	C941 27 21	BEQ		BRANCH IF END OF LINE
1297	C943 BD B2 6D	JSK	SYNCOMMA #'A'	SYNTAX CHECK FOR COMMA
	C946 C6 41 C948 BD B2 6F	JSR		*ASCII FILE? *SYNTAX CHECK ON CONTENTS OF ACCB
	C94B 26 E4		LC931	RETURN IF NO MORE CHARACTERS ON LINE
	C94D 73 Ø9 58		DASCFL	SET CRUNCHED/ASCII FLAG TO ASCII
	C950 8D 04		LC956	OPEN A SEQUENTIAL FILE FOR OUTPUT
13Ø3	C952 4F	CLRA		SET ZERO FLAG - CAUSE ENTIRE FILE TO BE LISTED
1304	C953 7E B7 64	JMP	LIST	'LIST' THE FILE TO CONSOLE OUT
13Ø5				
1306			ENTIAL FILE FOR INPUT/OUTPUT - USI	E THE SYSTEM
1307	COE6 06 4E		AT THE TOP OF FCBS	OUTPUT FILE TYPE
	C956 86 4F C958 8C 86 49	LC956 LDA LC958 CMPX		SKIP TWO BYTES
	C959 86 49	LC959 LDA		INPUT FILE TYPE
	C95B F6 Ø9 5B		FCBACT	GET NUMBER OF RESERVED FILES CURRENTLY RESERVED
	C95E 5C	INCB		ADD ONE - USE ONE ABOVE HIGHEST RESERVED FCB
1313	C95F D7 6F	STB	DEVNUM	SAVE IT IN DEVICE NUMBER
1314	C961 7E C4 68		LC468	OPEN A FILE & INITIALIZE FCB
1315			CHED FILE - A PREAMBLE OF THREE BY	
1316	0004 00 50		1 = \$FF, 2,3 = LENGTH OF BASIC PI	
	C964 8D FØ C966 86 FF	LC964 BSR	LC956 #\$FF	OPEN A SEQUENTIAL FILE FOR OUTPUT
	C968 BD CB 52		#\$FF LCB52	BASIC FILE FLAG CONSOLE OUT
	C968 BD CB 52		VARTAB	LOAD ACCD WITH START OF VARIABLES
	C96D 93 19		TXTTAB	SUBTRACT START OF BASIC
			LCB52	CONSOLE OUT FILE LENGTH MS BYTE
1323	C96F BD CB 52 C972 1F 98	TFR		PULL LS BYTE INTO ACCA
1324	C974 BD CB 52	JSR	LCB52	CONSOLE OUT FILE LENGTH LS BYTE
1325	C977 9E 19	LDX	TXTTAB	POINT X TO START OF BASIC
1326	C979 A6 80 C97B BD CB 52	LC979 LDA	, Χ+	GET BYTE FROM BASIC
		JSR	LUB52	SEND TO CONSOLE OUT
	C97E 9C 1B	CMPX	VARTAB	COMPARE TO END OF BASIC
	C98Ø 26 F7	RNF	LU3/3	KEEP GOING IF NOT AT END
	C982 7E A4 2D C985 8E C2 8E	BNE JMP LC985 LDX	#RASFYT	CLOSE FILE POINT TO 'BAS' EXTENSION (DEFAULT)
	C988 7E C8 8A	.TUX	LC88A	GET FILENAME.EXT FROM BASIC
1333	0300 /L 00 0A	UIT	20007	GET TEENMIETENT THOSE BASIC
1334		* MERGE COMMA	ND	
	C98B 4F	MERGE CLRA		RUN FLAG (Ø = DON'T RUN)
	C98C C6 FF	LDB		MERGE FLAG (\$FF = MERGE)
1337	C98E 2Ø 12	BRA	LC9A2	GO LOAD THE FILE
1338				
1339	C99Ø 81 22	* RUN RAM VEC DVEC18 CMPA	IOR #!"!	CHECK EUD EILENAME DELIMITED (DOUBLE OUOTE)
	C990 81 22 C992 10 26 B9 06		XVEC18	CHECK FOR FILENAME DELIMITER (DOUBLE QUOTE) NONE - JUMP TO EXBAS RUN RAM HOOK
	C996 86 Ø2	LDA		RUN FLAG - DON'T CLOSE ALL FILES BEFORE RUN
	C998 20 07		LC9A1	LOAD THE FILE
1344		2.01	<del>-</del>	

1345		* LOAD 0			
1346	C99A 81 4D	LOAD	CMPA		*
1347	C99C 10 27 05 45 C9A0 4F			LCEE5	*BRANCH IF LOADM
1348 1349	C9A1 5F	LC9A1	CLRA CLRB		RUN FLAG = ZERO (DON'T RUN) CLEAR MERGE FLAG
1350	C9A2 B7 Ø9 59	LC9A2		DRUNFL	RUN FLAG (Ø = DON'T RUN, 2 = RUN)
1351	C9A5 F7 Ø9 5E			DMRGFL	MERGE FLAG (Ø = NO MERGÉ, \$FF = MERGE)
1352	C9A8 8D DB			LC985	GO GET FILENAME, ETC. FROM BASIC
1353	C9AA 9D A5			GETCCH	GET CURRENT INPUT CHAR
1354	C9AC 27 10			LC9BE	BRANCH IF END OF LINE
1355 1356	C9AE BD B2 6D C9B1 C6 52			SYNCOMMA #'R'	SYNTAX CHECK FOR COMMA
1357	C9B3 BD B2 6F			LB26F	*IS NEXT CHAR 'R'? RUN AFTER LOAD
1358	C9B6 BD A5 C7			LA5C7	SYNTAX ERROR IF ANY MORE CHARS ON LINE
1359	C9B9 86 Ø3			#\$03	*SET FLAGS TO RUN AND CLOSE ALL FILES
1360	C9BB B7 Ø9 59			DRUNFL	*BEFORE THE FILE IS RUN
1361	C9BE 8D 99	LC9BE		LC959	GRAB FCB FOR INPUT FILE
1362 1363	C9CØ B6 Ø9 58			DASCFL	*CHECK ASCII FLAG AND BRANCH
1364	C9C3 27 ØB C9C5 7D Ø9 5E			LC9DØ DMRGFL	*IF CRUNCHED BASIC FILE IS THIS A MERGE?
1365	C9C8 26 Ø3			LC9CD	BRANCH IF MERGE
1366	C9CA BD AD 19			LAD19	DO A 'NEW' - ERASE VARIABLES, RESET VARIABLES
1367	C9CD 7E AC 7C	LC9CD	JMP	LAC7C	GO TO BASIC'S MAIN LOOP, IT WILL LOAD PROGRAM
1368					
1369 1370	C9DØ B6 Ø9 57	* LUAD 1		RUNCHED BASIC FILE DFLTYP	*CHECK FILE TYPE (MUST BE BASIC:0) & CHECK
1371	C9D3 BA Ø9 5E	LCJUU		DMRGFL	*MERGE FLAG (MUST BE NO MERGE: Ø)
1372	C9D6 10 26 DC 3C			LA616	'BAD FILE MODE' ERROR IF MERGE OR NON-BASIC
1373	C9DA BD AD 19			LAD19	DO A 'NEW' - RESET POINTERS, ERASE VARIABLES
1374	C9DD 73 Ø9 5D		COM	DLODFL	* SET THE LOAD FLAG TO \$FF - THIS WILL CAUSE A NEW TO
1375	0050 00 00 55	*	10-	1.0050	* OCCUR IF AN ERROR OCCURS WHILE THE PROGRAM IS BEING LOADED
1376	C9EØ BD CC E2			LCCE2	GET CHAR FROM BUFFER - SHOULD BE \$FF
1377 1378	C9E3 BD CC E2 C9E6 34 Ø2		PSHS	LCCE2 A	GET ANOTHER - MS BYTE OF LENGTH SAVE MS BYTE ON STACK
1379	C9E8 BD CC E2			LCCE2	LS BYTE OF LENGTH OF PROGRAM
1380	C9EB 1F 89		TFR		PUT LS BYTE INTO ACCB
1381	C9ED 35 Ø2		PULS	A	NOW ACCD CONTAINS LENGTH OF PROGRAM
1382	C9EF D3 19			TXTTAB	ADD BEGINNING OF BASIC
1383	C9F1 BD AC 37			LAC37	SEE OF ENOUGH ROOM IN RAM FOR THIS FILE
1384 1385	C9F4 9E 19 C9F6 BD C5 97	LC9F6		TXTTAB LC597	GET START OF BASIC READ A CHAR FROM CONSOLE IN
1386	C9F9 D6 7Ø	LUFFU		CINBFL	BUFFER EMPTY?
1387	C9FB 26 Ø4			LCAØ1	BRANCH IF SO
1388	C9FD A7 8Ø			, Χ+	STORE CHAR
1389	C9FF 2Ø F5		BRA	LC9F6	GET ANOTHER CHARACTER
139Ø 1391	CAØ1 7F Ø9 5D	LCAØ1	CLD	DLODFL	CLEAR LOAD FLAG - LOAD WAS ERROR FREE
1391	CAØ4 9F 1B	LUMBI		VARTAB	SAVE NEW START OF VARIABLES
1393	0.0.1 3. 15	* MAKE S		AST THREE BYTES LOADED WERE ZERO	ONTE HER OTHER OF THE PROCESS
1394	CAØ6 C6 Ø3			#\$03	CHECK THREE BYTES
1395	CAØ8 A6 82	LCAØ8	LDA	, -X	CHECK A BYTE
1396	CAØA 26 Ø3			LCAØF	BRANCH IF NON-ZERO
1397 1398	CAØC 5A CAØD 26 F9		DECB BNE	LCAØ8	DECREMENT COUNTER KEEP CHECKING IF NOT DONE
1399	CAØF 9E 1B	LCAØF		VARTAB	GET START OF VARIABLES
1400	CA11 9F 1B	LCA11		VARTAB	SAVE START OF VARIABLES
1401	CA13 6F 8Ø		CLR	, Χ+	CLEAR A BYTE
1402	CA15 5A		DECB		DECREMENT COUNTER
1403	CA16 2A F9			LCA11	KEEP CLEARING BYTES IF NOT DONE
1404	CA18 BD A4 2D CA1B BD AD 21	LCA18		LA42D	CLOSE SELECTED FILE
1405 1406	CA1E BD 82 9C			LAD21 XVEC18	DO PART OF NEW - ERASE VARIABLES, RESET INPUT PTR INITIALIZE EXBAS GRAPHICS VARIABLES
1407	CA21 BD AC EF			LACEF	RELOCATE ALL THE BASIC NEXT LINE POINTERS
1408	CA24 77 Ø9 59			DRUNFL	CHECK LSB OF RUN FLAG
1409	CA27 25 Ø3		BLO	LCA2C	BRANCH IF DON'T CLOSE ALL FILES
1410	CA29 BD A4 26			LA426	CLOSE ALL FILES
	CA2C 77 Ø9 59	LCA2C		DRUNFL	TEST BIT 1 OF RUN FLAG
	CA2F 10 25 E3 6B CA33 7E AC 73			LAD9E LAC73	BRANCH TO COMM INTERPRETATION LOOP IF BIT 1 SET RETURN TO DIRECT MODE
1414	555 / E NO / 5		0111	2.10.0	NEIGHT TO DIRECT HODE
	CA36 ØD 6F	DVEC13	TST	DEVNUM	* CHECK DEVICE NUMBER AND
	CA38 2E DE			LCA18	* TRY TO RUN FILE IF IT IS A DISK FILE
1417	CA3A 39		RTS		
1418		* (1005	ALL 51	LE DIECEDS DAM VECTOR	
1419 1420	CA3B F6 Ø9 5B	* CLUSE DVEC7		LE BUFFERS RAM VECTOR FCBACT	GET THE NUMBER OF RESERVED FILE BUFFERS
	CASE 5C	D. L. C. /	INCB	. 05.101	ADD ONE
1422	CA3F 34 Ø4	LCA3F	PSHS		SAVE IT
	CA41 D7 6F			DEVNUM	STORE IT IN DEVICE NUMBER
	CA43 8D ØE			LCA53	CLOSE FILE
	CA45 35 Ø4 CA47 5A		PULS DECB	В	GET BACK NUMBER OF FILE BUFFERS DECREMENT FILE BUFFER COUNTER
1426	CA47 5A CA48 26 F5			LCA3F	BRANCH IF ALL FILES NOT CLOSED
	CA4A 39	LCA4A	RTS		
1429					
1430		* CLOSE			
1431		DVEC8		DEVNUM	* CHECK DEVICE NUMBER AND RETURN
	CA4D 10 2F B8 35			XVEC8	* IF NOT A DISK FILE
	CA51 32 62 CA53 BD C7 14	LCA53		\$02,S LC714	PURGE RETURN ADDRESS OFF OF THE STACK POINT X TO CORRECT FCB
	CA56 ØF 6F	FOWDS			SET DEVICE NUMBER TO SCREEN
	CA58 9F F1	LCA58	STX	DEVNUM FCBTMP	SAVE FILE BUFFER POINTER
	CA5A A6 ØØ		LDA	FCBTYP,X	GET THE TYPE OF THIS FILE
	CA5C 27 EC			LCA4A	RETURN IF FILE NOT OPEN
	CASE 34 Ø2		PSHS		SAVE FILE TYPE
1440	CA60 6F 00		LLK	FCBTYP, X	CLOSE THE FILE - ZERO OUT THE FILE TYPE

	CA62 E6 Ø1		LDB FCBDRV,X	* GET DRIVE NUMBER AND
1442 1443	CA64 D7 EB		STB DCDRV	* SAVE IT IN DSKCON VARIABLE = CHECK FOR OUTPUT TYPE AND
1443	CA66 81 20 CA68 26 16		CMPA #OUTFIL BNE LCA8Ø	= BRANCH IF NOT OUTPUT TYPE FILE
1445	0.100 20 10		5112 257102	500000 10 00000 1012 1122
1446		* CLOSE	A SEQUENTIAL OUTPUT FILE	
1447	CA6A E6 88 18		LDB FCBLFT,X	GET THE NUMBER OF CHARACTERS IN BUFFER
1448	CA6D 86 80	_	LDA #\$8Ø	* SET THE PRE-SAVED BIT TO INDICATE THAT THE DATA
1449 1450	CA6F ED 88 13	•	STD FCBLST,X	* HAS ALREADY BEEN SAVED ON DISK SAVE THE NUMBER OF BYTES USED IN THE LAST SECTOR
	CA72 6C Ø4		INC FCBSEC,X	INCREMENT THE SECTOR NUMBER
	CA74 E6 Ø3		LDB FCBCGR,X	GET THE CURRENT GRANULE NUMBER
1453	CA76 BD C7 25		JSR LC725	POINT X TO FILE ALLOCATION TABLE
	CA79 A7 Ø1		STA FAT1,X	GET THE CURRINT GRANULE NUMBER POINT X TO FILE ALLOCATION TABLE SET FAT DATA NOT VALID FLAG (ACCA ← Ø) ADD GRANULE OFFSET TO FAT POINTER
1455	CA7B 3A		ABX	ADD GRANULE OFFSET TO FAT POINTER
1456 1457	CA7C 6C Ø6	*	INC FATCON,X	* INCREMENT GRANULE DATA (ADD ONE SECTOR TO LAST * GRANULE) SKIP PAST THE SIX FAT CONTROL BYTES
	CA7E 20 71		BRA LCAF1	UPDATE FAT AND DIRECTORY
1459	CA8Ø 81 4Ø	LCA8Ø	CMPA #RANFIL	RANDOM FILE?
1460	CA82 26 6D		BNE LCAF1	NO - UPDATE FAT AND DIRECTORY
1461				
1462 1463	CA84 EC Ø9		A RANDOM FILE LDD FCBRLN,X	GET RECORD LENGTH
	CA86 AE ØB		LDX FCBBUF,X	POINT X TO RANDOM FILE BUFFER
	CA88 31 8B		LEAY D.X	POINT Y TO END OF RANDOM FILE BUFFER
1466	CA8A 34 36		PSHS Y,X,B,A	SAVE POINTERS ON STACK
	CA8C DE 1B		LDU VARTAB	GET START OF VARIABLES
	CASE 11 93 1D	LCA8E	CMPU ARYTAB	COMPARE TO START OF ARRAYS
	CA91 27 ØE CA93 A6 41		BEQ LCAA1 LDA \$01,U	BRANCH IF ALL VARIABLES CHECKED GET 2ND BYTE OF VARIABLE NAME
	CA95 33 42		LEAU \$02,U	MOVE POINTER TO START OF DESCRIPTOR
	CA97 2A Ø2		BPL LCA9B	BRANCH IF VARIABLE - NUMERIC
1473	CA99 8D 28		BSR LCAC3	ADJUST STRING VARIABLE IF IN RANDOM FILE BUFFER
	CA9B 33 45	LCA9B	LEAU \$05,U	MOVE POINTER TO NEXT VARIABLE
	CA9D 20 EF CA9F 35 40	LCAOE	BRA LCASE	PROCESS ANOTHER VARIABLE
	CA9F 35 40 CAA1 11 93 1F	LCA9F LCAA1	PULS U CMPU ARYEND	GET ADDRESS OF NEXT ARRAY TO U COMPARE TO END OF ARRAYS
	CAA4 27 3A	LUMMI	BEQ LCAEØ	BRANCH IF END OF ARRAYS
	CAA6 1F 3Ø		TFR U,D	* SAVE ARRAY START IN ACCD, ADD OFFSET
	CAA8 E3 42		ADDD \$02,U	* TO NEXT ARRAY AND SAVE ADDRESS OF
	CAAA 34 Ø6		PSHS B,A	* NEXT ARRAY ON THE STACK
	CAAC A6 41 CAAE 2A EF		LDA \$Ø1,U BPL LCA9F	GET 2ND LETTER OF VARIABLE NAME BRANCH IF NUMERIC
	CABØ E6 44		LDB \$04,U	GET THE NUMBER OF DIMENSIONS
	CAB2 58		ASLB	X2:2 BYTES PER DIMENSION
	CAB3 CB Ø5		ADDB #\$Ø5	5 BYTES CONSTANT PER ARRAY DESCRIPTOR
	CAB5 4F		CLRA	CLEAR MSB OF OFFSET - (ONLY 125 DIMENSIONS ALLOWED)
	CAB6 33 CB		LEAU D,U	POINT U TO START OF THIS ARRAY'S VARIABLES
	CAB8 11 A3 E4 CABB 27 E2	LUABS	CMPU ,S BEQ LCA9F	AT END OF THIS ARRAY? YES
	CABD 8D Ø4		BSR LCAC3	ADJUST STRING VARIABLE IF IN RANDOM FILE BUFFER
	CABF 33 45		LEAU \$Ø5,U	MOVE POINTER TO NEXT DESCRIPTOR
	CAC1 20 F5		BRA LCAB8	CHECK NEXT VARIABLE
1494		*	TO 055 15 4 070110 10 1001750	THE DANGEN STATE DUSTED ADEA. TO THE TO
1495 1496				IN THE RANDOM FILE BUFFER AREA. IF IT IS IT WILL BE DELETED. IF IT IS HIGHER IN THE RANDOM
1497				N QUESTION, THE LENGTH OF THE CURRENT
1498				ADDRESS OF THE STRING BECAUSE THE CURRENT
1499		* BUFFE	R IS BEING DELETED (CLOSED).	
	CAC3 AE 42	LCAC3	LDX \$02,U	POINT X TO START OF STRING
	CAC5 BC Ø9 48		CMPX RNBFAD BHS LCAD8	COMPARE TO START OF FREE RANDOM FILE BUFFER AREA
1502 1503	CAC8 24 ØE CACA AC 64		CMPX \$Ø4,S	RETURN IF > START OF FREE RANDOM FILE BUFFER AREA COMPARE TO START OF THIS FILE'S RANDOM BUFFER
1504	CACC 25 ØB		BLO LCAD9	BRANCH IF < START OF THIS FILE'S RANDOM BUFFER
1505		*		** BUG ** THIS SHOULD BE A BRANCH TO A RETURN
1506	CACE AC 66		CMPX \$06,S	COMPARE TO END OF THIS FILE'S RANDOM BUFFER
1507	CADØ 25 Ø7		BLO LCAD9	BRANCH IF < END OF THIS FILE'S RANDOM BUFFER
	CAD2 1F 10 CAD4 A3 62		TFR X,D SUBD \$02,S	SAVE POINTER IN ACCD SUBTRACT RECORD LENGTH FROM START OF STRING ADDRESS
	CAD4 A3 62 CAD6 ED 42		STD \$02,U	SAVE NEW START OF STRING ADDRESS
	CAD8 39	LCAD8	RTS	
	CAD9 6F C4	LCAD9	CLR ,U	CLEAR THE LENGTH OF THE STRING
	CADB 6F 42		CLR \$02,U	* CLEAR THE ADDRESS
	CADD 6F 43		CLR \$03,U	* OF THE STRING
1515 1516	CADF 39	*DEMOVE	RTS	BUFFER FOR A 'CLOSED' RANDOM FILE
	CAEØ 35 56		PULS A,B,X,U	*U = END OF RANDOM FILE BUFFER, X = START OF RANDOM
1518		*		*FILE BUFFER, ACCD = RECORD LENGTH
1519				
1520				NVENIENT PLACE TO FIX THE BUG WHICH
1521 1522		** CAUS	ES THE SYSTEM TO HANG IF AN ER	ROR IS ENCOUNTERED DURING 'COPY'
1522		*	CMPU FCBADR	* IS THE END OF THIS FCB'S BUFFER ABOVE THE END
1524		*		* OF THE START OF THE FCB AREA
1525		*	BLO LCAE2	NO - FREE UP THE SPACE USED BY THIS FILE IN RANDOM BUFFER
1526		*	LDX #DFLBUF	YES - DOING A 'COPY'; RESET START OF RANDOM BUFFER
1527		*	BRA LCAEE	
1528 1529		* RANDO	M FILE BUFFER AREA	
1529 1530		* REMOV	E RESERVED SPACE FOR CLOSED FT	LE FROM RANDOM FILE BUFFER SPACE
	CAE2 11 B3 Ø9 48	LCAE2		AT THE BOTTOM OF FREE RANDOM BUFFER AREA?
	CAE6 27 Ø6		BEQ LCAEE	BRANCH IF THERE
			LDA III	= GRAB A SOURCE BYTE AND
1533	CAE8 A6 CØ		LDA ,U+	
1533 1534	CAE8 A6 CØ CAEA A7 8Ø		STA ,X+	= MOVE IT TO DESTINATION
1533 1534 1535	CAE8 A6 CØ	LCAEE		

1537	CAF1 BD C7 25	LCAF1	JSR LC725	POINT X TO PROPER FILE ALLOCATION TABLE
1538	CAF4 6A ØØ		DEC FATØ,X	REMOVE ONE ACTIVE FILE
1539	CAF6 6D Ø1		TST FAT1,X	NEW DATA IN FAT RAM IMAGE?
	CAF8 27 Ø3 CAFA BD C6 F1		JSR LC6F1	NO WRITE OUT FILE ALLOCATION TABLE TO DISK
	CAFD 9E F1	LCAFD	LDX FCBTMP	GET FILE BUFFER POINTER
	CAFF 35 Ø2		PULS A	GET FILE TYPE
	CBØ1 81 2Ø		CMPA #OUTFIL	IS IT A SEQUENTIAL OUTPUT FILE?
	CBØ3 27 Ø8		BEQ LCBØD	YES
1546 1547	CBØ5 81 4Ø CBØ7 26 CF		CMPA #RANFIL BNE LCAD8	IS IT A RANDOM FILE? RETURN IF NOT A RANDOM FILE (SEQUENTIAL INPUT)
1548	CBØ9 A6 ØF		LDA FCBFLG,X	* TEST THE GET/PUT FLAG AND
1549	CBØB 27 ØA		BEQ LCB17	* BRANCH IF 'GET'
1550 1551		+ UDITE	CONTENTS OF FILE BUFFER TO DISK	
	CBØD BD C7 33	LCBØD	CONTENTS OF FILE BUFFER TO DISK JSR LC733	GET PROPER TRACK & SECTOR NUMBERS
	CB10 33 88 19		LEAU FCBCON,X	POINT U TO START OF FCB DATA
1554	CB13 DF EE		STU DCBPT	SET UP FILE BUFFER POINTER FOR DSKCON
1555 1556	CB15 8D 2C CB17 A6 88 13	LCR17	BSR LUB43	GO WRITE A SECTOR CHECK THE PRE-SAVED FLAG
1557	CB1A 2A BC	LUDIT	BPL LCAD8	RETURN IF RECORD HAS ALREADY BEEN SAVED ON DISK
	CB1C E6 88 12		LDB FCBDIR,X	GET DIRECTORY NUMBER OF THIS FILE
	CB1F C4 Ø7		ANDB #\$07	8 ENTRIES PER SECTOR
	CB21 86 20 CB23 3D		LDA #DIRLEN MUL	DIRLEN BYTES PER DIRECTORY ENTRY GET SECTOR OFFSET FOR THIS ENTRY
	CB24 CE Ø6 ØØ		LDU #DBUFØ	* GET READ/WRITE BUFFER Ø AND
	CB27 DF EE		STU DCBPT	* SAVE IT IN DSKCON REGISTER
	CB29 31 CB		LEAY D,U LDB FCBDIR,X	Y POINTS TO CORRECT DIRECTORY ENTRY
	CB2B E6 88 12 CB2E 54		LDB FCBDIR,X LSRB	GET DIRECTORY ENTRY NUMBER *
	CB2F 54		LSRB	*
1568	CB3Ø 54		LSRB	* DIVIDE BY 8; EIGHT DIRECTORY ENTRIES PER SECTOR
	CB31 CB Ø3		ADDB #\$Ø3	ADD BIAS; FIRST 3 SECTORS NOT DIRECTORY
	CB33 D7 ED CB35 CC 11 Ø2		STB DSEC LDD #\$1102 STA DCTRK	STORE SECTOR NUMBER DIRECTORY TRACK - READ OP CODE
	CB38 97 EC		STA DCTRK	STORE TRACK NUMBER
	CB3A 8D Ø9		BSR LCB45	GO READ DIRECTORY
	CB3C EC 88 13		LDD FCBLST, X	GET NUMBER OF BYTES IN THE LAST SECTOR
	CB3F 84 7F CB41 ED 2E		STA DCTRK BSR LCB45 LDD FCBLST,X ANDA #\$7F STD DIRLST,Y LDB #\$03 STB DCOCC	MASK OFF THE PRE-SAVED FLAG SAVE NUMBER OF BYTES IN LAST SECTOR OF FILE IN DIRECTORY
	CB43 C6 Ø3	LCB43	LDB #\$03	WRITE OP CODE
	CB45 D7 EA	LCB45	31B DCOFC	SAVE DSKCON OP CODE VARIABLE
1579 1580	CB47 7E D5 FF		JMP LD5FF	GO READ/WRITE SECTOR
1581		* CONSOL	E OUT RAM HOOK TST DEVNUM	
1582	CB4A ØD 6F	DVEC3		CHECK DEVICE NUMBER
1583 1584	CB4C 10 2F B7 23 CB50 32 62		LBLE XVEC3 LEAS \$02,S	BRANCH TO EX BASIC IF NOT A DISK FILE POP RETURN OFF STACK
1585		* SEND A	CHARACTER IN ACCA TO A DISK FILE. A	CARRIAGE RETURN WILL RESET THE
1586		* PRINT	POSITION AND CONTROL CODES WILL NOT I	INCREMENT THE PRINT POSITION.
1587 1588	CB52 34 16 CB54 8E Ø9 26	LCB52	PSHS X,B,A LDX #FCBV1-2	SAVE REGISTERS POINT X TO TABLE OF FILE NUMBER VECTORS
	CB57 D6 6F		LDB DEVNUM	GET CURRENT FILE NUMBER
1590	CB59 58		ASLB	2 BYTES PER FCB ADDRESS
	CB5A AE 85		LDX B,X LDB FCBTYP,X	POINT X TO PROPER FCB
	CB5C E6 84 CB5E C1 10		CMPB #INPFIL	GET FILE TYPE IS IT AN INPUT FILE?
	CB60 27 36		BEQ LCB98	RETURN IF SO
	CB62 81 ØD		CMPA #CR	CARRIAGE RETURN (ENTER)
	CB64 26 Ø2 CB66 6F Ø6		BNE LCB68 CLR FCBPOS,X	NO CLEAR PRINT POSITION IF CARRIAGE RETURN
		LCB68	CMPA #SPACE	*
1599	CB6A 25 Ø2		CITE #31 ACL	
1600			BLO LCB6E	*BRANCH IF CONTROL CHAR
	CB6C 6C Ø6		BLO LCB6E INC FCBPOS,X	INCREMENT PRINT POSITION
16Ø1 16Ø2		LCB6E	BLO LCB6E	
1601 1602 1603	CB6C 6C Ø6 CB6E C1 4Ø CB7Ø 26 1A	LCB6E	BLO LCB6E INC FCBPOS,X CMPB #RANFIL BNE LCB8C BYTE INTO A RANDOM FILE	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM
1601 1602 1603	CB6C 6C Ø6 CB6E C1 4Ø CB7Ø 26 1A	LCB6E	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM GET 'PUT' BYTE COUNTER
1601 1602 1603	CB6C 6C Ø6 CB6E C1 4Ø CB7Ø 26 1A	LCB6E * PUT A	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM
1601 1602 1603 1604 1605 1606	CB6C 6C 06 CB6E C1 40 CB70 26 1A CB72 EC 88 17 CB75 C3 00 01 CB78 10 A3 09	LCB6E * PUT A	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH
1601 1602 1603 1604 1605 1606 1607 1608	CB6C 6C 06 CB6E C1 40 CB70 26 1A CB72 EC 88 17 CB75 C3 00 01 CB78 10 A3 09 CB78 10 22 01 72 CB7F ED 88 17	LCB6E * PUT A	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER  ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH  SAVE NEW 'PUT' BYTE COUNTER
1601 1602 1603 1604 1605 1606 1607 1608 1609	CB6C 6C Ø6 CB6E C1 40 CB70 26 1A CB72 EC 88 17 CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB7B 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 88	LCB6E * PUT A	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER
1601 1602 1603 1604 1605 1606 1607 1608 1609	CB6C 6C Ø6 CB6E C1 40 CB70 26 1A CB72 EC 88 17 CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB7B 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 88	LCB6E * PUT A	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER  ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH  SAVE NEW 'PUT' BYTE COUNTER
1601 1602 1603 1604 1605 1606 1607 1608 1609	CB6C 6C Ø6 CB6E C1 40 CB70 26 1A CB72 EC 88 17 CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB7B 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 88	LCB6E * PUT A	BLO LCB6E INC FCBPOS,X CMPB #RAWFIL BNE LCB8C BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRLN,X LBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER
1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613	CBGC 6C 06 CBGE C1 40 CB70 26 1A CB72 EC 88 17 CB75 C3 00 01 CB78 10 A3 09 CB7B 10 22 01 72 CB7F ED 88 17 CB82 AE 0B	LCB6E * PUT A	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCB8C BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRLN,X LBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK
1601 1602 1603 1604 1605 1606 1607 1608 1609	CB6C 6C Ø6 CB6E C1 40 CB70 26 1A CB72 EC 88 17 CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB7B 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 88	LCB6E * PUT A	BLO LCB6E INC FCBPOS,X CMPB #RAWFIL BNE LCB8C BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRLN,X LBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER
1601 1602 1603 1604 1605 1606 1607 1608 1610 1611 1612 1613 1614 1615	CB6C 6C Ø6 CB6E C1 40 CB70 26 1A  CB72 26 1A  CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB78 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 Ø8 CB86 35 Ø2 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18	LCB6E * PUT A  * WRITE	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRLN,X LBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT
1601 1602 1603 1604 1605 1606 1607 1610 1611 1612 1613 1614 1615 1616 1617	CB6C 6C Ø6 CB6E C1 40 CB70 26 1A  CB72 EC 88 17 CB75 C3 00 01 CB78 10 A3 09 CB7B 10 22 01 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 8B CB84 30 8B CB86 35 02 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB8F E6 88 18	LCB6E * PUT A  * WRITE	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCB8C BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRLN,X LBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS B,X,PC A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X LDB FCBLFT,X LDB FCBLFT,X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO ANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT AND BRANCH
1601 1602 1603 1604 1605 1606 1607 1608 1610 1611 1612 1613 1614 1615 1616 1617 1618	CB6C 6C Ø6 CB6E C1 40 CB70 26 1A  CB72 EC 88 17 CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB78 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 Ø8 CB86 35 Ø2 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB9F E6 88 18	LCB6E * PUT A  * WRITE LCB8C	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRLN,X LBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT
1601 1602 1603 1604 1605 1606 1607 1608 1610 1611 1612 1613 1614 1615 1616 1617 1618	CB6C 6C Ø6 CB6E C1 40 CB70 26 1A  CB72 EC 88 17 CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB78 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 Ø8 CB86 35 Ø2 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB9F E6 88 18	LCB6E * PUT A  * WRITE LCB8C	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCB8C BYTE INTO A RANDOM FILE LOD FCBPUT,X ADDD #\$0001 CMPD FCBRLN,X LBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS B,X,PC A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X LDB FCBLFT,X BEQ LCB9A ABX STA FCBCON-1,X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT * THE BUFFER IS FULL
1601 1602 1603 1604 1605 1606 1607 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621	CB6C 6C 06 CB6E C1 40 CB70 26 1A CB72 EC 88 17 CB75 C3 00 01 CB78 10 A3 09 CB78 10 22 01 72 CB7F ED 88 17 CB82 AE 08 CB84 30 88 CB84 30 88 CB86 A7 1F CB8A 35 94 CB8C 6C 88 18 CB8C 6C 88 18 CB8F E6 88 18	LCB6E * PUT A  * WRITE LCB8C	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRUN,X LBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS A STA -1,X PULS B,X,PC A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X LDB FCBLFT,X BEQ LCB9A ABX	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT TO TOB ADDRESS
1601 1602 1603 1604 1605 1606 1607 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1621 1622	CBGC 6C 06 CBGE C1 40 CB70 26 1A CB72 EC 88 17 CB75 C3 00 01 CB78 10 A3 09 CB78 10 22 01 72 CB7F ED 88 17 CB82 AE 08 CB84 30 88 CB84 30 88 CB86 35 02 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB92 27 06 CB95 A7 88 18 CB95 A7 88 18 CB98 35 96	LCB6E * PUT A  * WRITE LCB8C	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRLN,X LBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS B,X,PC A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X LDB FCBLFT,X LDB FCBLFT,X BEQ LCB9A ABX STA FCBCON-1,X PULS A,B,X,PC	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT TO TOB ADDRESS
1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1622 1623	CBGC 6C Ø6 CBGE C1 40 CB70 26 1A  CB72 26 1A  CB72 26 1A  CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB76 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 39 Ø8 CB86 35 Ø2 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB9C 37 Ø6 CB94 3A CB95 A7 88 18 CB98 35 96	LCB6E * PUT A  * WRITE LCB8C  LCB98 * WRITE	BLO LCBGE INC FCBPOS,X CMPB #RAMFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRUN,X LEBHI LCCF1 STD FCBPUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS B,X,PC  A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X LDB FCBLFT,X LCDB A BX STA FCBCON-1,X PULS A,B,X,PC  OUT A FULL BUFFER AND RESET BUFFER PSHS U,Y	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH 'SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT TO FOB ADDRESS STORE NEW CHARACTER (SKIP PAST 25 CONTROL BYTES AT FCB START)  SAVE REGISTERS
1601 1602 1603 1604 1605 1606 1607 1608 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1623	CBGC 6C 96 CBGE C1 40 CB70 26 1A  CB72 EC 88 17 CB75 C3 00 01 CB78 10 A3 09 CB78 10 A3 09 CB76 EB 88 17 CB82 AE 08 CB84 39 88 CB86 35 02 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB92 27 96 CB94 3A CB95 A7 88 18 CB94 3A CB95 A7 88 18 CB98 35 96  CB9A 34 60 CB9A 7 89 01 18	LCB6E * PUT A  * WRITE LCB8C  LCB98 * WRITE	BLO LCBGE INC FCBPOS, X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT, X ADDD #\$0001 CMPD FCBRLN, X LBHI LCCF1 STD FCBPUT, X LDX FCBBUF, X LEAX D, X PULS A STA -1, X PULS A STA -1, X PULS B, X, PC A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT, X LDB FCBLFT, X BEQ LCB9A ABX STA FCBCON-1, X PULS A,B, X, PC OUT A FULL BUFFER AND RESET BUFFER PSHS U, Y STA SECLEN+FCBCON-1, X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT TO FOB ADDRESS STORE NEW CHARACTER (SKIP PAST 25 CONTROL BYTES AT FCB START)  SAVE REGISTERS STORE LAST CHARACTER IN BUFFER
1601 1602 1603 1604 1605 1606 1607 1608 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625	CBGC 6C 06 CBGE C1 40 CBGE C1 40 CB70 26 1A  CB72 EC 88 17 CB75 C3 00 01 CB78 10 A3 09 CB7B 10 22 01 72 CB7F ED 88 17 CB82 AE 08 CB84 30 8B CB86 35 02 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB95 A7 88 18 CB92 27 06 CB94 3A CB95 A7 88 18 CB95 A7 88 18 CB98 35 96  CB9A 34 60 CB9A 34 60 CB9C A7 89 01 18 CB8C BA 06 CB9C A7 89 01 18 CB8C BA 06 CB9C A7 89 01 18 CB8C BB D C6 58	* WRITE LCB8C  * WRITE LCB8C	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRIN,X LBHI LCCF1 STD FCBBUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS B,X,PC A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X BEQ LCB9A ABX STA FCBCON-1,X PULS A,B,X,PC  OUT A FULL BUFFER AND RESET BUFFER PSHS U,Y STA SECLEN+FCBCON-1,X JSR LCG58	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO ANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT TO FCB ADDRESS STORE NEW CHARACTER (SKIP PAST 25 CONTROL BYTES AT FCB START)  SAVE REGISTERS STORE LAST CHARACTER IN BUFFER INCREMENT RECORD NUMBER
1601 1602 1603 1604 1605 1606 1607 1608 1609 1610 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1626 1627 1628	CBGC 6C 96 CBGE C1 40 CB70 26 1A  CB72 26 1A  CB75 C3 90 91 CB78 10 A3 99 CB78 10 22 91 72 CB76 EB 88 17 CB82 AE 98 CB84 39 8B CB86 35 92 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB92 27 96 CB94 3A CB95 A7 88 18 CB95 A7 88 18 CB96 A7 89 91 18 CB90 A7 89 91 18 CB90 A7 89 91 18 CBA3 E6 91 CBA3 E6 91 CBA5 D7 EB	* WRITE LCB9A	BLO LCBGE INC FCBPOS, X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT, X ADDD #\$0001 CMPD #FCBRUN, X LBHI LCCF1 STD FCBPUT, X LDX FCBBUF, X LEAX D, X PULS A STA -1, X PULS B, X, PC A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT, X LDB FCBLFT, X BEQ LCB9A ABX STA FCBCON-1, X PULS A,B, X, PC OUT A FULL BUFFER AND RESET BUFFER PSHS U,Y STA SECLEN+FCBCON-1, X JSR LC658 LDB FCBDRY, X STB CDCRY	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT AND BRANCH * IF THE BUFFER IS FULL ADD CHARACTER COUNT TO FGB ADDRESS STORE NEW CHARACTER (SKIP PAST 25 CONTROL BYTES AT FCB START)  SAVE REGISTERS STORE LAST CHARACTER IN BUFFER INCREMENT RECORD NUMBER * GET DRIVE NUMBER AND SAVE * IT IN DSKCON CONTROL TABLE
1601 1602 1603 1604 1607 1608 1610 1611 1612 1613 1614 1615 1616 1617 1618 1620 1621 1622 1623 1624 1625 1626 1627 1626	CBGC 6C Ø6 CBGE C1 40 CBGE C1 40 CB70 26 1A  CB72 EC 88 17 CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB7B 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 8B CB84 30 8B CB86 35 Ø2 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB95 A7 88 18 CB92 27 Ø6 CB94 3A CB95 A7 88 18 CB95 A7 88 18 CB96 A7 89 Ø1 18 CB96 A7 89 Ø1 18 CBAØ ØD C6 58 CBA3 E6 Ø1 CBA3 E6 Ø1 CBA5 FE E8 CBA3 E6 Ø1 CBA5 FE E8 CBA5 E6 Ø1 CBA6 FE E8 CBA6 E6 Ø1 CBA6 E6 Ø1 CBA7 E6 Ø1 CBA6 E6 Ø1 CBA7 E6 Ø1 CBA7 E6 Ø1 CBA7 E6 Ø4	* WRITE LCB9A	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRIN,X LBHI LCCF1 STD FCBRUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS B,X,PC A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X BEQ LCB9A ABX STA FCBCON-1,X PULS A,B,X,PC  OUT A FULL BUFFER AND RESET BUFFER PSHS U,Y STA SECLEN+FCBCON-1,X JSR LC658 LDB FCBDRV,X STB DCDRV INC FCBSEC,X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO ANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT AND BRANCH * IF THE BUFFER IS FULL ADD CHARACTER COUNT TO FCB ADDRESS STORE NEW CHARACTER (SKIP PAST 25 CONTROL BYTES AT FCB START)  SAVE REGISTERS STORE LAST CHARACTER IN BUFFER INCREMENT RECORD NUMBER * GET DRIVE NUMBER AND SAVE * IT IN DSKCON CONTROL TABLE INCREMENT SECTOR NUMBER
1601 1602 1603 1604 1605 1606 1607 1609 1610 1611 1612 1613 1614 1615 1616 1617 1621 1622 1623 1624 1625 1626 1627 1628 1628 1628 1630	CBGC 6C 96 CBGE C1 40 CB72 EC 88 17 CB75 C3 00 01 CB78 10 A3 09 CB78 10 A2 09 CB76 ED 88 17 CB82 AE 08 CB84 30 88 CB86 35 02 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB92 27 06 CB94 3A CB95 A7 88 18 CB94 3A CB95 A7 88 18 CB94 3A CB95 A7 88 18 CB94 A7 CB94 A8 CB95 A7 88 18 CB94 A7 CB95 A7 88 18 CB96 A7 89 01 18 CB40 BD C6 58 CB3 E6 01 CBA 60 04 CBA	* WRITE LCB9A	BLO LCBGE INC FCBPOS, X CMPB #RAMFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT, X ADDD #\$0001 CMPD FCBRUN, X LEBHI LCCF1 STD FCBPUT, X LDX FCBBUF, X LEAX D, X PULS A STA -1, X PULS B, X, PC  A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT, X LDB FCBLFT, X BEQ LCB9A ABX STA FCBCON-1, X PULS A, B, X, PC  OUT A FULL BUFFER AND RESET BUFFER PSHS U,Y STA SECLEN+FCBCON-1, X JSR LCGB0 LDB FCBDRY, X STB DCDRY INC FCBSEC, X JSR LCBBD	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO RANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT AND BRANCH * IF THE BUFFER IS FULL ADD CHARACTER COUNT TO FCB ADDRESS STORE NEW CHARACTER (SKIP PAST 25 CONTROL BYTES AT FCB START)  SAVE REGISTERS STORE LAST CHARACTER IN BUFFER INCREMENT RECORD NUMBER * GET DRIVE NUMBER AND SAVE * IT IN DSKCON CONTROL TABLE INCREMENT SECTOR NUMBER WEITE THE FILE BUFFER TO DISK
1601 1602 1603 1604 1605 1606 1607 1608 1611 1612 1613 1614 1615 1616 1617 1618 1619 1620 1621 1622 1623 1624 1625 1626 1627 1628 1629 1631	CBGC 6C Ø6 CBGE C1 40 CBGE C1 40 CB70 26 1A  CB72 EC 88 17 CB75 C3 Ø0 Ø1 CB78 10 A3 Ø9 CB7B 10 22 Ø1 72 CB7F ED 88 17 CB82 AE Ø8 CB84 30 8B CB84 30 8B CB86 35 Ø2 CB88 A7 1F CB8A 35 94  CB8C 6C 88 18 CB95 A7 88 18 CB92 27 Ø6 CB94 3A CB95 A7 88 18 CB95 A7 88 18 CB96 A7 89 Ø1 18 CB96 A7 89 Ø1 18 CBAØ ØD C6 58 CBA3 E6 Ø1 CBA3 E6 Ø1 CBA5 FE E8 CBA3 E6 Ø1 CBA5 FE E8 CBA5 E6 Ø1 CBA6 FE E8 CBA6 E6 Ø1 CBA6 E6 Ø1 CBA7 E6 Ø1 CBA6 E6 Ø1 CBA7 E6 Ø1 CBA7 E6 Ø1 CBA7 E6 Ø4	* WRITE LCB9A	BLO LCBGE INC FCBPOS,X CMPB #RANFIL BNE LCBBC BYTE INTO A RANDOM FILE LDD FCBPUT,X ADDD #\$0001 CMPD FCBRIN,X LBHI LCCF1 STD FCBRUT,X LDX FCBBUF,X LEAX D,X PULS A STA -1,X PULS B,X,PC A BYTE TO SEQUENTIAL OUTPUT FILE INC FCBLFT,X BEQ LCB9A ABX STA FCBCON-1,X PULS A,B,X,PC  OUT A FULL BUFFER AND RESET BUFFER PSHS U,Y STA SECLEN+FCBCON-1,X JSR LC658 LDB FCBDRV,X STB DCDRV INC FCBSEC,X	INCREMENT PRINT POSITION IS IT RANDOM FILE? BRANCH IF NOT RANDOM  GET 'PUT' BYTE COUNTER ADD ONE COMPARE TO RECORD LENGTH 'FR' ERROR IF 'PUT' BYTE COUNTER > RECORD LENGTH SAVE NEW 'PUT' BYTE COUNTER POINT TO ANDOM FILE BUFFER POINTER POINT TO ONE PAST END OF CURRENT RECORD DATA PULL DATA FROM STACK STORE IN DATA BUFFER RESTORE REGISTERS AND RETURN  INCREMENT CHARACTER COUNT * GET CHARACTER COUNT AND BRANCH * IF THE BUFFER IS FULL ADD CHARACTER COUNT TO FCB ADDRESS STORE NEW CHARACTER (SKIP PAST 25 CONTROL BYTES AT FCB START)  SAVE REGISTERS STORE LAST CHARACTER IN BUFFER INCREMENT RECORD NUMBER * GET DRIVE NUMBER AND SAVE * IT IN DSKCON CONTROL TABLE INCREMENT SECTOR NUMBER

1633	CBBØ BD C7 25		JSR	LC725	POINT X TO PROPER ALLOCATION TABLE
1634	CBB3 3A		ABX	ELTOON V	ADD THE GRANULE NUMBER TO FAT POINTER
1635 1636	CBB4 33 Ø6	*	LEAU	FATCON,X	*POINT U TO THE CORRECT GRANULE IN FAT - SKIP PAST *THE SIX FAT CONTROL BYTES
1637	CBB6 A6 24		LDA	FCBSEC,Y	GET CURRENT SECTOR FOR THIS GRANULE
1638	CBB8 81 Ø9			#\$Ø9	MAX SECTOR NUMBER (9 SECTORS/GRANULE)
1639	CBBA 25 Ø8			LCBC4	BRANCH IF NOT AT END OF GRANULE
1640	CBBC 6F 24			FCBSEC,Y	CLEAR SECTOR NUMBER
	CBBE BD C7 8F			LC78F FCBCGR,Y	GET NEXT FREE GRANULE
1643	CBC1 A7 23 CBC3 8C 8A CØ	LCBC3		#\$8ACØ	SAVE NEW GRANULE IN FCB SKIP TWO BYTES NO DATA STORED IN NEW SECTOR YET
1644	CBC4 8A CØ		ORA		FORCE GRANULE NUMBER TO BE FINAL GRANULE IN FILE
1645	CBC6 A7 C4			,U	STORE IN MAP
1646	CBC8 BD C5 7C			LC57C	UPDATE FILE ALLOCATION TABLE
1647 1648	CBCB 35 6Ø CBCD 35 96			Y,U A,B,X,PC	RESTORE REGISTERS RESTORE REGISTERS AND RETURN
1649	CDCD 33 90		TOLS	۸,۵,۸,۱۵	RESTORE REGISTERS AND RETORN
1650		* DIR C	OMMAND	1	
1651	CBCF BD D1 62	DIR		LD162	SCAN DRIVE NUMBER FROM INPUT LINE
1652 1653	CBD2 BD C7 6D CBD5 BD B9 58			LC76D LB958	GET FAT FOR THIS DRIVE PRINT CARRIAGE RETURN TO CONSOLE OUT
1654	CBD8 CC 11 Ø2			#\$1102	* GET TRACK 17 AND
1655	CBDB 97 EC			DCTRK	* READ OP CODE AND
1656	CBDD D7 EA			DCOPC	* SAVE IN DSKCON VARIABLES
1657	CBDF C6 Ø3		LDB	#\$Ø3	START WITH SECTOR 3 (FIRST DIRECTORY SECTOR)
1658 1659		* DEVD	A DIDE	CTORY SECTOR INTO THE I/O BUFFER	
1660	CBE1 D7 ED	LCBE1		DSEC	SAVE SECTOR NUMBER IN DSKCON VARIABLE
1661	CBE3 8E Ø6 ØØ			#DBUFØ	* USE I/O BUFFER Ø FOR DATA TRANSFER
1662	CBE6 9F EE			DCBPT	* SAVE IN DSKCON VARIABLE
1663 1664	CBE8 BD D5 FF		JSK	LD5FF	READ A SECTOR
1665		* SEND	DIRECT	ORY INFORMATION TO CONSOLE OUT	
	CBEB 35 4Ø	LCBEB	PULS		SAVE TOP OF STACK
	CBED BD A5 49			LA549	GO DO A BREAK CHECK
	CBFØ 34 4Ø		PSHS		RESTORE STACK
1669 1670	CBF2 A6 84 CBF4 27 38			DIRNAM,X LCC2E	TEST FILE NAME FIRST BYTE BRANCH IF KILLED
1671	CBF6 43		COMA		FF = END OF DIRECTORY
1672	CBF7 27 44		BEQ	LCC3D	RETURN IF END OF DIRECTORY
1673	CBF9 34 10		PSHS		SAVE DIRECTORY POINTER ON STACK
1674 1675	CBFB C6 Ø8 CBFD BD B9 A2			#\$Ø8 LB9A2	NUMBER CHARACTERS TO PRINT SEND FILENAME TO CONSOLE OUT
	CCØØ 8D 3F			LCC41	SEND BLANK TO CONSOLE OUT
1677	CCØ2 C6 Ø3		LDB	#\$03	NUMBER CHARACTERS TO PRINT
1678	CCØ4 BD B9 A2			LB9A2	SEND EXTENSION TO CONSOLE OUT
1679 1680	CCØ7 8D 38 CCØ9 E6 ØØ			LCC41 FCBTYP,X	SEND BLANK TO CONSOLE OUT GET FILE TYPE
	CCØB C1 ØA			#10	* CHECK THE NUMBER OF DECIMAL DIGITS IN
	CCØD 24 Ø2			LCC11	* ACCB: IF THERE IS ONLY ONE DIGIT,
	CCØF 8D 3Ø			LCC41	* SEND BLANK TO CONSOLE OUT
	CC11 4F	LCC11			CLEAR MS BYTE OF ACCO
1685 1686	CC12 BD BD CC CC15 8D 2A			LBDCC LCC41	PRINT ACCD IN DECIMAL TO CONSOLE OUT SEND BLANK TO CONSOLE OUT
1687	CC17 AE E4		LDX		X NOW POINTS TO DIRECTORY ENTRY
1688	CC19 86 42			#'A'+1	ASCII BIAS
1689	CC1B AB ØC			DIRASC,X	ADD TO ASCII FLAG
	CC1D 8D 1F CC1F E6 ØD			LCC3E DIRGRN,X	PRINT CHARACTER AND BLANK TO CONSOLE OUT GET FIRST GRANULE IN FILE
	CC21 8D 21			LCC44	COUNT GRANULES
1693	CC23 1F 89			A,B	SAVE COUNT IN ACCB
1694	CC25 4F		CLRA		CLEAR MS BYTE OF ACCD
	CC26 BD BD CC CC29 BD B9 58			LBDCC LB958	PRINT ACCD IN DECIMAL TO CONSOLE OUT SEND CARRIAGE RETURN TO CONSOLE OUT
1697	CC2C 35 10		PULS		PULL DIRECTORY POINTER OFF OF THE STACK
1698	CC2E 3Ø 88 2Ø	LCC2E		DIRLEN,X	MOVE X TO NEXT DIRECTORY ENTRY
1699	CC31 8C Ø7 ØØ			#DBUFØ+SECLEN	END OF I/O BUFFER?
1700 1701	CC34 25 B5 CC36 D6 ED			LCBEB DSEC	BRANCH IF MORE DIRECTORY ENTRIES IN BUFFER GET CURRENT SECTOR
1701	CC38 5C		INCB		BUMP COUNT
1703	CC39 C1 12			#SECMAX	SECMAX SECTORS IN DIRECTORY TRACK
1704	CC3B 23 A4	LCC3D		LCBE1	GET NEXT SECTOR
17Ø5 17Ø6	CC3D 39 CC3E BD A2 82	LCC3D	RTS	LA282	FINISHED SEND CHARACTER TO CONSOLE OUT
1707	CC41 7E B9 AC			LB9AC	SEND BLANK TO CONSOLE OUT
17Ø8					
1709				ACCB POINTING TO FIRST GRANULE IN	
1710 1711	CC44 BD C7 25	* GRANU LCC44		THE FILE IN ACCA, THE GRANULE DA	TA FOR THE LAST SECTOR IN ACCUMENT OF THE LAST SECTOR
	CC47 33 Ø6	20044		FATCON,X	POINT U TO START OF GRANULE DATA
1713	CC49 4F		CLRA		RESET GRANULE COUNTER
1714	CC4A 4C	LCC4A	INCA		INCREMENT GRANULE COUNTER
1715	CC4B 81 44 CC4D 10 22 F9 D5			#GRANMX LC626	CHECKED ALL 68 GRANULES? YES - 'BAD FILE STRUCTURE' ERROR
1716	CC51 30 C4		LEAX		POINT U TO START OF GRANULE DATA
	CC53 3A		ABX	,	ADD POINTER TO FIRST GRANULE
1719	CC54 E6 84		LDB		GET THIS GRANULE'S CONTROL BYTE
172Ø 1721	CC56 C1 CØ CC58 25 FØ			#\$CØ LCC4A	IS THIS THE LAST GRANULE IN FILE? NO - KEEP GOING
1721	CC5A 39		RTS	LUCTA	NO - KELI GUING
1723					
1724		* INPUT			
1725		DVFC10	ΓST	DEVNUM	* CHECK DEVICE NUMBER AND RETURN
	CC5B ØD 6F	5.2015		I CCDD	* TE NOT A DICK EILE
1726	CC5D 2F 5E	5,12015	BLE	LCCBD #LBØ69	* IF NOT A DISK FILE = CHANGE THE RETURN ADDRESS ON THE STACK TO RE-ENTER BASIC'S INPUT
		512015	BLE	#LBØ69	* IF NOT A DISK FILE = CHANGE THE RETURN ADDRESS ON THE STACK TO RE-ENTER BASIC'S INPUT = ROUTINE AT A DIFFERENT PLACE THAN THE CALLING ROUTINE

1729	CC64 8E Ø2 DD		LDX	#LINBUF+1	POINT X TO THE LINE INPUT BUFFER
1730	CC67 C6 2C		LDB	#','	=
1731	CC69 D7 Ø1		STB	CHARAC	=COMMA IS READ ITEM SEPARATOR (TEMPORARY STRING SEARCH FLAG)
1732	CC6B 96 Ø6			VALTYP	* GET VARIABLE TYPE AND BRANCH IF
1733	CC6D 26 Ø2			LCC71	* IT IS A STRING
1734	CC6F C6 20	1.0071		#SPACE	SPACE = NUMERIC SEARCH DELIMITER
1735 1736		LCC71		LCCE2 #SPACE	GET AN INPUT CHARACTER SPACE?
1737	CC73 81 20 CC75 27 FA			LCC71	YES - GET ANOTHER CHARACTER
1737	CC77 81 22		CMPA		QUOTE?
1739	CC79 26 ØA			LCC85	NO
1740	CC7B C1 2C		CMPB		SEARCH CHARACTER = COMMA?
1741	CC7D 26 Ø6			LCC85	NO - NUMERIC SEARCH
1742	CC7F 1F 89		TFR		* SAVE DOUBLE QUOTE AS
1743	CC81 D7 Ø1		STB	CHARAC	* THE SEARCH FLAG
1744	CC83 20 22		BRA	LCCA7	SAVE DOUBLE QUOTES AS FIRST ITEM IN BUFFER
1745					
		LCC85	CMPB		*
1747	CC87 27 11			LCC9A	*BRANCH IF INPUTTING A STRING VARIABLE
1748 1749	CC89 81 ØD CC8B 26 ØD		CMPA	LCC9A	IS THE INPUT CHARACTER A CARRIAGE RETURN NO
1750	CC8D 8C Ø2 DD			#LINBUF+1	*IF AT THE START OF INPUTBUFFER, CHECK FOR A
1751	CC9Ø 27 44			LCCD6	*FOLLOWING LINE FEED AND EXIT ROUTINE
1752	CC92 A6 1F		LDA		=IF THE INPUT CHARACTER PRECEEDING THE CR WAS A LINE FEED,
1753	CC94 81 ØA		CMPA		=THEN INSERT THE CR IN THE INPUT STRING, OTHERWISE
1754	CC96 26 3E		BNE	LCCD6	=CHECK FOR A FOLLOWING LINE FEED AND EXIT THE ROUTINE
1755	CC98 86 ØD		LDA	#CR	RESTORE CARRIAGE RETURN AS THE INPUT CHARACTER
1756	CC9A 4D	LCC9A	TSTA		*CHECK FOR A NULL (ZERO) INPUT CHARACTER AND
1757	CC9B 27 17			LCCB4	*IGNORE IT IF IT IS A NULL
1758	CC9D 91 Ø1			CHARAC	=
1759	CC9F 27 1D			LCCBE	=CHECK TO SEE IF THE INPUT CHARACTER MATCHES
1760	CCA1 34 Ø4		PSHS		=EITHER ACCB OR CHARAC AND IF IT DOES, THEN
1761 1762	CCA3 A1 EØ CCA5 27 17		CMPA	LCCBE	=BRANCH TO CHECK FOR ITEM SEPARATOR OR =TERMINATOR SEQUENCE AND EXIT ROUTINE
1762	CCA7 A7 8Ø	LCCA7	STA		STORE NEW CHARACTER IN BUFFER
1764	CCA9 8C Ø3 D6	LOOM		#LINBUF+LBUFMX	END OF INPUT BUFFER
1765	CCAC 26 Ø6			LCCB4	NO
1766	CCAE 8D 46			LCCF6	GET A CHARACTER FROM CONSOLE IN
1767	CCBØ 26 Ø6			LCCB8	EXIT ROUTINE IF BUFFER EMPTY
1768	CCB2 20 1E		BRA	LCCD2	CHECK FOR CR OR CR/LF AND EXIT ROUTINE
1769					
1770	CCB4 8D 4Ø	LCCB4		LCCF6	GET A CHARACTER FROM CONSOLE IN
1771	CCB6 27 CD			LCC85	BRANCH IF BUFFER NOT EMPTY
1772	CCB8 6F 84	LCCB8		, X	PUT A ZERO AT END OF BUFFER WHEN DONE
1773	CCBA 8E Ø2 DC CCBD 39	LCCBD	LDX RTS	#LINBUF	POINT (X) TO LINBUF - RESET POINTER
1774		20000	KIS		
1775				FM SEPARATOR OR TERMINATOR AND FX	IT THE INPUT ROUTINE
1775 1776	CCBE 81 22	* CHECK	FOR IT	EM SEPARATOR OR TERMINATOR AND EX	
1775 1776 1777	CCBE 81 22 CCCØ 27 Ø4		FOR IT	#'"'	QUOTE?
1775 1776	CCBE 81 22 CCCØ 27 Ø4 CCC2 81 20	* CHECK	FOR IT CMPA BEQ		
1775 1776 1777 1778	CCCØ 27 Ø4	* CHECK	FOR IT CMPA BEQ CMPA	#'"' LCCC6	QUOTE? YES
1775 1776 1777 1778 1779	CCCØ 27 Ø4 CCC2 81 2Ø	* CHECK	FOR IT CMPA BEQ CMPA BNE BSR	#"" LCCC6 #SPACE LCCB8 LCCF6	QUOTE? YES SPACE?
1775 1776 1777 1778 1779 1780 1781 1782	CCCØ 27 Ø4 CCC2 81 2Ø CCC4 26 F2 CCC6 8D 2E CCC8 26 EE	* CHECK LCCBE	FOR IT CMPA BEQ CMPA BNE BSR BNE	#**** LCCC6 #\$PACE LCCB8 LCCF6 LCCB8	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY
1775 1776 1777 1778 1779 1780 1781 1782 1783	CCCØ 27 Ø4 CCC2 81 2Ø CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCCA 81 2Ø	* CHECK LCCBE	FOR IT CMPA BEQ CMPA BNE BSR BNE CMPA	#*** LCCC6 #SPACE LCCBB LCCF6 LCCBB #SPACE	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE?
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784	CCCØ 27 Ø4 CCC2 81 2Ø CCC4 26 F2 CCC6 8D 2E CCC8 86 EE CCCA 81 2Ø CCCC 27 F8	* CHECK LCCBE	FOR IT CMPA BEQ CMPA BNE BSR BNE CMPA BEQ	#*** LCCC6 #SPACE LCCB8 LCCF6 LCCB8 #SPACE LCCC6	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCCA 81 20 CCCC 27 F8 CCCC 81 2C	* CHECK LCCBE	FOR IT CMPA BEQ CMPA BNE BSR BNE CMPA BEQ CMPA	#**** LCCC6 #SPACE LCCB8 LCCCB LCCB8 #SPACE LCCC6 #*, ***	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)?
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCCA 81 20 CCCC 27 F8 CCCE 81 2C CCCD 27 E6	* CHECK LCCBE LCCC6	FOR IT CMPA BEQ CMPA BNE BSR BNE CMPA BEQ CMPA BEQ	#"" LCCCB LCCBB LCCF6 LCCBB #SPACE LCCCB #"," LCCCB	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCCA 81 20 CCCC 27 F8 CCCC 81 2C	* CHECK LCCBE	FOR IT CMPA BEQ CMPA BNE BSR BNE CMPA BEQ CMPA BEQ CMPA	#"" LCCCB LCCBB LCCF6 LCCBB #SPACE LCCCB #"," LCCCB	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)?
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCCA 81 20 CCCC 27 F8 CCCC 81 2C CCC0 27 E6 CCC0 27 E6 CCC0 28  08	* CHECK LCCBE LCCC6	FOR IT CMPA BEQ CMPA BNE BSR BNE CMPA BEQ CMPA BEQ CMPA BNE	#*** LCCC6 #SPACE LCCB8 LCCCB LCCCB #SPACE LCCCCC #*,' LCCBB #CR LCCCDE	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCCA 81 20 CCCC 27 F8 CCCE 81 2C CCC00 27 E6 CCD0 27 E6 CCD2 81 0D	* CHECK LCCBE LCCC6	FOR IT CMPA BEQ CMPA BNE BSR BNE CMPA BEQ CMPA BEQ CMPA BNE BSR	#*** LCCC6 LCCBB LCCF6 LCCBB LCCC6 #SPACE LCCC6 #',' LCCBB #CR	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN?
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 81 20 CCCC 27 F8 CCCE 81 2C CCD0 27 E6 CCD2 81 0D CCD4 26 08 CCD6 8D 1E CCD8 26 DE CCD8 81 0A	* CHECK LCCBE LCCC6	FOR IT CMPA BEQ CMPA BNE BSR BMPA CMPA BEQ CMPA BNE BSR BNE CMPA BNE BSR BNE CMPA	#*** LCCC6 #SPACE LCCB8 LCCC6 LCCC6 #SPACE LCCCC6 LCCCC LCCCC #*,' LCCB8 LCCDE LCCCF6 LCCCF6 LCCCF6 LCCCB8 #LF	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCCA 81 20 CCCC 27 F8 CCCE 81 2C CCC00 27 E6 CCC0 21 0D CCCC 26 08 CCC0 26 08 CCCA 26 08 CCCA 27 04 CCCC 27 04	* CHECK LCCBE  LCCC6  LCCD2  LCCD6	FOR IT CMPA BEQ CMPA BNE CMPA BEQ CMPA BEQ CMPA BNE CMPA BNE CMPA BNE CMPA BNE CMPA BNE CMPA BNE CMPA BEQ	#*** LCCCB LCCCB LCCCB LCCCB LCCCB LCCCB #SPACE LCCCC #*,' LCCBB #CR LCCCF6 LCCCB LCCCF6 LCCDB LCCCB	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793	CCCØ 27 Ø4 CCC2 81 2Ø CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 81 2Ø CCCC 27 F8 CCCE 81 2C CCDØ 27 E6 CCDØ 27 E6 CCDØ 281 ØD CCC4 26 Ø8 CCC6 80 1E CCD6 8D 1E CCD8 26 DE CCD8 27 DA CCDC 27 DA	* CHECK LCCBE LCCC6	FOR IT CMPA BEQ CMPA BNE CMPA BEQ CMPA BNE CMPA BNE CMPA BNE CMPA BNE CMPA BNE BSR BNE CMPA BNE CMPA BSR	#"" LCCCB LCCBB LCCF6 LCCCBB #SPACE LCCCC6 #',' LCCCBB #CR LCCCB LCCCB LCCCBL L	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCCA 81 20 CCCC 27 F8 CCCE 81 2C CCC00 27 E6 CCC0 21 0D CCCC 26 08 CCC0 26 08 CCCA 26 08 CCCA 27 04 CCCC 27 04	* CHECK LCCBE  LCCC6  LCCD2  LCCD6	FOR IT CMPA BEQ CMPA BNE CMPA BEQ CMPA BNE CMPA BNE CMPA BNE CMPA BNE CMPA BNE BSR BNE CMPA BNE CMPA BSR	#*** LCCCB LCCCB LCCCB LCCCB LCCCB LCCCB #SPACE LCCCCC #*,' LCCCB #CR LCCCCB	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1789 1790 1791 1792 1793 1794 1795	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 27 F6 CCC0 81 0D CCC0 26 0B CCC0 27 E6 CCC0 81 0D CCC0 26 0B CCC0 81 0D CCC0 27 C6 CCC0 81 0D CCC0 81 0D CCC0 81 0D CCC0 80 1E CCC0 80 1E CCC0 80 1C CCC0 80 D6	* CHECK LCCBE  LCCC6  LCCC2  LCCD6	FOR IT CMPA BEQ CMPA BEQ CMPA BEQ CMPA BEQ CMPA BNE CMPA BNE CMPA BNE CMPA BNE CMPA BNE CMPA BREQ BSR BRA	#*** LCCCB LCCCB LCCCB LCCCB LCCCB LCCCCB #SPACE LCCCCB #*, ' LCCCB LCCCCB	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED; TREAT CR,LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795	CCCØ 27 Ø4 CCC2 81 2Ø CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 81 2Ø CCCC 27 F8 CCCE 81 2C CCDØ 27 E6 CCDØ 27 E6 CCDØ 281 ØD CCC4 26 Ø8 CCC6 80 DE	* CHECK LCCBE  LCCC6  LCCD2  LCCD6	FOR IT CMPA BEQ CMPA BNE BEQ CMPA BNE BSR BNE CMPA BNE BSR BNE CMPA BNE BSR BRA BSR	#"" LCCC6 #\$PACE LCCB8 LCCC6 LCCB8 #\$PACE LCCC6 #", LCCB8 #CR LCCDE LCCF6 LCCB8 #LF LCCB8 #LF LCCB8 LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BOCK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1795	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCCA 81 20 CCCC 27 F8 CCCE 81 2C CCC0 27 E6 CCC2 81 0D CCCC 26 08 CCD4 26 08 CCD4 26 08 CCD4 26 08 CCD4 27 DA CCD 27 DA CCCB 8D 1C CCCB 8D 1C CCCB 8D 1C CCCB 8D 1C CCCB 20 D6 CCCE 8D 1C CCCE 27 DA CCCE 8D 1C CCCE 20 D6 CCCE 27 DA	* CHECK LCCBE  LCCC6  LCCC2  LCCD6	FOR IT CMPA BEQ CMPA BEQ CMPA BEQ CMPA BEQ CMPA BEQ CMPA BNE BSR BNE CMPA BSR BRA BEQ BSR BRA BEQ	#*** LCCC6 #SPACE LCCBB LCCF6 LCCCBB #SPACE LCCCC6 #*,'' LCCBB #CR LCCDE LCCF6 LCCDB LCCF6 LCCBB LCCF6 LCCBB LCCCBB LCCCBB LCCCBB LCCCBB LCCCBB LCCCBB LCCCCBL LCCCBB LCCCCBL LCCCCCL LCCCCBB LCCCCCL LCCCCBB LCCCCCL LCCCCBB	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1789 1790 1791 1792 1793 1794 1795 1796 1797	CCCØ 27 Ø4 CCC2 81 2Ø CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 81 2Ø CCCC 27 F8 CCCE 81 2C CCDØ 27 E6 CCDØ 27 E6 CCDØ 281 ØD CCC4 26 Ø8 CCC6 80 DE	* CHECK LCCBE  LCCC6  LCCC2  LCCD6	FOR IT CMPA BEQ CMPA BNE BSR CMPA BEQ CMPA BEQ CMPA BNE CMPA BNE BSR BNE CMPA BNE BSR BRA BSR	#"" LCCC6 #\$PACE LCCB8 LCCC6 LCCB8 #\$PACE LCCC6 #", LCCB8 #CR LCCDE LCCF6 LCCB8 #LF LCCB8 #LF LCCB8 LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BOCK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1789 1790 1791 1792 1793 1794 1795 1796 1797 1798	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 81 0D CCC0 27 F6 CCC0 81 0D CCC0 26 0B CCC0 81 0D CCC0 81 0D CCC0 81 0D CCC0 81 0D CCC0 80 1E CCC0 80 1C CCC0 80 1C CCC2 8D 12 CCC2 7 15 CCC6 8D C7 14	* CHECK LCCBE  LCCC6  LCCC2  LCCD6	FOR IT	#*** LCCC6 #SPACE LCCB8 LCCF6 LCCCB8 #SPACE LCCC6 #*,',' LCCB8 #CR LCCDE LCCDE LCCDE LCCDE LCCDE LCCB8 LCCF6 LCCB8 LCCF6 LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR,LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 27 F6 CCC0 81 0D CCC0 27 E6 CCC0 81 0D CCC0 26 08 CCC0 81 0D CCC0 27 E6 CCC0 81 0D CCC0 81 0D CCC0 80 1E CCC0 80 1C CCC0 20 D6 CCC2 8D 12 CCC6 8D 12	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCDE	FOR ITI CMPA BEQ CMPA BNE BNE CMPA BEQ CMPA BEQ CMPA BEQ CMPA BEQ BSR BNE CMPA BSR BNE BBR BRA BEQ JSR BCA JSR LDB CMPB	#"" LCCC6 #\$PACE LCCBB LCCF6 LCCCB #"," LCCBB #CR LCCDE LCCF6 LCCBB #LF LCCBB #LF LCCBB LCCF6 LCCFB LCCFB LCCF6 LCCFB LCCFB LCCFB LCCFB LCCFB LCCFB LCCFB LCT14 FCBTYP, X	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITHM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR,LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802	CCCØ 27 Ø4 CCC2 81 2Ø CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 81 2Ø CCCC 27 F8 CCCE 81 2C CCCØ 27 F6 CCD2 81 ØD CCC4 26 Ø8 CCC6 80 D 1E CCD8 26 DE CCD8 27 DA CCD6 8D 1C CCC6 8D 1C CC6 8D 1C C	* CHECK LCCBE  LCCC6  LCCC2  LCCD6	FOR ITI CMPA BEQ CMPA BNE BNE BNE CMPA BEQ CMPA BEQ CMPA BEQ CMPA BEQ BNE	#"" LCCC6 #SPACE LCCBB LCCF6 LCCBB #SPACE LCCC6 #',' LCCBB #CR LCCDE LCCF6 LCCBB #LF LCCBB LCCFC LCCSB LCCCC LCCSB LCCCC LCCSB LCCCC LCCSB LCCCC LCCSB LCCCC LCCCSB LCCCC LCCCC LCCCSB LCCCC LCCCCC LCCCC LC	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTE INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1786 1786 1787 1798 1790 1791 1792 1793 1794 1795 1796 1797 1798 1797 1798 1799 1798 1799 1790 1801 1801 1801	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 27 F6 CCC0 81 0D CCC0 27 E6 CCC0 81 0D CCC0 26 08 CCC0 81 0D CCC0 27 E6 CCC0 81 0D CCC0 81 0D CCC0 80 1E CCC0 80 1C CCC0 20 D6 CCC2 8D 12 CCC6 8D 12	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCDE	FOR ITI CMPA BEQ CMPA BNE BNE BNE CMPA BEQ CMPA BEQ CMPA BEQ CMPA BEQ BNE	#"" LCCCB LCCCB LCCCB LCCCB LCCCB #SPACE LCCCB #"," LCCCB #"," LCCCB LCCCC LCCCB LCCCC LCCCB LCCCC LCCCB LCCCC LCCCB LCCCCL LCCCB LCCCCCL LCCCB LCCCCCL LCCCB LCCCCCL LCCCB LCCCCCCCC	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITHM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR,LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1798 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1802 1801 1802 1803	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 81 0D CCC0 27 F6 CCC0 81 0D CCC0 26 08 CCC0 81 0D CCC0 27 E6 CCC0 81 0D CCC0 81 0D CCC0 81 0D CCC0 80 1E CC0 80 1C CCC0 20 D6 CCC2 8D 12 CCC6 8D 12 CCC7 14 CCC9 10 26 F6 43 CCF1 C6 4A CCF1 C6 4A CCF1 C6 4A	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCDE	FOR IT CMPA BEQ CMPA BSR BNE CMPA BNE CMPA BNE CMPA BNE BNE CMPA BNE BNE CMPA BNE LDB BRA LDB LDB LDB LDB LDB LDB	#"" LCCC6 #SPACE LCCB8 LCCF6 LCCCB8 #SPACE LCCC6 #',' LCCB8 #CR LCCDE LCCF6 LCCB8 #LF LCCB8 LCCF6 LCCB8 LCCF6 LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCAB4 LCCFC LCCAB4 LCCFC LCCAB4 LCCFC LCCAB4 LCCFC LCCAB4 LCCFC LCCAB4 LCCAAA4 LCCAAA44 LCCAAA444 LCCAAA44 LCCAAA444 LCCAAA444 LCCAAA444 LCCAAA444 LCCAAA444 LCCAA	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER
1775 1776 1777 1778 1779 1780 1780 1781 1782 1783 1784 1785 1786 1787 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1802 1803 1804	CCCØ 27 Ø4 CCC2 81 2Ø CCC4 26 F2 CCC6 8D 2E CCC8 8D 2E CCC8 81 2Ø CCCC 27 F8 CCCE 81 2C CCC0 81 0D CCC0 26 F6 CCD2 81 ØD CCC0 26 F6 CCD2 81 ØD CCC0 27 F6 CCD2 81 ØD CCC0 26 F0 CCC0 80 1E CCC0 80 1E CCC0 80 1C CCC0 90 F0 CCC0 16 CCC0 16 CCC0 16 CCC0 16 CCC0 17 CC	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCDE	FOR ITI CMPA BEQ CMPA BNE BNE BNE BNE CMPA BNE BNE BNE CMPA BNE	#"" LCCC6 #\$PACE LCCBB LCCF6 LCCCB #"," LCCBB #CR LCCDE LCCF6 LCCBB #LF LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCFB LCCF6 LCCFB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCAB LCCF6 LCCAB LCCF6 LCCAB LCCF6 LCCAB LCABA LCAAAAAAAAAAAAAAAAAAAAAAAAA	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTE INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER  GET A CHAR FROM INPUT BUFFER  GET ACHAR FROM SPACEORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1786 1786 1787 1798 1790 1791 1792 1793 1794 1795 1796 1797 1798 1797 1798 1799 1799 1798 1799 1798 1799 1801 1801 1801 1803 1804 1806	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 21 20 CCCC 27 F8 CCC0 27 F6 CCC0 21 CC CCC0 27 DA CCC0 26 D 1C CCC0 27 DA CCC0 28 D 1C CCC0 29 D6 CCC2 8D 12 CCC6 27 T5 CCC6 BD C7 14 CCC9 CCC 26 CC CCC6 CCC 27 T5 CCC6 CCC6 CC 27 T5 CCC6 CC 27 T5 CCC6 CCC6 CC 27 T5 CCC6 CCC6 CC 27 T5 CCC	* CHECK LCCBE  LCCC6  LCCD2 LCCD6  LCCDE LCCE2  LCCF1  LCCF6	FOR IT CMPA BEQ CMPA BSR BNE CMPA BEQ CMPA BEQ BSR BNE BSR BNE BSR BNE BSR BNE BSR BNE LMPA BEQ BSR BSR BRA BEQ JSR LOMPA LDB LDB JJR JSR TST	#"" LCCC6 #SPACE LCCB8 LCCF6 LCCCB8 #SPACE LCCC6 #',' LCCB8 #CR LCCDE LCCF6 LCCB8 #LF LCCB8 LCCF6 LCCB8 LCCF6 LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCAB4 LCCFC LCCAB4 LCCFC LCCAB4 LCCFC LCCAB4 LCCFC LCCAB4 LCCFC LCCAB4 LCCAAA4 LCCAAA44 LCCAAA444 LCCAAA44 LCCAAA444 LCCAAA444 LCCAAA444 LCCAAA444 LCCAAA444 LCCAA	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1798 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1801 1802 1803 1804 1805 1806	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 21 20 CCCC 27 F8 CCC0 27 F6 CCC0 21 CC CCC0 27 DA CCC0 26 D 1C CCC0 27 DA CCC0 28 D 1C CCC0 29 D6 CCC2 8D 12 CCC6 27 T5 CCC6 BD C7 14 CCC9 CCC 26 CC CCC6 CCC 27 T5 CCC6 CCC6 CC 27 T5 CCC6 CC 27 T5 CCC6 CCC6 CC 27 T5 CCC6 CCC6 CC 27 T5 CCC	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCDE	FOR ITI CMPA BEQ CMPA BNE BNE BNE BNE CMPA BNE BNE BNE CMPA BNE	#"" LCCC6 #\$PACE LCCBB LCCF6 LCCCB #"," LCCBB #CR LCCDE LCCF6 LCCBB #LF LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCFB LCCF6 LCCFB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCAB LCCF6 LCCAB LCCF6 LCCAB LCCF6 LCCAB LCABA LCAAAAAAAAAAAAAAAAAAAAAAAAA	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTE INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER  GET A CHAR FROM INPUT BUFFER  GET ACHAR FROM SPACEORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1798 1791 1792 1793 1794 1795 1796 1797 1798 1797 1798 1800 1801 1802 1803 1804 1805 1806 1807	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 21 20 CCCC 27 F8 CCC0 27 F6 CCC0 21 CC CCC0 27 DA CCC0 26 D 1C CCC0 27 DA CCC0 28 D 1C CCC0 29 D6 CCC2 8D 12 CCC6 27 T5 CCC6 BD C7 14 CCC9 CCC 26 CC CCC6 CCC 27 T5 CCC6 CCC6 CC 27 T5 CCC6 CC 27 T5 CCC6 CCC6 CC 27 T5 CCC6 CCC6 CC 27 T5 CCC	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB	FOR IT CMPA BEQ CMPA BSR BSR BNE CMPA BNE BSR	#"" LCCC6 #\$PACE LCCBB LCCF6 LCCCB #"," LCCBB #CR LCCDE LCCBB LCCF6 LCCBB #LF LCCBB LCCF6 LCCBB LCCF6 LCCFB LCCF6 LCCFB LCCF6 LCCFB LCTA LCCFB LCTA LCTB LCTB LCTA LCTB LCTB LCTA LCTB LCTB LCTA LCTB LCTB LCTB LCTB LCTB LCTB LCTB LCTB	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTE INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER  GET A CHAR FROM INPUT BUFFER  GET ACHAR FROM SPACEORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1786 1786 1787 1788 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1801 1801 1802 1803 1804 1806 1807 1806	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC8 26 EE CCC8 21 20 CCC2 7 F8 CCC8 1 20 CCC0 27 F6 CCC9 21 00 CCC0 27 F6 CC00 27 E6 CC00 281 00 CCC04 26 08 CC06 80 1E CC08 26 DE CC08 81 0A CC0C 27 DA CC0E 80 1C CCC0 27 DA CCCE 80 1C CCC6 20 D6 CCC2 8D 12 CCC6 27 15 CCE6 E6 00 CCC2 8D 12 CCC6 27 14 CCC9 20 D6 CCC6 26 C7 14 CCC9 CCC6 C6 C7 14 CCC9 CCC6 C7 14 CCC9 CCC7 7 14 CCC9 CC7 7 15 CCC6 CC7 7 14 CCC7 7 15 CCC6 7 16 CCC7 7 15	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCF6  LCCFB  * MOVE T	FOR IT CMPA BEQ CMPA BNE BNE BNE CMPA BNE	#*** LCCC6 #SPACE LCCBB LCCF6 LCCCBB #SPACE LCCCC6 #*,' LCCBB #CR LCCDE LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCFC LCCBB LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46 LA176 CINBFL	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF FRECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1798 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1801 1802 1803 1804 1805 1806 1807 1808	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 21 20 CCCC 27 F8 CCC0 27 F6 CCC0 21 CC CCC0 27 DA CCC0 26 D 1C CCC0 27 DA CCC0 28 D 1C CCC0 29 D6 CCC2 8D 12 CCC6 27 T5 CCC6 BD C7 14 CCC9 CCC 26 CC CCC6 CCC 27 T5 CCC6 CCC6 CC 27 T5 CCC6 CC 27 T5 CCC6 CCC6 CC 27 T5 CCC6 CCC6 CC 27 T5 CCC	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB	FOR IT CMPA BEQ CMPA BEQ CMPA BEQ CMPA BNE BNE CMPA BNE BNE CMPA BNE BNE CMPA BSR BRA BSR BC BSR BC BSR BC BSR BC BSR BC BSR BSR BC BSR BSR BC BSR BSR BC BSR BSR BC BSR B	#*** LCCC6 #SPACE LCCBB LCCF6 LCCCBB #SPACE LCCCC6 #*,' LCCBB #CR LCCDE LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCFC LCCBB LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46 LA176 CINBFL	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTE INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER  GET A CHAR FROM INPUT BUFFER  GET ACHAR FROM SPACEORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1798 1799 1791 1792 1793 1794 1795 1796 1797 1798 1799 1799 1800 1801 1804 1806 1806 1806 1807 1808 1808	CCC2 27 04 CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 21 00 CCC0 27 F6 CCC0 21 00 CCC0 27 E6 CC00 27 E6 CC00 27 E6 CC00 27 E6 CC00 20 00 CC00 27 E6 CC00 20 00 CC00 20 20 00 CC00 20 20 20 CC00 20 CC00 20 20 CC00	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCF6  LCCFB  * MOVE T	FOR IT CMPA BEQ CMPA BSR BNE CMPA BEQ CMPA BEQ CMPA BSR BNE BSR BRA BEQ JSR LBNE LBNE LBNE LBNE LBNE LBNE LBNE LBNE	#"" LCCC6 #SPACE LCCB8 LCCF6 LCCB8 #SPACE LCCC6 #"," LCCB8 #CR LCCDE LCCF6 LCCB8 LCCF6 LCCB8 LCFC LCCB8 LCFF LCCB8 LCCFC LCCB8 LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46 LA176 CINBFL PUT POINTER BACK ONE (DISK FILE) X,B	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG  SAVE REGISTERS ON STACK
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1790 1791 1792 1793 1794 1795 1796 1797 1798 1796 1801 1801 1801 1803 1804 1806 1807 1807 1809 1809 1809 1809 1809 1810 1811 1809 1810 1811 1812	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC8 26 EE CCC8 21 20 CCC2 7 F8 CCC6 81 2C CCC9 27 E6 CCC9 21 00 CCC0 27 E6 CC02 81 00 CCC0 27 E6 CC02 81 00 CCC04 26 08 CCD4 26 08 CCD4 26 08 CCD4 27 DA CCD6 80 1E CC08 27 DA CCC0 80 1C CCC0 27 DA CCC0 28 D 12 CCC0 27 DA CCC0 27 DA CCC0 28 D 12 CCC0 27 DA CCC0 28 D 12 CCC0 27 DA CCC0 28 D 12 CCC0 28 D 1	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCF6  LCCFB  * MOVE T	FOR IT CMPA BEQ CMPA BNE BNE BNE CMPA BNE	#"" LCCC6 #SPACE LCCBB LCCF6 LCCCB #SPACE LCCCC6 #",' LCCBB #CR LCCDE LCCF6 LCCBB #LF LCCBB LCCF6 LCCF6 LCCFB LCCF6 LCCFB LCTA LCCBB LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46 LA176 CINBFL  PUT POINTER BACK ONE (DISK FILE) X,B LC714	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTE INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE PROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG  SAVE REGISTERS ON STACK POINT X TO PROPER FCB
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1786 1787 1788 1790 1791 1792 1793 1794 1795 1796 1797 1798 1798 1799 1801 1801 1802 1803 1804 1805 1806 1807 1809 1809 1809 1809 1809 1809 1809 1809	CCC2 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 27 F6 CCC0 81 0D CCC0 27 F6 CCC0 81 0D CC0 26 08 CCC0 81 0D CC0 26 08 CCC0 81 0D CC0 27 0A CC0 80 12 CC0 80 12 CCC0 80 12 CCC0 27 DA CC0 80 10 CCC0 27 DA CC0 80 10 CCC6 8D 11 CCC6 8D 12 CCC6 8D 12 CCC6 8D 12 CCC6 8D 140 CCC6 8D 140 CCC6 8D 16 CCC7 37 E AC 46 CC7 57 E AC 46	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB  * MOVE T LCCFC	FOR IT CMPA BEQ CMPA BER BNE BNE CMPA BEQ CMPA BEQ BNE	#"" LCCC6 #SPACE LCCBB LCCF6 LCCCB #SPACE LCCCC6 #",' LCCBB #CR LCCDE LCCF6 LCCBB #LF LCCBB LCCF6 LCCF6 LCCF6 LCCF6 LCCFB LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46  LA176 CINBFL  PUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC334 #2*47 LC46  LC174 FCBTYP,X #RANFIL LC714 FCBTYP,X #RANFIL LC714 FCBTYP,X #RANFIL LC714 FCBTYP,X #RANFIL LC714 FCBTYP,X	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR.LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF FRECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1790 1791 1792 1793 1794 1795 1796 1797 1798 1796 1797 1801 1801 1801 1803 1804 1807 1806 1807 1809 1810 1811 1812 1813 1814	CCC2 27 04 CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 27 F6 CCC0 21 00 CCC0 27 F6 CCC0 28 1 00 CCC0 27 F6 CCC0 28 1 00 CCC0 27 F6 CCC0 27 F6 CCC0 27 F6 CCC0 28 10 CCC0 27 F6 CCC0 27 F6 CCC0 27 F6 CCC0 28 10 CCC0 27 F6 CCC0 27 F6 CCC0 27 F6 CCC0 28 10 CCC0 27 F6 CCC0 27 F6 CCC0 27 F7 CCC0 28 F7 CCC0 27	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB  * MOVE T LCCFC	FOR IT CMPA BEQ CMPA BER BNE BNE CMPA BEQ CMPA BEQ BNE	#"" LCCC6 #SPACE LCCB8 LCCF6 LCCB8 #SPACE LCCC6 #"," LCCBB #CR LCCDE LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCFC LCCBB LCCFC LCCBB LCCFC LCCBB LCCFC LCCBB LCT14 FCBTYP,X #RANFIL LC344 #2*37 LAC46 LA176 CINBFL  DUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF FRECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE  BRANCH IF NOT A RANDOM FILE GERAB THE RANDOM FILE 'GET' POINTER,
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1798 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1800 1801 1801 1802 1803 1804 1805 1806 1807 1808 1807 1808 1809 1811 1812 1813 1814 1815 1816	CCC2 27 04 CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 27 F6 CCC0 21 00 CCC0 27 F6 CCC0 21 00 CCC0 27 F6 CCC0 21 00 CCC0 27 E6 CCC0 21 00 CCC0 27 E6 CCC0 21 00 CCC0 26 D0 CCC0 27 E6 CCC0 20 D1 CCC0 20 D6 CCC2 80 12 CCC2 20 D6 CCC2 80 12 CCC4 27 15 CCC6 80 C7 14 CCC9 10 26 F6 43 CCF1 C6 4A CCF3 7E AC 46 CCF6 8D A1 76 CCF9 00 70 CCF6 8D A1 76 CCF6 8D A1 76 CCF6 8D A1 76 CCF6 8D A1 76 CCF6 8D C7 14 CCCF6 8D C7 14 CCCF	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB  * MOVE T LCCFC	FOR IT CMPA BEQ CMPA BSR BNE CMPA BNE BSR BNE CMPA BNE BSR	#"" LCCC6 #SPACE LCCB8 LCCF6 LCCB8 #SPACE LCCC6 #"," LCCB8 #CR LCCDE LCCF6 LCCB8 LCCF6 LCCB8 LCCF6 LCCB8 LCCF6 LCCFB LCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCCFC LCCB8 LCT14 #2*37 LAC46 LA176 CINBFL  DUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC012 FCBGTYP,X #RANFIL LC012 FCBGTYP,X #RANFIL LC012 FCBGTYP,X #RANFIL LC012	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE  *GRAB THE RANDOM FILE 'GET' POINTER, *MOVE IT BACK ONE AND RESTORE IT
1775 1776 1777 1778 1778 1780 1781 1782 1783 1784 1785 1786 1787 1798 1799 1791 1792 1793 1794 1795 1796 1800 1801 1802 1803 1804 1806 1807 1806 1807 1808 1809 1811 1812 1814 1815 1816	CCC2 87 04 CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC8 26 EE CCC8 27 F8 CCC2 81 20 CCC2 81 20 CCC2 81 20 CCC2 81 00 CCC4 26 08 CCD4 26 08 CCD4 26 08 CCD4 26 08 CCD6 80 1E CCD8 27 DA CCDE 80 1C CCCB 20 D6 CCCB 27 DA CCDE 80 1C CCCB 20 D6 CCCB 21 00 CCCB 20 D6 CCCB 21 00 CCCB 27 DA CCD 27 DA CCD 27 DA CCD 27 DA CCD 27 DA CCCB 20 D6 CCCB 21 00 CCCB 20 D6 CCCB 21 40 CCCB	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB  * MOVE T LCCFC	FOR IT CMPA BEQ CMPA BSR BSR BCMPA BEQ CMPA BEQ CMPA BEQ BSR BNE CMPA BEQ BSR BNE BNE BSR BRA BEQ BSR BSR BRA BEQ JSR LDB LBNE LBNE LBNE JMP STST RTS LDB CMPB LBNE LDB LBNE LDB LBNE LDB	#"" LCCC6 #SPACE LCCBB LCCF6 LCCCB #SPACE LCCCC6 #',' LCCBB #CR LCCDE LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCF6 LCCFB LCCF6 LCCFB LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46  LA176 CINBFL  PUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC374 FCBTYP,X #RANFIL LC1714 FCBTYP,X #RANFIL LC714 FCBTYP,X #RANFIL LC714 FCBTYP,X #RANFIL LC715 FCBGET,X #\$0001 FCBGET,X	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? I'NPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE  *GRAB THE RANDOM FILE 'GET' POINTER, *MOVE IT BACK ONE AND RESTORE IT *
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1798 1790 1791 1792 1793 1794 1795 1796 1797 1798 1799 1802 1803 1804 1805 1806 1807 1808 1808 1809 1811 1812 1813 1814 1815 1816 1817 1818	CCC2 27 04 CCC2 81 20 CCC4 26 F2 CCC6 8D 2E CCC8 26 EE CCC8 27 F8 CCCE 81 20 CCC0 27 F6 CCC0 81 0D CCC0 27 F6 CCC0 81 0D CCC0 27 E6 CCC0 81 0D CCC0 80 1E CCC0 80 1C CCC0 27 DA CCC0 80 1C CCC0 20 D6 CCC2 8D 12 CCC4 27 15 CCC6 8D 1C CCC7 7 CA CCC6 8D 1C CCC7 37 E AC CCC6 8D A1 76 CCC7 37 E AC CCC6 8D A1 76 CCC7 30 T 4 CCC6 8D A1 76 CCC6 8D A1 76 CCC6 8D A1 76 CCC7 34 14 CCC6 8D A1 76 CCC6 8D	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB  * MOVE T	FOR IT CMPA BEQ CMPA BSNE BSNE BSNE CMPA BNE BSNE BNE CMPA BSNE BNE CMPA BSNE BNE LBD BSNE BNE LBD JSNE LDB	#*** LCCC6 #SPACE LCCB8 LCCF6 LCCCB8 LCCC6 #**,' LCCB8 #SPACE LCCCC6 #**,' LCCB8 #CCR LCCCB LCCCF6 LCCCB8 LCCF6 LCCCB8 LCCF6 LCCB8 LCCF6 LCCB8 LCCF6 LCCB8 LCCFC LCCFB LCCFC LCCFB LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46 LA176 CINBFL  PUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC334 #2*37 LAC46 LA176 CINBFL  CUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC112 FCBTYP,X #RANFIL LC112 FCBTYP,X #RANFIL LC112 FCBTYP,X #RANFIL LC112 FCBGTYP,X #RANFIL	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF FRECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE *GRAB THE RANDOM FILE 'GET' POINTER, *MOVE IT BACK ONE AND RETURN
1775 1776 1777 1778 1778 1780 1781 1782 1783 1784 1785 1786 1787 1792 1793 1794 1792 1793 1794 1795 1796 1797 1798 1803 1804 1805 1806 1807 1808 1808 1808 1808 1809 1811 1811 1811	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC6 81 20 CCCC 27 F8 CCC6 81 2C CCD0 27 E6 CCD2 81 0D CCC2 81 0D CCC4 26 08 CCD4 26 08 CCD6 81 10 CCCD 81 0D CCCD 81 0D CCCD 81 0D CCCD 80 1E CCD0 27 DA CCD6 8D 1C CCCD 80 1C CCCE 80 1C	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB  * MOVE T LCCFC	FOR IT CMPA BEQ CMPA BSR BSR BNE CMPA BEQ CMPA BEQ CMPA BEQ CMPA BEQ JSR BNE	#"" LCCC6 #SPACE LCCBB LCCF6 LCCCB #SPACE LCCCC6 #',' LCCBB #CR LCCDE LCCF6 LCCBB #LF LCCBB LCCF6 LCCBB LCCFC LCCB	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTE INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE  *GRAB THE RANDOM FILE? BRANCH IF NOT A RANDOM FILE  *GRAB THE RANDOM FILE 'GET' POINTER, **MOVE IT BACK ONE AND RESTORE IT  * *ESTORE REGISTERS AND RETURN SAVE THE CHARACTER IN THE CACHE
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1799 1791 1792 1793 1794 1795 1796 1801 1802 1803 1804 1806 1807 1806 1807 1808 1809 1811 1812 1813 1814 1815 1816 1817 1818	CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC8 26 EE CCC8 21 20 CCC2 27 F8 CCC6 81 2C CCC9 21 E2 CCC9 21 E2 CCC9 21 E2 CCC9 26 E8 CCC9 27 E6 CC19 27 E6 CC19 27 E6 CC19 26 B1 CCC10 27 E6 CC10 27 E7 CC10 20 E7	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB  * MOVE T	FOR IT CMPA BEQ CMPA BSR BNE CMPA BSR BR	#"" LCCC6 #SPACE LCCBB LCCF6 LCCCBB #SPACE LCCCBB #SPACE LCCCCB #",' LCCBB #CR LCCDE LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCF6 LCCF6 LCCFB LCCF6 LCCFB LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46  LA176 CINBFL  DUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC334 #2*437 LAC46  LA176 CINBFL  DUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC012 FCBGET,X #\$0001 FCBGET,X #\$0001 FCBGET,X B,X,PC FCBCCT,X FCBCCT,X FCBCCT,X FCBCCT,X	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? I'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF FRECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE  *GRAB THE RANDOM FILE 'GET' POINTER, **MOVE IT BACK ONE AND RETURN SAVE THE CHARACTER IN THE CACHE SET THE CACHE FLAG TO SFF - DATA IN CACHE
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1798 1790 1791 1792 1793 1794 1795 1796 1797 1798 1796 1797 1802 1803 1804 1807 1806 1807 1806 1807 1806 1807 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1818 1819 1818 1819 1818 1819 1818 1819 1818 1819 1818 1819 1818 1819 1811 1812 1813 1814 1815 1816 1817 1818 1819 1821 1821 1821 1821 1821 1821	CCC0 27 04 CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC6 81 20 CCCC 27 F8 CCC6 81 2C CCD0 27 E6 CCD2 81 0D CCC2 81 0D CCC4 26 08 CCD4 26 08 CCD6 81 10 CCCD 81 0D CCCD 81 0D CCCD 81 0D CCCD 80 1E CCD0 27 DA CCD6 8D 1C CCCD 80 1C CCCE 80 1C	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB  * MOVE T	FOR IT CMPA BEQ CMPA BSR BNE CMPA BSR BR	#"" LCCC6 #SPACE LCCBB LCCF6 LCCCB #SPACE LCCCC6 #',' LCCBB #CR LCCDE LCCF6 LCCBB #LF LCCBB LCCF6 LCCBB LCCFC LCCB	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTE INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? 'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF RECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE  *GRAB THE RANDOM FILE? BRANCH IF NOT A RANDOM FILE  *GRAB THE RANDOM FILE 'GET' POINTER, **MOVE IT BACK ONE AND RESTORE IT  * *ESTORE REGISTERS AND RETURN SAVE THE CHARACTER IN THE CACHE
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1798 1791 1792 1793 1794 1795 1796 1797 1798 1797 1798 1800 1801 1802 1803 1804 1805 1806 1807 1808 1809 1811 1812 1811 1812 1813 1814 1815 1816 1817 1818 1819 1819 1811 1812 1813 1814 1815 1816 1817 1818 1819 1819 1819 1811 1812 1813 1814 1815 1816 1817 1818 1819 1819 1819 1819 1819 1819	CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC8 26 EE CCC8 21 20 CCC2 27 F8 CCC6 81 2C CCC9 21 E2 CCC9 21 E2 CCC9 21 E2 CCC9 26 E8 CCC9 27 E6 CC19 27 E6 CC19 27 E6 CC19 26 B1 CCC10 27 E6 CC10 27 E7 CC10 20 E7	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCF1  LCCF6  LCCF6  LCCFC  T  LCCFC  LCCF1	FOR IT CMPA BEQ CMPA BSR BSR BNE CMPA BEQ CMPA BEQ CMPA BEQ JER BNE	#"" LCCC6 #SPACE LCCBB LCCF6 LCCCBB #SPACE LCCCBB #SPACE LCCCCB #",' LCCBB #CR LCCDE LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCF6 LCCF6 LCCFB LCCF6 LCCFB LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46  LA176 CINBFL  DUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC334 #2*437 LAC46  LA176 CINBFL  DUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC012 FCBGET,X #\$0001 FCBGET,X #\$0001 FCBGET,X B,X,PC FCBCCT,X FCBCCT,X FCBCCT,X FCBCCT,X	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? I'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF FRECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE  *GRAB THE RANDOM FILE 'GET' POINTER, **MOVE IT BACK ONE AND RETURN SAVE THE CHARACTER IN THE CACHE SET THE CACHE FLAG TO SFF - DATA IN CACHE
1775 1776 1777 1778 1779 1780 1781 1782 1783 1784 1785 1786 1787 1788 1799 1791 1792 1793 1794 1795 1796 1801 1802 1803 1804 1806 1807 1806 1807 1808 1809 1810 1811 1812 1813 1814 1815 1816 1817 1818 1819 1821 1821 1821	CCC2 81 20 CCC4 26 F2 CCC6 80 2E CCC8 26 EE CCC8 21 20 CCC2 27 F8 CCC6 81 2C CCC9 21 E2 CCC9 21 E2 CCC9 21 E2 CCC9 26 E8 CCC9 27 E6 CC19 27 E6 CC19 27 E6 CC19 26 B1 CCC10 27 E6 CC10 27 E7 CC10 20 E7	* CHECK LCCBE  LCCC6  LCCD2  LCCD6  LCCDE  LCCCE2  LCCF1  LCCF6  LCCFB  * MOVE T	FOR IT CMPA BEQ CMPA BSR BNE CMPA BSR BNE CMPA BSR BNE CMPA BSR BNE BSR	#"" LCCC6 #SPACE LCCBB LCCF6 LCCCBB #SPACE LCCCBB #SPACE LCCCCB #",' LCCBB #CR LCCDE LCCF6 LCCBB LCCF6 LCCBB LCCF6 LCCF6 LCCF6 LCCFB LCCF6 LCCFB LCT14 FCBTYP,X #RANFIL LC334 #2*37 LAC46  LA176 CINBFL  DUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC334 #2*437 LAC46  LA176 CINBFL  DUT POINTER BACK ONE (DISK FILE) X,B LC714 FCBTYP,X #RANFIL LC012 FCBGET,X #\$0001 FCBGET,X #\$0001 FCBGET,X B,X,PC FCBCCT,X FCBCCT,X FCBCCT,X FCBCCT,X	QUOTE? YES SPACE? NO - EXIT ROUTINE GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY SPACE? YES - GET ANOTHER CHARACTER COMMA (ITEM SEPARATOR)? YES - EXIT ROUTINE CARRIAGE RETURN? NO GET A CHARACTER FROM CONSOLE IN EXIT ROUTINE IF BUFFER EMPTY LINE FEED? TREAT CR, LF AS A CR YES - EXIT ROUTINE BACK UP PTR INPUT POINTER ONE EXIT ROUTINE  GET A CHAR FROM INPUT BUFFER - RETURN IN ACCA RETURN IF BUFFER NOT EMPTY POINT X TO START OF FILE BUFFER GET FILE TYPE IS IT RANDOM FILE TYPE? I'INPUT PAST END OF FILE ERROR IF NOT RANDOM 'WRITE/INPUT PAST END OF FRECORD ERROR IF RANDOM JUMP TO THE ERROR HANDLER  GET A CHAR FROM INPUT BUFFER SET FLAGS ACCORDING TO CONSOLE INPUT FLAG  SAVE REGISTERS ON STACK POINT X TO PROPER FCB GET FILE TYPE OF THIS FCB IS IT A RANDOM FILE? BRANCH IF NOT A RANDOM FILE  *GRAB THE RANDOM FILE 'GET' POINTER, **MOVE IT BACK ONE AND RETURN SAVE THE CHARACTER IN THE CACHE SET THE CACHE FLAG TO SFF - DATA IN CACHE

1825	CD1D C1 Ø5	CMPB #\$Ø5	FIVE BYTES IN A FLOATING POINT NUMBER
1826	CD1F 10 25 E7 27	LBCS LB44A	'FC' ERROR IF ⇔ 5 BYTES
1827	CD23 ØF Ø6	CLR VALTYP	SET VARIABLE TYPE TO NUMERIC
1828	CD25 7E BC 14	JMP LBC14	COPY A PACKED FP NUMBER FROM (X) TO FPAØ
1829		+ MINIT COMMAND	
183Ø 1831	CD28 BD B1 43	* MKN\$ COMMAND MKN JSR LB143	'TM' ERROR IF VALTYP=STRING
1832	CD2B C6 Ø5	LDB #\$Ø5	FIVE BYTES IN A FLOATING POINT NUMBER
1833	CD2D BD B5 ØF	JSR LB5ØF	RESERVE FIVE BYTES IN STRING SPACE
1834	CD3Ø BD BC 35	JSR LBC35	PACK FPAØ AND STORE IT IN STRING SPACE
1835	CD33 7E B6 9B	JMP LB69B	SAVE STRING DESCRIPTOR ON STRING STACK
1836	0000 72 00 90	0111 25035	SAVE STRING BESONT FOR ON STRING STACK
1837		* LOC COMMAND	
1838	CD36 8D Ø5	LOC BSR LCD3D	POINT X TO FILE BUFFER
1839	CD38 EC Ø7	LDD FCBREC,X	GET RECORD NUMBER (RANDOM FILE) OR SECTOR CTR (SEQUENTIAL)
1840	CD3A 7E B4 F4	LCD3A JMP GIVABF	PUT ACCD IN FPAØ
1841			
1842			M A BASIC STATEMENT, SET PRINT
1843		* PARAMETERS ACCORDING TO I	
1844	CD2D 0C CE	* OPEN. RETURN WITH (X) POI	
1845 1846	CD3D 96 6F CD3F 34 Ø2	LCD3D LDA DEVNUM PSHS A	* GET CURRENT DEVICE NUMBER AND * SAVE IT ON THE STACK
1847	CD41 BD B1 43	JSR LB143	'TM' ERROR IF VALTYP=STRING
1848	CD44 BD A5 AE	JSR LA5AE	CHECK FOR VALID DEVICE NUMBER/SET PRINT PARAMETERS
1849	CD47 ØD 6F	TST DEVNUM	* CHECK DEVICE NUMBER
1850	CD49 10 2F E6 FD	LBLE LB44A	* BRANCH IF NOT DISK FILE 'ILLEGAL FUNCTION CALL'
1851	CD4D BD C7 14	JSR LC714	POINT (X) TO FILE BUFFER
1852	CD5Ø 35 Ø2	PULS A	* GET OLD DEVICE NUMBER OFF OF THE STACK AND
1853	CD52 97 6F	STA DEVNUM	* SAVE IT AS DEVICE NUMBER
1854	CD54 6D ØØ	TST FCBTYP,X	IS FILE OPEN?
1855	CD56 10 27 D6 A1	LBEQ LA3FB	'FILE NOT OPEN' ERROR IF NOT OPEN
1856	CD5A 39	RTS	
1857			
1858	0050 00 50	* LOF COMMAND	DOLLEY V. TO STUE DUSSED
1859	CD5B 8D EØ	LOF BSR LCD3D	POINT X TO FILE BUFFER
1860	CD5D A6 Ø1 CD5F 97 EB	LDA FCBDRV,X	* GET DRIVE NUMBER AND SAVE IT
1861 1862	CD61 E6 Ø2	STA DCDRV LDB FCBFGR,X	* IN DSKCON VARIABLE GET FIRST GRANULE OF FILE
1863	CD63 34 1Ø	PSHS X	SAVE FCB POINTER ON STACK
1864	CD65 BD CC 44	JSR LCC44	FIND TOTAL NUMBER OF GRANULES IN THIS FILE
1865	CD68 4A	DECA	SUBTRACT THE LAST GRANULE IN THE FILE
1866	CD69 C4 3F	ANDB #\$3F	GET NUMBER OF SECTORS USED IN LAST GRANULE
1867	CD6B 34 Ø4	PSHS B	SAVE NUMBER OF SECTORS IN LAST GRANULE ON STACK
1868	CD6D 1F 89	TFR A,B	* CONVERT ACCA TO POSITIVE
1869	CD6F 4F	CLRA	* 2 BYTE VALUE IN ACCD
1870	CD7Ø BD C7 49	JSR LC749	MULT NUMBER OF FULL GRANULES BY 9
1871	CD73 EB EØ	ADDB ,S+	ADD NUMBER SECTORS IN LAST TRACK
1872	CD75 89 ØØ	ADCA HERR	PROPAGATE CARRY TO MS BYTE OF ACCD
		ADCA #\$ØØ	TROTAGNIE GARRY TO NO BITE OF ACCE
1873	CD77 35 10	PULS X	GET FCB POINTER BACK
1874	CD77 35 10 CD79 34 02	PULS X PSHS A	GET FCB POINTER BACK SAVE ACCA ON STACK
1874 1875	CD77 35 10 CD79 34 02 CD7B A6 00	PULS X PSHS A LDA FCBTYP,X	GET FCB POINTER BACK SAVE ACCA ON STACK * GET FILE TYPE OF THIS FCB AND
1874 1875 1876	CD77 35 10 CD79 34 02 CD7B A6 00 CD7D 81 40	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL	GET FCB POINTER BACK SAVE ACCA ON STACK * GET FILE TYPE OF THIS FCB AND * CHECK TO SEE IF IT'S A RANDOM FILE
1874 1875 1876 1877	CD77 35 10 CD79 34 02 CD7B A6 00 CD7D 81 40 CD7F 35 02	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A	GET FCB POINTER BACK SAVE ACCA ON STACK * GET FILE TYPE OF THIS FCB AND * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA
1874 1875 1876 1877 1878	CD77 35 10 CD79 34 02 CD7B A6 00 CD7D 81 40	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL	GET FCB POINTER BACK SAVE ACCA ON STACK * GET FILE TYPE OF THIS FCB AND * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE
1874 1875 1876 1877 1878 1879	CD77 35 10 CD79 34 02 CD7B A6 00 CD7D 81 40 CD7F 35 02	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A	GET FCB POINTER BACK SAVE ACCA ON STACK * GET FILE TYPE OF THIS FCB AND * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA
1874 1875 1876 1877 1878 1879 1880	CD77 35 10 CD79 34 02 CD7B A6 00 CD7D 81 40 CD7F 35 02	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A	GET FCB POINTER BACK SAVE ACCA ON STACK * GET FILE TYPE OF THIS FCB AND * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE *IS THE LENGTH OF THE FILE
1874 1875 1876 1877 1878 1879 1880 1881	CD77 35 10 CD79 34 02 CD7B A6 00 CD7D 81 40 CD7F 35 02	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A * * CALCULATE LOF FOR A RANDO	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE
1874 1875 1876 1877 1878 1879 1880	CD77 35 10 CD79 34 02 CD7B A6 00 CD7D 81 40 CD7F 35 02	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884	CD77 35 10 CD79 34 02 CD78 A6 00 CD70 81 40 CD77 35 02 CD81 26 B7 CD83 34 10 CD83 34 10 CD85 93 8A	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCO (NUMBER OF SECTORS)
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884	CD77 35 10 CD79 34 02 CD78 A6 00 CD7D 81 40 CD7F 35 02 CD81 26 B7 CD83 34 10 CD85 93 8A CD87 27 03	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FFOM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886	CD77 35 10 CD79 34 02 CD78 A6 00 CD7D 81 40 CD7F 35 02 CD81 26 B7 CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01	PULS X PSHS A LDA FGBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDC * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCO (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE IOOZ USED
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886	CD77 35 10 CD79 34 02 CD78 A6 00 CD70 81 40 CD77 35 02 CD81 26 B7 CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCO (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPA®
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887	CD77 35 10 CD79 34 02 CD78 A6 00 CD7D 81 40 CD7F 35 02 CD81 26 B7 CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8C 06 4F	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC SUBD #\$0001 LCDBC SUBD #\$0001 LCDBC BSR LCD3A LDB FP0EXP	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE IOOZ USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888	CD77 35 10 CD79 34 02 CD78 A6 00 CD7D 81 40 CD7F 35 02 CD81 26 B7 CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E D6 4F CD90 27 04	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDC * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE IOOZ USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8C 06 4F CD90 27 04 CD92 CB 08	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890	CD77 35 10 CD79 34 02 CD78 A6 00 CD7D 81 40 CD7F 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC SUBD #\$0001 LCDBC SUBD #\$0001 LCDBC BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FFOM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE IOOZ USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ B  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD71 35 02 CD81 26 87  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC SUBD #\$0001 LCDBC BSR LCD3A LDB FF0EXP BEQ LCD6 ADDB #\$08 STB FP0EXP LCD96 ADBB #\$08 STB FFPEXP LCD96 JSR LBC5F	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP LCD96 STB FP0EXP LCD96 JSR LBC5F LDX , S	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1888 1889 1890 1891 1892	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD71 35 02 CD81 26 87  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC SUBD #\$0001 LCDBC BSR LCD3A LDB FF0EXP BEQ LCD6 ADDB #\$08 STB FP0EXP LCD96 ADBB #\$08 STB FFPEXP LCD96 JSR LBC5F	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1887 1898 1899 1890 1891 1892 1893	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 B0 BC 5F CD99 AE E4 CD98 C 88 13	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC SUBD #\$0001 LCDBC SUBD #\$0001 LCDBC BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP LCD96 JSR LBC5F LDX , S LDD FCBLST, X	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1887 1889 1890 1891 1892 1893 1894	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD99 AE E4 CD99 EC 88 13 CD9E 84 7F	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 STB FF0EXP LCD96 ADDB #\$78 LCD96 LCD96 ADDB #\$78	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE
1874 1875 1876 1877 1878 1889 1881 1882 1883 1884 1885 1886 1887 1898 1899 1891 1892 1893 1894 1895	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AC E4 CD98 E6 88 13 CD9E 84 7F CD48 8D 98	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 LDX ,S LDD FCBLST,X ANDA #\$7F BSR LCD3A	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ  GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1889 1891 1892 1893 1894 1895 1896 1897 1898	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 13 CD9E 84 7F CD48 80 98 CD44 96 5C CDA6 06 4F	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 ADDB #\$75 BSR LCD3A CLR RESSGN LDA FF1EXP LDB FF0EXP	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ  GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPAI PRIOR TO ADDITION
1874 1875 1876 1877 1879 1880 1881 1882 1883 1884 1885 1886 1887 1891 1892 1892 1893 1894 1895 1896 1897 1898	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 13 CD9E 84 7F CDA0 80 98 CDA2 0F 62 CDA4 96 5C	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001  LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 ADDB #\$76 BEQ LCD96 ADDB #\$76 BEQ LCD96 ADDB #\$76 BEQ LCD96 ADDB #\$77 BER LEC5F LDX , S LDD FCBLST, X ANDA #\$77 BSR LCD3A CLR RESSGN LDA FP1EXP LDB FP0EXP LB9C5	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A CA  *IF NOT CA  *IF C
1874 1875 1876 1877 1878 1889 1881 1882 1883 1884 1885 1886 1887 1898 1890 1891 1892 1893 1894 1895 1896 1897 1898 1899 1990	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD71 81 40 CD71 85 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 07 4F CD96 ED 8C 5F CD99 AE E4 CD98 EC 88 13 CD91 84 7F CD80 80 98 CD84 96 5C CD84 96 5C CD84 96 5C CD84 96 5C CD86 BD 8C 5F	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP LCD96 JSR LBC5F LDX ,S LDD FCBLST,X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FP0EXP LDB FF0EXP STB LCD3A CR RESSGN LDA FP1EXP LDB FF0EXP LBBC5	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT TONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD B TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS
1874 1875 1876 1877 1879 1880 1881 1882 1883 1884 1885 1886 1887 1899 1891 1892 1893 1894 1895 1896 1897 1899 1990 1901	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 CD94 D7 4F CD96 BD BC 5F CD99 BC 88 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 13 CD9E 84 7F CDA0 80 98 CDA2 0F 62 CDA4 96 5C CDA6 06 4F CDA6 BD BC 5F	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 STB FF0EXP LCD96 ADDB #\$08 STB FF0EXP LCD96 LDX ,S LDD FCBLST,X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FF1EXP LDB FF0EXP JSR LBC5F	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPAI PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1887 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD75 35 02 CD81 26 87  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD95 BC 88 13 CD96 84 7F CDA0 80 98 CDA2 07 62 CDA4 96 5C CDA4 96 5C CDA4 96 5C CDA6 D6 4F CDA8 BD BC 5F CDA8 BD BC	PULS X PSHS A PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC SUBD #\$0001 LCDBC BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP BEQ LCD96 ADDB #\$78 LCD96 ADB #\$78 LDD FCBLST, X ANDA #\$7F BSR LCD3A CLR RESSON LDA FP1EXP LDB FF0EXP BSR LCD3A CLR RESSON LDA FF1EXP LDB FF0EXP LBBC5F LBC5F LBC5	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1889 1899 1891 1892 1893 1894 1895 1896 1897 1991 1902 1901	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 ED 8C 5F CD99 AE E4 CD98 EC 88 13 CD95 EA 7F CDA0 80 98 CDA2 0F 62 CDA4 96 5C CDA4 96 5C CDA6 BB BC 5F CDA8 BD BC 5F CDA9 BC 5C CDA6 BB BC 5F CDA8 BD BC 5F CDA9 55 C CDA8 BD BC 5F CDA8 CD86 CC 09	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP LCD96 JSR LEC5F LDX ,S LDD FCBLST,X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FP0EXP LDB FF0EXP STB LCD3A CR RESSGN LDA FP1EXP LDB FF0EXP LBBC5F PULS X LBC5F	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT TONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD B TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH
1874 1875 1876 1877 1879 1880 1881 1882 1883 1884 1885 1886 1887 1899 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 CD94 07 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 CD94 07 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 CD94 07 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 CD94 07 4F CD98 BD BC 5F CDA8 BD BC 5C CDA6 D6 4F CDA8 BD BC 5F CDA6 D6 4F CDA6 D6 4F CDA8 BD BC 5F CDA6 D7 CDB6 EC 09 CDB8 BD BC 5F CDA6 D7 CDB7 BB BD BC 5F CDA6 D7 CDB7 BB BD BC 5F CDA6 D7 CDB7 BB BD BC 5F CDA6 D8 C6 09 CDB8 BD BC 5F CDB8 BD BC 5F CDB8 EC 09 CDB	PULS X PSHS A PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC SUBD #\$0001 LCDBC BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP BEQ LCD96 ADDB #\$78 LCD96 ADB #\$78 LDD FCBLST, X ANDA #\$7F BSR LCD3A CLR RESSON LDA FP1EXP LDB FF0EXP BSR LCD3A CLR RESSON LDA FF1EXP LDB FF0EXP LBBC5F LBC5F LBC5	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ  GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPAI PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH  * PUT IT INTO FPAØ
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1889 1899 1891 1892 1893 1894 1895 1896 1897 1991 1902 1901	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 ED 8C 5F CD99 AE E4 CD98 EC 88 13 CD95 EA 7F CDA0 80 98 CDA2 0F 62 CDA4 96 5C CDA4 96 5C CDA6 BB BC 5F CDA8 BD BC 5F CDA9 BC 5C CDA6 BB BC 5F CDA8 BD BC 5F CDA9 55 C CDA8 BD BC 5F CDA8 CD86 CC 09	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001  LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 ANDA #\$7F BSR LCD3A CLR RESSGN LDA FF1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT TONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD B TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1887 1892 1893 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1906	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 ED BC 5F CD99 AE E4 CD95 E4 7F CDA0 80 98 CD40 D7 4F CDA0 80 98 CDA2 0F 62 CDA4 96 5C CDA6 D6 4F CDA8 BD BC 5F CDA9 BC 6P CDB9	PULS X PSHS A PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC SUBD #\$0001 LCDBC BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP BEQ LCD96 ADDB #\$78 LDD FCBLST,X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP LDB FF0EXP BSR LCD3A CLR RESSGN LDA FF1EXP LDB FF0EXP LBBC5F PULS X LDD FCBRIN,X LBC5F PULS X LDD FCBRIN,X LBC5F PULS X LDD FCBRIN,X LCD3A CLR RESSGN LCD3A CLR RESSGN LCD3A CLR RESSGN LCD3A CLR RESSGN	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH  * PUT IT INTO FPAØ FORCE QUOTIENT SIGN = POSITIVE
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1889 1890 1891 1892 1893 1894 1895 1896 1897 1901 1902 1903 1904 1905 1906	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 93 CD89 83 00 01 CD8C 8D AC CD8E 16 4F CD90 27 04 CD92 CB 08 CD94 07 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 13 CD95 84 7F CDA0 80 98 CD44 96 5C CDA4 96 5C CDA4 96 5C CDA6 BB BC 5F CD98 BB BC 5F CD98 BB BC 5F CDA9 85 CC CDA6 BB BC 5F CDA9 85 CC CDA6 BB BC 5F CD88 BB BC 5F CD88 BB BC 5F CD88 BB BC 5F CD88 CC 99 CD82 BB 86 CD84 0F 62 CD86 0F 62 CD86 BC 69 CD86 0F 62	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001  LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP LCD96 JSR LBC5F LDX ,S LDD FCBLST, X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FP0EXP JSR LBC5F PULS X LBC	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET REPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH  * PUT IT INTO FPAØ FORCE QUOTIENT SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND
1874 1875 1876 1877 1879 1880 1881 1882 1883 1884 1885 1886 1887 1899 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1905 1906 1907	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD83 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 CD94 07 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 CD94 07 4F CD98 EC 88 CD94 07 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 CD94 07 4F CD98 EC 88 CD94 07 4F CD98 EC 88 CD94 07 5C CD88 CD84 07 6C CD86 D6 4F CD88 BD BC 5F CD88 BD BC 5F CD88 BD BC 5F CD88 BB BC 5F CD89 EC 09 CD89 BB BC 5C CD89 BC 69 CD89 BB BC 5C CD86 06 5C CD88 BB BC 5C CD86 D6 5C CD88 BB BC 5C CD86 D6 5C CD88 CD86 F CD86 CB6 5C CD86 C	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001  LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 ADDB FF0EXP LCD96 ASSON LDA FF1EXP LCD96 ASSON LCD FCBRLN, X BSR LCD3A CLR RESSON LCD FF1EXP LCD FCBRLN, X BSR LCD3A CLR RESSON LCD FF1EXP	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH  * PUT IT INTO FPAØ FORCE QUOTIENT SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO DIVISION
1874 1875 1876 1877 1888 1889 1881 1882 1883 1884 1885 1886 1887 1899 1899 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1909 1909 1909 1909 1909 1909 1909 1909 1909 1910 1911	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 ED BC 5F CD99 AE E4 CD98 E4 7F CDA0 80 98 CD40 D7 4F CDA0 80 98 CDA2 0F 62 CDA4 96 5C CDA6 D6 4F CDA8 BD BC 5F CDA8 BD BC 5F CDB8 BD BB BB 91	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001  LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 ADDB FF0EXP LCD3A CLR RESSGN LDA FF1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN, X BSR LCD3A CLR RESSGN LDA FF1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN, X BSR LCD3A CLR RESSGN LDA FF1EXP LDB FF0EXP JSR LBB91 JMP INT	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH  * PUT IT INTO FPAØ FORCE QUOTIENT SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO DIVISION  DIVIDE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1887 1891 1892 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1906 1907 1909 1910 1911 1912	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 8D BC 5F CD99 AE E4 CD98 EC 88 13 CD9E 84 7F CDA0 8D 98 CDA2 06 65 CDA4 96 5C CDA6 06 4F CDA8 BD BC 5F CD98 BD BC 5F CD99 8C 5C CDA6 06 4F CDA8 BD BC 5F CD89 BC 62 CDA9 65 C CDA6 06 4F CDA8 BD BC 5F CD88 BD BC 5F CD89 BC 62 CDA9 65 C CDA6 06 4F CDA8 BD BC 5F CDB8 BC 62 CDA6 96 5C CDA6 96 5C CDA6 96 5C CDB6 96 5C CDB6 96 5C CDB6 96 5C CDB8 D BC 5F CDB8 BC 64 CDB8 BC 64 CDB8 BC 67 CDB8 CDB8 CDB8 CDB8 CDB8 CDB8 CDB8 CDB8	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCDBC SUBD #\$0001 LCDBC BSR LCD3A LDB FP0EXP BEQ LCDBC ADDB #\$08 STB FF0EXP BEQ LCD96 ADDB #\$78 LCD96 ADDB #\$808 STB FP0EXP LDX ,S LDD FCBLST, X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP LBB FF0	GET FGB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH  * PUT IT INTO FPAØ FORCE QUOTIENT SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO DIVISION DIVIDE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER
1874 1875 1876 1877 1878 1889 1881 1882 1883 1884 1885 1886 1887 1899 1890 1891 1892 1893 1894 1895 1896 1899 1903 1904 1905 1907 1908 1907 1908 1907 1908 1909 1910 1911 1912 1913	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 93 CD89 83 00 01 CD8C 8D AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 07 4F CD96 2D 8C 5F CD99 AE E4 CD98 EC 88 13 CD95 84 7F CDA0 8D 8C 5F CDA0 8D 8C 5F CDA0 96 5C CDA4 96 5C CDA4 96 5C CDA6 D6 4F CDA8 8D 8C 5F CD88 CC 09 CD88 CC 09 CD82 CD 96 5C CDA6 D6 4F CDB8 CC 09 CD82 CD 96 5C CDB8 D6 5C CDB8 CC 09 CD82 B0 86 CD84 0F 62 CD86 96 5C CD86 96 5C CD86 96 5C CD86 96 5C CD88 B0 B1 43	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001  LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP BEQ LCD96 ADDB #\$7F BCQ LCD96 ADDB #\$7F BCG LD3A LDB FP0EXP BCQ LCD96 ADDB #\$08 STB FP0EXP LCD96 JSR LBC5F LDX ,S LDD FCBLST, X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FP0EXP JSR LBC5F PULS X LBD FCBRLN, X BSR LCD3A CLR RESSGN LDD FF0EXP JSR LBB91 JMP INT  * FREE COMMAND FREE JSR LB143	GET FCB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA  *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD B TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO DOITION  ADD TO FPAØ FORCE QUOTIENT SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO DIVISION  DIVIDE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER  * NUMBER TYPE CHECK
1874 1875 1876 1877 1888 1889 1881 1882 1883 1884 1885 1886 1887 1899 1899 1893 1894 1895 1896 1897 1898 1899 1900 1901 1902 1903 1904 1905 1906 1907 1908 1909 1911 1912 1911 1912 1911 1912 1911 1912 1913 1914	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 B7  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 80 AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD98 EC 88 13 CD9E 84 7F CD40 80 BS CD44 96 5C CDA6 06 4F CDA6 06 4F CDA6 06 4F CDA6 07 62 CDA6 06 4F CDA6 06 5C CDA6 06 4F CDA6 06 5C CDA6 06 4F CDA6 06 4F CDA6 07 CDB6 06 4F CDB7 07 CDB7	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001  LCD8C SUBD #\$0001  LCD8C SUBD #\$0001  LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FP0EXP LCD96 ADDB FF0EXP LDX S LDD FCBLST, X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FP0EXP JSR LBC5F PULS X LDD FCBRLN, X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN, X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBB91 JMP INT  * FREE COMMAND FREE JSR LB143 JSR LB70E	GET FOB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCO (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE IOOZ USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH  * PUT IT INTO FOB FORCE QUOTIENT SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  DIVIDE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER  * NUMBER TYPE CHECK  *EVALUATE NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1886 1887 1898 1899 1890 1891 1892 1893 1894 1895 1896 1897 1902 1903 1904 1907 1908 1907 1908 1909 1910 1911 1912 1913 1914 1915	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD75 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD86 80 AC CD86 106 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 8D 8C 5F CD99 AE E4 CD98 EC 88 13 CD95 84 7F CDA0 80 98 CDA2 0F 62 CDA4 96 5C CDA6 06 4F CDA8 8D 8C 5F CD98 8D 8C 5F CD99 BC 88 13 CD92 81 7F CDA0 80 98 CDA2 0F 62 CDA4 96 5C CDA6 96 5C CDB6 96 5C CDB6 96 5C CDB6 96 5C CDB8 96 5C CDB8 96 5C CDB8 97 62 CDB8 98 91 CDB0 76 8C CDB8 98 91 CDB0 77 8C CDB8 B0 87 06 CDC0 80 81 43 CDC3 8D 87 06 CDC6 C1 03	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FP0EXP BEQ LCD8C ADDB #\$001 LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 ADDB #\$76 ADDB #\$78 LCD96 ADDB #\$78 LCD96 ADDB #\$98 STB FF0EXP LDX ,S LDD FCBLST,X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP LDB FF0EXP LBB LB143 LB70E CMPB #\$03	GET FOB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY  * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  -ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF -BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH  * PUT IT INTO FPAØ FORCE QUOTIENT SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO DIVISION DIVIDE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER  * NUMBER TYPE CHECK  *EVALUATE NUMBER CEXPRESSION AND RETURN VALUE IN ACCB ONLY 4 LEGAL DRIVES
1874 1875 1876 1877 1878 1889 1881 1882 1883 1884 1885 1886 1887 1899 1891 1892 1893 1894 1895 1896 1897 1898 1899 1901 1902 1903 1904 1905 1906 1907 1908 1909 1909 1910 1911 1912 1913 1914 1915 1916	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD77 35 02 CD81 26 87  CD83 34 10 CD85 93 8A CD87 27 93 CD89 83 00 01 CD8C 80 AC CD8E 06 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 8D 8C 5F CD99 AE E4 CD98 EC 88 13 CD95 84 7F CDA0 8D 98 CDA2 0F 62 CDA4 96 5C CDA5 8D 8C 5F CD88 EC 88 13 CD92 8F 62 CDA4 96 5C CDA6 D6 4F CDA8 BD 8C 5F CDB8 EC 88 CD84 07 65 CDA8 BD 8C 5F CDB8 EC 88 CD84 07 65 CDA8 BD 8C 5F CDA9 CD80 EC 89 CD80 EC 89 CD80 EC 89 CD80 BC 87 CD80 EC 87 CD80 EC 89 CD80 BC 87 CD80 EC 87 CD80 BC	PULS X PSHS A LDA FCBTYP, X CMPA #RANFIL PULS A BNE LCO3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCOBC SUBD #\$0001  LCDBC BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 JSR LBC5F LDX ,S LDD FCBLST, X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X SECONDA LDB FF0EXP LBC5F PULS X LBC5F PULS X BSR LCD3A CLR RESSGN LDD FCBLST, X BSR LCD3A CLR RESSGN LDD FCBRLN, X BSR LCD3A CLR RESSGN LB70E LB91 JSR LB70E L	GET FOB POINTER BACK SAVE ACCA ON STACK  * GET FILE TYPE OF THIS FCB AND  * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE  *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE  FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCO (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE IOOZ USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø  * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO ADDITION  —ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF —BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH  * PUT IT INTO FPAØ FORCE QUOTIENT SIGN = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO JOURISION DIVIDE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER  * NUMBER TYPE CHECK *EVALUATE NUMBER! C EXPRESSION AND RETURN VALUE IN ACCB ONLY 4 LEGAL DRIVES 'DEVICE NUMBER' ERROR IF DRIVE NUMBER IS > 3
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1887 1899 1890 1891 1992 1993 1906 1907 1908 1909 1911 1912 1913 1914 1915 1916	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD75 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD86 80 AC CD86 B0 64F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD96 BB BC 5F CD99 AE E4 CD96 BB BC 5F CD98 BB BC 5F CD98 BB BC 5F CD98 BB BC 5F CD98 BB BC 5F CD88 BB BC 5F CD88 BB BC 5F CD88 BB BC 5F CD86 BB BC 5F CD87 BC 62 CD86 BB BC 5F CD88 BC	PULS X PSHS A PSHS A LOAF FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BEQ LCD8C ADDB #\$0001 LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP BEQ LCD96 ADDB #\$700 BEC LCD96 BEC LCD96 ADDB #\$700 BEC LCD96 BEC	GET FCB POINTER BACK SAVE ACCA ON STACK * GET FILE TYPE OF THIS FCB AND * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF FRAØ = Ø  * ADD B TO EXPONENT OF FRAØ BRANCH IF FPAØ = Ø  * ADD B TO EXPONENT (MULTIPLY FPAØ BY * 256 BYTES/SECTOR, AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPAI POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SITEN = POSITIVE * GET EXPONENTS OF FPAØ AND * FPAI PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPAI POINT X TO FCB * GET RECORD LENGTH * PUT IT INTO FPAØ FORCE SUM SIGN = POSITIVE * GET EXPONENTS OF PAØ AND * FPAI PRIOR TO ADDITION  =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER  * NUMBER TYPE CHECK *EVALUATE NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER  * NUMBER TYPE CHECK *EVALUATE NUMBER FROR IF DRIVE NUMBER IS > 3 SAVE IN DRIVE NUMBER SAVE IN DRIVE NUMBER
1874 1875 1876 1877 1878 1879 1880 1881 1882 1883 1884 1885 1889 1899 1890 1891 1892 1893 1894 1895 1896 1897 1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD75 35 02 CD81 26 87  CD82 38 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD8C 8D AC CD8E D6 4F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 ED BC 5F CD99 AE E4 CD98 EC 88 13 CD96 84 7F CDA0 80 98 CDA2 0F 62 CDA4 96 5C CDA6 D6 4F CDA8 BD BC 5F CD98 EC 88 13 CD92 BB C5 CDA6 96 5C CDA6 D6 4F CDA8 BD BC 5F CD88 BD BC 5F CD89 EC 88 13 CDC0 BB BC 5F CD80 BC 62 CDA9 96 5C CDA6 D6 4F CDA8 BD BC 5F CDB8 BD BC 5F CDB9 EC 88 CD84 0F 62 CDB8 BD BC 5F CDB8 BC 5F CD8 BC 5F CD9	PULS X PSHS A LDA FCBTYP,X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BSR LCD3A LDB FP0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP LCD96 ADDB #\$70 BEQ LCD96 ADDB #\$70 BEQ LCD96 ADDB #\$88 STB FF0EXP LCD96 ADDB #\$70 BEQ LCD96 ADDB #\$80 BEQ LCD96 ADDB #\$80 STB FF0EXP LDD FCBLST,X ANDA #\$7F BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F PULS X LDD FCBRLN,X BSR LCD3A CLR RESSGN LDA FP1EXP LDB FF0EXP JSR LBC5F LDB FF0EXP JSR LBC5F LDB FF0EXP JSR LBC5F LDB FF0EXP	GET FCB POINTER BACK SAVE ACCA ON STACK * GET FILE TYPE OF THIS FCB AND * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ONE SECTOR - THE LAST SECTOR MAY NOT BE 100Z USED PUT ACCD INTO FPAØ GET EXPONENT OF FPAØ BRANCH IF FPAØ = Ø * ADD 8 TO EXPONENT (MULTIPLY FPAØ BY * 256 BYTES/SECTOR) AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPA1 POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR TO NUMBER OF FORCE SUM SIGN = POSITIVE * GET EXPONENTS OF FPAØ AND * FPA1 PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPA1 POINT X TO FCB  * GET RECORD LENGTH * PUT IT INTO FPAØ FORCE QUOTIENT SIGM = POSITIVE  * GET EXPONENTS OF FPAØ AND  * FPA1 PRIOR TO DIVISION DIVIDE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER  * NUMBER TYPE CHECK *EVALUATE NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD ONLY 4 LEGAL DRIVES 'DEVICE NUMBER' ERROR IF DRIVE NUMBER IS > 3 SAVE IN DRIVE NUMBER GET FILE ALLOCATION TABLE AND STORE IN BUFFER
1874 1875 1876 1877 1878 1880 1881 1882 1883 1884 1885 1886 1887 1899 1890 1891 1992 1993 1906 1907 1908 1909 1911 1912 1913 1914 1915 1916	CD77 35 10 CD79 34 02 CD78 46 00 CD70 81 40 CD75 35 02 CD81 26 87  CD82 34 10 CD85 93 8A CD87 27 03 CD89 83 00 01 CD86 80 AC CD86 B0 64F CD90 27 04 CD92 CB 08 CD94 D7 4F CD96 BD BC 5F CD99 AE E4 CD96 BB BC 5F CD99 AE E4 CD96 BB BC 5F CD98 BB BC 5F CD98 BB BC 5F CD99 BC 5F CD99 AE E4 CD96 CD86 BB BC 5F CD98 E5 BB 13 CD96 BC 5F CD88 BB BC 5F CD88 BC 5F CD8	PULS X PSHS A PSHS A LOAF FCBTYP, X CMPA #RANFIL PULS A BNE LCD3A  *  * CALCULATE LOF FOR A RANDO * NUMBER OF RECORDS IN THE PSHS X SUBD ZERO BEQ LCD8C SUBD #\$0001 LCD8C BEQ LCD8C ADDB #\$0001 LCD8C BSR LCD3A LDB FF0EXP BEQ LCD96 ADDB #\$08 STB FF0EXP BEQ LCD96 ADDB #\$700 BEC LCD96 BEC LCD96 ADDB #\$700 BEC LCD96 BEC	GET FCB POINTER BACK SAVE ACCA ON STACK * GET FILE TYPE OF THIS FCB AND * CHECK TO SEE IF IT'S A RANDOM FILE RESTORE ACCA *IF NOT A RANDOM FILE, THEN THE TOTAL NUMBER OF SECTORS IN THE FILE *IS THE LENGTH OF THE FILE  M FILE - THE LENGTH OF A RANDOM FILE IS THE FILE.  SAVE FCB POINTER ON STACK SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF ZERO SECTORS SUBTRACT ZERO FROM ACCD (NUMBER OF SECTORS) BRANCH IF FRAØ = Ø  * ADD B TO EXPONENT OF FRAØ BRANCH IF FPAØ = Ø  * ADD B TO EXPONENT (MULTIPLY FPAØ BY * 256 BYTES/SECTOR, AND SAVE NEW EXPONENT SAVE NUMBER OF BYTES IN FULL SECTORS IN FPAI POINT X TO FCB GET NUMBER OF BYTES IN LAST SECTOR MASK OFF THE PRE-SAVED BYTE PUT NUMBER BYTES IN LAST SECTOR INTO FPAØ FORCE SUM SITEN = POSITIVE * GET EXPONENTS OF FPAØ AND * FPAI PRIOR TO ADDITION =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES IN FPAI POINT X TO FCB * GET RECORD LENGTH * PUT IT INTO FPAØ FORCE SUM SIGN = POSITIVE * GET EXPONENTS OF PAØ AND * FPAI PRIOR TO ADDITION  =ADD NUMBER BYTES IN LAST SECTOR TO NUMBER OF =BYTES IN FULL SECTORS SAVE TOTAL NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER  * NUMBER TYPE CHECK *EVALUATE NUMBER OF BYTES BY NUMBER OF BYTES IN A RECORD CONVERT FPAØ TO AN INTEGER  * NUMBER TYPE CHECK *EVALUATE NUMBER FROR IF DRIVE NUMBER IS > 3 SAVE IN DRIVE NUMBER SAVE IN DRIVE NUMBER

1921	CDD6 6F E2		CLR ,-S	SPACE FOR FREE GRANULE COUNTER
1922	CDD6 C6 44		LDB #GRANMX	GET MAXIMUM NUMBER OF GRANULES
1923	CDDA A6 80 CDDC 43	LCDDA	LDA ,X+ COMA	GET GRANULE DATA *FREE GRANULES \$FF
1925			BNE LCDE1	*BRANCH IF NOT FREE
	CDDF 6C E4		INC ,S	INCREMENT FREE GRANULE COUNTER
	CDE1 5A CDE2 26 F6	LCDE1	DECB BNE LCDDA	DECREMENT GRANULE COUNTER BRANCH IF NOT DONE
1929			PULS B	GET FREE GRANULE COUNTER TO ACCB
1930	CDE6 7E B4 F3		JMP LB4F3	LOAD ACCB INTO FPAØ
1931 1932		* DRIVE	E COMMAND	
1933	CDE9 BD B7 ØB			EVALUATE EXPR; RETURN VALUE IN ACCB
	CDEC C1 Ø3		JSR EVALEXPB CMPB #\$Ø3	MAX DRIVE NUMBER = 3
	CDEE 10 22 D8 2D CDF2 F7 09 5A		LBHI LA61F STB DEFDRV	'DEVICE #' ERROR IF DRIVE NUMBER > 3 SAVE DEFAULT DRIVE NUMBER
1937	CDF5 39		RTS	SAVE DEFAUET BRIVE NORDER
1938		+ FVALU	JATE EXPRESSION RAM VECTOR	
1939 1940	CDF6 A6 64		LDA \$Ø4,S	= CHECK STACKED PRECEDENCE FLAG AND IF IT IS NOT AN END
1941	CDF6 A6 64 CDF8 26 13			= OF OPERATION, BRANCH TO EXTENDED BASIC'S EXPRESSION
1942	CDEA AF 6F	*	IDV ¢ar c	= EVALUATION ROUTINE
1943	CDFA AE 65 CDFC 8C AF 9A CDFF 26 ØC		LDX \$05,S CMPX #LAF9A	*
1945	CDFF 26 ØC CEØ1 AE 62		CMPX #LAF9A BNE LCEØD LDX \$02,S CMPX #LB166 BNE LCEØD LDX #LCEIØ STX \$05,S	* CHECK TWO RETURN ADDRESSES BACK ON THE STACK
			LDX \$02,S	* TO SEE IF THE CALL TO EVALUATE EXPRESSION IS
	CEØ3 8C B1 66 CEØ6 26 Ø5		BNE LCEØD	* COMING FROM THE "LET" COMMAND - BRANCH OUT IF  * NOT COMING FROM 'LET'
1949	CEØ8 8E CE 1Ø		LDX #LCE1Ø	= IF COMING FROM 'LET', REPLACE THE RETURN ADDR
	CEØB AF 65	LCEAD	STX \$05,S	= WITH THE DISK BASIC 'LET' MODIFIER ADDRESS
1951	CEØD 7E 88 46	LCEND	JMP XVEC15	EXTENDED BASIC EXPRESSION EVALUATION
1953			10DIFIER	= EVALUATION ROUTINE  *  *  * CHECK TWO RETURN ADDRESSES BACK ON THE STACK  * TO SEE IF THE CALL TO EVALUATE EXPRESSION IS  * COMING FROM THE 'LET' COMMAND - BRANCH OUT IF  * NOT COMING FROM 'LET'  = IF COMING FROM 'LET', REPLACE THE RETURN ADDR  = WITH THE DISK BASIC 'LET' MODIFIER ADDRESS  EXTENDED BASIC EXPRESSION EVALUATION  PULL VARIABLE TYPE OFF OF THE STACK  SET CARRY IF SIRING, CLEAR CARRY IF NUMERIC  DO A 'TM' CHECK
1954	CE10 35 02 CE12 46		PULS A RORA	PULL VARIABLE TYPE OFF OF THE STACK
1955	CE13 BD B1 48		RORA JSR LB148 LBEQ LBC33 LDX FPAØ+2 LDD \$02,X CMPD #DFLBUF BLO LCE2B SUBD FCBADR	DO A 'TM' CHECK
1957	CE16 10 27 EE 19		LBEQ LBC33	IF NUMERIC VARIABLE, PACK FPAU INTO VARDES
1958	CE1A 9E 52		LDX FPAØ+2	POINT X TO STRING DESCRIPTOR
1959	CEIC EC 02 CEIE 10 83 09 89		CMPD #DFLBUF	GET ADDRESS OF SIRING * COMPARE TO START OF RANDOM FILE BUFFERS
1961	CE22 25 07 CE24 B3 09 4A		BLO LCE2B	* AND BRANCH IF LOWER
	CE24 B3 Ø9 4A CE27 1Ø 25 E1 86			SUBTRACT OUT THE END OF RANDOM FILE BUFFERS
1963 1964	CE2/ 10 25 E1 86	*	LBCS LAFB1	*BRANCH IF STRING STORED IN RANDOM FILE BUFFER - *MOVE IT INTO THE STRING SPACE
1965	CE2B 7E AF A4	LCE2B	JMP LAFA4	BRANCH BACK TO BASIC S 'LET' COMMAND
1966 1967		*M0D1E1	ER FOR EXBAS COMMAND INTERP	DETATION HANDIED
	CE2E 81 CA	DXCVEC	CMPA #\$CA	TOKEN FOR DLOAD?
1969	CE2E 81 CA CE3Ø 27 1C CE32 81 C8	DXCVEC	BEQ LCE4E	YES
1970	CE32 81 C8 CE34 10 26 B3 04		CMPA #\$C8 LBNE L813C	TOKEN FOR PMODE? NO
1972		* DISK	BASIC MODIFIER FOR PMODE -	ALLOWS FOR THE RAM THE DOS USES
	CE38 9D 9F		JSR GETNCH	GET NEXT CHARACTER FROM BASIC
1974	CE3A 81 2C CE3C 10 27 C8 10		CMPA #',' LBEQ L9650	CHECK FOR COMMA BRANCH IF COMMA
1976	CE4Ø BD B7 ØB		JSR EVALEXPB CMPB #\$04	EVALUATE EXPRESSION; RETURN VALUE IN ACCB
	CE43 C1 Ø4		CMPB #\$Ø4	CHECK FOR PMODE 4
1978	CE45 10 22 E6 01 CE49 96 BC		LBHI LB44A LDA GRPRAM	'FC' ERROR IF PMODE > 4 NUMBER BLOCKS BEFORE GRAPHICS PAGES
1980	CE4B 7E 96 2E		JMP L962E	JUMP TO EXEAS' PMODE COMMAND
1981		+ DICK	DACTO DI GAD MODIFIED	
1982 1983	CE4E BD A4 29	LCE4E	BASIC DLOAD MODIFIER JSR LA429	CLOSE FILES
1984	CE51 9D 9F		JSR GETNCH	GET NEXT CHARACTER FROM BASIC
1985	CE53 7E 8C 1B		JMP L8C1B	JUMP TO EXEAS' DLOAD
1986 1987	CE56 C1 34	DXIVEC	CMPB #(\$9A-\$8Ø)*2 LBNE L8168 JSR LB262 LDA DEVNUM PSHS A JSR LA5AE JSR LA4Ø6 TST DEVNUM BLE LCE8Ø JSR LC714 LDB FCBTYP,X CMPB #RANFIL BNE LCE8Ø PULS A STA DEVNUM LDD FCBPUT,X JMP GIVABF JSR LA35F PULS A STA DEVNUM LDB DEVPOS JMP LB4F3	MODIFIED TOKEN FOR POS
1988	CE58 10 26 B3 0C		LBNE L8168	IF NOT POS, GO TO EXBAS SECONDARY COMM HANDLER
1989	CE5C BD B2 62		JSR LB262	SYNTAX CHECK FOR '(' AND EVALUATE EXPRESSION
1990 1991	CEST 90 0F CE61 34 02		PSHS A	* GET DEVICE NUMBER AND * SAVE IT ON STACK
1992	CE63 BD A5 AE		JSR LA5AE	EVALUATE DEVICE NUMBER
1993	CE66 BD A4 Ø6		JSR LA4Ø6	TEST DEVICE NUMBER
1994	CE69 0D 6F CE6B 2F 13		ISI DEVNUM	* CHECK DEVICE NUMBER AND BRANCH * IF NOT A DISK FILE
1996	CE6D BD C7 14		JSR LC714	POINT X TO FCB
1997	CE70 E6 00		LDB FCBTYP,X	GET FILE TYPE
1998 1999	CE72 CI 40 CE74 26 ØA		CMLR #KWNLIF	DIRECT/RANDOM FILE? BRANCH IF NOT A RANDOM FILE
2000	CE76 35 Ø2		PULS A	* RESTORE DEVICE NUMBER
2001	CE78 97 6F		STA DEVNUM	*
2002 2003	CE/A EC 88 1/ CE7D 7E B4 F4		JMP GIVABE	=GRAB THE 'PUT' DATA ITEM COUNTER AND CONVERT =IT TO A FLOATING POINT NUMBER
2004	CE8Ø BD A3 5F	LCE8Ø	JSR LA35F	SET PRINT PARAMETERS
2005	CE83 35 Ø2		PULS A	* RESTORE DEVICE NUMBER
2006 2007	CE85 9/ 6F		SIA DEVNUM	* =GET PRINT POSITION AND
2008	CE89 7E B4 F3		JMP LB4F3	-GET PRINT POSITION AND -CONVERT IT TO FLOATING POINT NUMBER IN FPAØ
2010 2011	CE8C 9D 9F	* SAVEM LCE8C	1 COMMAND JSR GETNCH	GET NEXT INPUT CHARACTER
2012	CE8E 8D 4F		BSR LCEDF	GET FILENAME, ETC.
	CE9Ø BD 83 6C		JSR L836C	EVALUATE EXPRESSION, PUT II (2 BYTES) ON STACK
			100 10000	DITTO
2014	CE93 BD 83 6C		JSR L836C CMPX \$02 S	DITTO COMPARE END ADDRESS TO START ADDRESS
2014 2015			JSR L836C CMPX \$02,S LBCS LB44A	DITTO COMPARE END ADDRESS TO START ADDRESS IF START > END, THEN 'ILLEGAL FUNCTION CALL'

2017				
	CE9C BD 83 6C		JSR L836C	EVAL EXPRESSION (TRANSFER ADDRESS), PUT ON STACK
2018	CE9F BD A5 C7		JSR LA5C7	SYNTAX ERROR IF ANY MORE CHARS ON THIS LINE
2019	CEA2 CC Ø2 ØØ		LDD #\$0200	* FILE TYPE=2, ASCII FLAG = CRUNCHED (0)
2020 2021	CEA5 FD Ø9 57 CEA8 BD C9 56		STD DFLTYP JSR LC956	* GET NEXT UNOPEN FILE AND INITIALIZE FCB
2022	CEAB 4F		CLRA	*ZERO FLAG - FIRST BYTE OF PREAMBLE
2023	CEAC 8D 2B		BSR LCED9	*WRITE A BYTE TO BUFFER
2024	CEAE EC 62		LDD \$02,S	GET END ADDRESS
2025	CEBØ A3 64		SUBD \$04,S	SUBTRACT THE START ADDRESS
2026	CEB2 C3 ØØ Ø1		ADDD #\$0001	THE SAVED DATA BLOCK WILL INCLUDE BOTH THE FIRST AND LAST BYTES
2027 2028	CEB5 1F Ø2		TFR D,Y	SAVE LENGTH IN Y
2020	CEB7 8D 1E CEB9 EC 64		BSR LCED7 LDD \$04,S	WRITE FILE LENGTH TO BUFFER - FIRST ARGUMENT OF PREAMBLE GET THE START ADDRESS
2030	CEBB 8D 1A		BSR LCED7	WRITE OUT THE START ADDRESS - SECOND PREAMBLE ARGUMENT
2031	CEBD AE 64		LDX \$Ø4,S	GET START ADDRESS
2032	CEBF A6 8Ø	LCEBF	LDA ,X+	GRAB A BYTE
2033	CEC1 BD CB 52		JSR LCB52	WRITE IT OUT
2Ø34 2Ø35	CEC4 31 3F CEC6 26 F7		LEAY -1,Y BNE LCEBF	DECREMENT BYTE COUNTER BRANCH IF ALL BYTES NOT DONE
2036	CEC8 86 FF		LDA #\$FF	FIRST BYTE OF POSTAMBLE
2037	CECA 8D ØD		BSR LCED9	WRITE IT OUT - EOF RECORD
2038	CECC 4F		CLRA	* FIRST ARGUMENT OF POSTAMBLE IS
2039	CECD 5F		CLRB	* A DUMMY - ZERO VALUE
2040 2041	CECE 8D Ø7		BSR LCED7	WRITE OUT POSTAMBLE FIRST ARGUMENT
2041	CEDØ 35 36 CED2 8D Ø3		PULS A,B,X,Y BSR LCED7	GET CONTROL ADDRESSES FROM THE STACK WRITE OUT THE TRANSFER ADDRESS - 2ND ARGUMENT
2043	CED4 7E A4 2D		JMP LA42D	GO CLOSE ALL FILES
2044				
2045			ACCD TO THE BUI	
2046	CED7 8D 00	LCED7	BSR LCED9	WRITE ACCA TO BUFFER, THEN SWAP ACCA,ACCB
2Ø47 2Ø48	CED9 BD CB 52 CEDC 1E 89	LCED9	JSR LCB52 EXG A,B	WRITE ACCA TO BUFFER SWAP ACCA,ACCB
2040	CEDE 39		RTS	SWAF ACCA, ACCB
2050	CEDF 8E C2 97	LCEDF	LDX #BINEXT	POINT TO .BIN EXTENSION
2051	CEE2 7E C8 8A		JMP LC88A	GET FILENAME, ETC.
2052				
2053			COMMAND	
2Ø54 2Ø55	CEE5 9D 9F CEE7 8D F6	LCEE5	JSR GETNCH BSR LCEDF	GET NEXT INPUT CHARACTER GET FILENAME, ETC.
2056	CEE9 BD C9 59		JSR LC959	OPEN NEXT AVAILABLE FILE FOR INPUT
2057	CEEC FC Ø9 57		LDD DFLTYP	GET FILE TYPE AND ASCII FLAG
2058	CEEF 83 Ø2 ØØ		SUBD #\$0200	FOR LOADM FILE: TYPE=2, ASCII FLAG=0
2059	CEF2 10 26 D7 20		LBNE LA616	'BAD FILE MODE' ERROR
2060	CEF6 9E 8A		LDX ZERO	ZERO OUT X REG - DEFAULT VALUE OF OFFSET
2061 2062	CEF8 9D A5 CEFA 27 Ø6		JSR GETCCH BEQ LCFØ2	GET CURRENT CHARACTER FROM BASIC BRANCH IF END OF LINE – NO OFFSET
2063	CEFC BD B2 6D		JSR SYNCOMMA	SYNTAX CHECK FOR COMMA
2064	CEFF BD B7 3D		JSR LB73D	EVALUATE EXPRESSION
2065	CFØ2 9F D3	LCFØ2	STX VD3	STORE OFFSET IN VD3
2066	CFØ4 BD A5 C7		JSR LA5C7	SYNTAX ERROR IF OTHER CHARACTERS ON LINE
2067				
2067			DE 1 1 DE 1 DE 1 DE 1	
2068	CE07 DD CC E2		REAMBLE/POSTAMBI	
2Ø68 2Ø69	CFØ7 BD CC E2 CFØA 34 Ø2	* GET PI LCFØ7	JSR LCCE2	GET FIRST BYTE
2068	CFØ7 BD CC E2 CFØA 34 Ø2 CFØC 8D 29			
2068 2069 2070	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2		JSR LCCE2 PSHS A	GET FIRST BYTE SAVE IT ON THE STACK
2068 2069 2070 2071 2072 2073	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF1Ø 8D 25		JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT
2068 2069 2070 2071 2072 2073 2074	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF1Ø 8D 25 CF12 D3 D3		JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET
2068 2069 2070 2071 2072 2073 2074 2075	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF1Ø 8D 25 CF12 D3 D3 CF14 DD 9D		JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECJP	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND
2068 2069 2070 2071 2072 2073 2074 2075 2076	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF1Ø 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1		JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECJP TFR D,X	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X
2068 2069 2070 2071 2072 2073 2074 2075	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF1Ø 8D 25 CF12 D3 D3 CF14 DD 9D		JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECJP	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF1Ø 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1 CF18 A6 EØ	LCFØ7	JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECJP TFR D,X LDA ,S+ LBNE LA42D	GET FIRST BYTE  SAVE IT ON THE STACK GET FIRST ARGUMENT  SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET  STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF1Ø 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1 CF18 A6 EØ CF1A 1Ø 26 D5 ØF	LCFØ7  * GET R	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F	LCFØ7	JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECUP TFR D,X LDA ,S+ LBNE LA42D  ECORD BYTE(S) JSR LC597	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF1Ø 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1 CF18 A6 EØ CF1A 1Ø 26 D5 ØF	LCFØ7  * GET R	JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECJP TFR D,X LDA ,S+ LBNE LA42D ECORD BYTE(S) JSR LC597 LDB CINBFL	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70	LCFØ7  * GET R	JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECUP TFR D,X LDA ,S+ LBNE LA42D  ECORD BYTE(S) JSR LC597	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 ØF  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84	LCFØ7  * GET R	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2080 2081 2082 2083 2084 2085 2085	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1 CF18 A6 EØ CF1A 1Ø 26 D5 ØF  CF1E BD C5 97 CF21 D6 7Ø CF23 27 Ø3 CF25 7E C3 34 CF28 A7 84 CF2A A1 8Ø	* GET R LCF1E	JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECUP TFR D,X LDA ,S+ LBNE LA42D ECORD BYTE(S) JSR LC597 LDB CINBFL BEQ LCF28 JMP LC334 STA ,X CMPA ,X+	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY **TEST TO SEE IF IT STORED PROPERLY AND
2068 2069 2070 2071 2072 2073 2074 2075 2077 2078 2077 2080 2081 2080 2081 2082 2083 2084 2085 2086 2087	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1 CF18 A6 EØ CF1A 1Ø 26 D5 ØF  CF1E BD C5 97 CF21 D6 7Ø CF23 27 Ø3 CF25 7E C3 34 CF28 A7 84 CF2A A1 8Ø CF2C 27 Ø3	* GET R LCF1E	JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECUP TFR D,X LDA ,S+ LBNE LA42D ECORD BYTE(S) JSR LC597 LDB CINSFL BEQ LCF28 JMP LC334 STA ,X CMPA ,X+ BEQ LCF31	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PUPPER STORE (NOT IN ROM OR BAD RAM)
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2080 2081 2082 2083 2084 2085 2086 2085 2086	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 7F 03 CF2C 7F D6 16	* GET R LCF1E LCF28	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY **TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF BRORE STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1 CF18 A6 EØ CF1A 1Ø 26 D5 ØF  CF1E BD C5 97 CF21 D6 70 CF23 27 Ø3 CF25 7E C3 34 CF28 A7 84 CF2A A1 8Ø CF2C 27 Ø3 CF2E 7E D6 16 CF31 31 3F	* GET R LCF1E	JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECUP TFR D,X LDA ,S+ LBNE LA42D ECORD BYTE(S) JSR LC597 LDB CINBFL BEQ LCF28 JMP LC334 STA ,X CMPA ,X+ BEQ LCF31 JMP LD616 LEAY -1,Y	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2080 2081 2082 2083 2084 2085 2086 2085 2086	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 7F 03 CF2C 7F D6 16	* GET R LCF1E LCF28	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY **TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF BRORE STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2090 2090 2090 2090 2090	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 27 03 CF2C 32 65 CF31 31 3F CF33 26 E9 CF35 20 D0	* GET R LCF1E LCF28 LCF31	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE OUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2080 2081 2082 2083 2084 2085 2086 2087 2088 2089 2090 2091 2091 2092 2093	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1 CF18 A6 EØ CF1A 1Ø 26 D5 ØF  CF1E BD C5 97 CF21 D6 7Ø CF23 27 Ø3 CF25 7E C3 34 CF28 A7 84 CF2A A1 8Ø CF2C 27 Ø3 CF2C 7E 06 16 CF31 31 3F CF33 26 E9 CF35 2Ø DØ  CF37 8D ØØ	* GET R LCF1E LCF28 LCF31 * READ CG37	JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECJP TFR D,X LDA ,S+ LBNE LA42D ECORD BYTE(S) JSR LC597 LDB CINSFL BEQ LCF28 JMP LC334 STA ,X CMPA ,X+ BEQ LCF31 JMP LD616 LEAY -1,Y BNE LCF97 BRA LCF97	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE SUFFER - RETURN THEM IN ACCD READ A BYTE, SAVE IT IN ACCB
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2078 2080 2081 2082 2083 2084 2085 2096 2087 2088 2099 20901 2091 2092 2093	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF1U 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 7E D6 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 BD 000 CF39 BD CC E2	* GET R LCF1E LCF28 LCF31	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET STATUS OF CONSOLE IN BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE  SUFFER - RETURN THEM IN ACCD GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2080 2081 2082 2083 2084 2085 2086 2087 2089 2090 2091 2092 2093 2093 2095	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1 CF18 A6 EØ CF1A 10 26 D5 ØF  CF1E BD C5 97 CF21 D6 70 CF23 27 Ø3 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 Ø3 CF2C 27 Ø3 CF2C 7E 06 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D ØØ CF37 8D ØØ CF37 8D ØØ CF38 D ØØ CF39 BD CC E2 CF3C 1E 89	* GET R LCF1E LCF28 LCF31 * READ CG37	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE SUFFER - RETURN THEM IN ACCD READ A BYTE, SAVE IT IN ACCB
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2078 2080 2081 2082 2083 2084 2085 2096 2087 2088 2099 20901 2091 2092 2093	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF1U 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 7E D6 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 BD 000 CF39 BD CC E2	* GET R LCF1E LCF28 LCF31 * READ CG37	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET STATUS OF CONSOLE IN BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE  SUFFER - RETURN THEM IN ACCD GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2080 2081 2082 2083 2084 2085 2086 2087 2088 2090 2091 2091 2092 2093 2094 2094 2095	CFØA 34 Ø2 CFØC 8D 29 CFØE 1F Ø2 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F Ø1 CF18 A6 EØ CF1A 10 26 D5 ØF  CF1E BD C5 97 CF21 D6 70 CF23 27 Ø3 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 Ø3 CF2C 27 Ø3 CF2C 7E 06 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D ØØ CF37 8D ØØ CF37 8D ØØ CF38 D ØØ CF39 BD CC E2 CF3C 1E 89	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET STATUS OF CONSOLE IN BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE  SUFFER - RETURN THEM IN ACCD GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2078 2081 2082 2083 2084 2085 2086 2091 2092 2093 2094 2095 2096 2097 2098	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 7C 03 CF2C 7E 06 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D 00 CF39 BD CC E2 CF3C 1E 89 CF3E 39  CF3F 9E A6	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE  SUFFER - RETURN THEM IN ACCD READ A BYTE, SAVE IT IN ACCB GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2078 2080 2081 2082 2083 2084 2089 2090 2091 2092 2093 2091 2092 2093 2094 2095 2096 2097 2098	CF0A 34 02 CF0E 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF2A A1 80 CF2C 27 03 CF2A 78 A7 84 CF2A A1 80 CF2C 27 03 CF2A 78 D6 16 CF31 31 3F CF33 26 E9 CF35 20 D0 CF37 8D 00 CF39 8D CC E2 CF3C 1E 89 CF3E 99 CF3E 90 CF3F 9E A6 CF4I 34 10	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET STATUS OF CONSOLE IN BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) '1/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE SUFFER - RETURN THEM IN ACCD  READ A BYTE, SAVE IT IN ACCB GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2081 2081 2082 2083 2084 2085 2086 2087 2088 2099 2090 2091 2092 2093 2094 2095 2096 2097 2099 2099 2099 2099 2099 2099 2099	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 27 0	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE BUFFER - RETURN THEM IN ACCD  READ A BYTE, SAVE IT IN ACCB GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK GET FILEWAME OF SOURCE FILE
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2078 2080 2081 2082 2083 2084 2085 2086 2091 2092 2093 2094 2097 2099 2100	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 7E 06 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D 00 CF39 BD CC CF3C 1E 89 CF3F 9E A6 CF41 34 10 CF43 8D 35 CF45 96 EB	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE  WEATHER OF THE STACK GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK GET FILENAME OF SOURCE FILE * SAVE DRIVE NUMBER
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2078 2080 2081 2082 2083 2084 2085 2096 2091 2092 2093 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2A 76 06 16 CF31 31 3F CF33 26 E9 CF35 20 D0 CF37 8D 00 CF37 8D 00 CF38 BD CC E2 CF3C 1E 89 CF35 39  CF34 30 35 CF45 96 E8 CF47 34 02	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET STATUS OF CONSOLE IN BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) '1/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE SUFFER - RETURN THEM IN ACCD  READ A BYTE, SAVE IT IN ACCB GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK GET FILENAME OF SOURCE FILE * SAVE DRIVE NUMBER * ON THE STACK
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2078 2080 2081 2082 2083 2084 2085 2086 2091 2092 2093 2094 2097 2099 2100	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 7E 06 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D 00 CF39 BD CC CF3C 1E 89 CF3F 9E A6 CF41 34 10 CF43 8D 35 CF45 96 EB	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE  WEATHER OF THE STACK GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK GET FILENAME OF SOURCE FILE * SAVE DRIVE NUMBER
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2078 2080 2081 2082 2083 2084 2089 2091 2092 2093 2091 2092 2093 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 37 8D 00 CF37 9E A6 CF38 39  CF3F 9E A6 CF41 34 10 CF43 8D 35 CF45 96 EB CF47 34 02 CF49 8D 2A	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET STATUS OF CONSOLE IN BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) '1/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE SUFFER - RETURN THEM IN ACCD READ A BYTE, SAVE IT IN ACCB GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK GET FILENAME OF SOURCE FILE * SAVE DRIVE NUMBER * ON THE STACK SYNTAX CHECK FOR 'TO' AND GET NEW FILENAME GET SOURCE DRIVE NUMBER COMPARE TO NEW FILE DRIVE NUMBER
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2080 2081 2082 2083 2084 2085 2086 2087 2098 2091 2092 2093 2094 2095 2096 2097 2098 2099 2101 2102 2103 2104 2105 2106	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 27 03 CF2C 7E D6 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D 00 CF38 39  CF3F 9E A6 CF3B 39  CF3F 9E A6 CF4T 34 10 CF43 8D 35 CF45 96 EB CF47 34 02 CF49 8D 2A CF48 B5 02 CF49 8D 2A CF48 B5 02 CF49 BD CA CF48 CF47 10 26 E4 F7	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE  SUFFER - RETURN THEM IN ACCD READ A BYTE, SAVE IT IN ACCB GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA,ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK GET FILENAME OF SOURCE FILE * SAVE DRIVE NUMBER * ON THE STACK SYNTAX CHECK FOR 'TO' AND GET NEW FILENAME GET SOURCE DRIVE NUMBER COMPARE TO NEW FILE DRIVE NUMBER 'FC' ERROR IF FILES ON DIFFERENT DRIVES
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2080 2081 2082 2083 2084 2085 2086 2091 2092 2093 2094 2095 2093 2094 2097 2098 2091 2092 2093 2094 2095 2100 2101 2102 2103 2104 2105 2106 2107	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF1U 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 7E 06 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D 00 CF39 BD CC E2 CF3C 1E 89 CF3F 9E A6 CF4I 34 10 CF43 8D 35 CF45 96 EB CF47 34 02 CF4B 35 02 CF4B 36 E8 CF4F 10 26 E4 F7 CF53 8D 28	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR LCCE2 PSHS A BSR LCF37 TFR D,Y BSR LCF37 ADDD VD3 STD EXECJP TFR D,X LDA ,S+ LBNE LA42D ECORD BYTE(S) JSR LCF97 LDB CINBFL BEQ LCF28 JMP LC334 STA ,X CMPA ,X+ BEQ LCF28 JMP LD616 LEAY -1,Y BNE LCF12 BNA LCF97 TWO BYTES FROM E BSR LCF39 JSR LCCE2 EXG A,B RTS E COMMAND LDX CHARAD PSHS X BSR LCF7A LDA DCDRV PSHS A BSR LCF75 PULS A CMPA DCDRV PSHS A CMPA DCDRV PSHS A CMPA DCDRV PSHS A CMPA DCDRV LSTA CF75 PULS A CMPA DCDRV LSTA CF75 LSTA CF75 PULS A CMPA DCDRV LSTA CF75 PUL	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE  SUFFER - RETURN THEM IN ACCD READ A BYTE, SAVE IT IN ACCB GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK GET FILEMAME OF SOURCE FILE * SAVE DRIVE NUMBER * ON THE STACK SYNTAX CHECK FOR 'TO' AND GET NEW FILENAME GET SOURCE DRIVE NUMBER COMPARE TO NEW FILE DRIVE NUMBER VERIFY THAT NEW FILE DRIVE NUMBER
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2090 2091 2092 2093 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 27 03 CF25 7E C5 3 44 CF2A A1 80 CF2C 27 03 CF25 7E C5 20 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D 00 CF37 8D 00 CF37 8D 00 CF37 8D 00 CF38 BD CC E2 CF36 1E 89 CF36 56 EB CF47 34 10 CF43 8D 35 CF45 96 EB CF47 34 02 CF49 8D 2A CF48 35 02 CF49 8D 2A CF48 35 02 CF49 8D 2A CF45 8D 2B CF45 35 10	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE  SUFFER - RETURN THEM IN ACCD READ A BYTE, SAVE IT IN ACCB GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA,ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK GET FILENAME OF SOURCE FILE * SAVE DRIVE NUMBER * ON THE STACK SYNTAX CHECK FOR 'TO' AND GET NEW FILENAME GET SOURCE DRIVE NUMBER COMPARE TO NEW FILE DRIVE NUMBER 'FC' ERROR IF FILES ON DIFFERENT DRIVES
2068 2069 2070 2071 2071 2072 2073 2074 2075 2076 2077 2088 2079 2080 2081 2082 2083 2084 2085 2086 2087 2090 2091 2092 2093 2094 2095 2096 2097 2096 2097 2098 2099 2091 2092 2093 2094 2095 2096 2097 2098 2099 2091 2092 2093 2094 2095 2096 2097 2098 2099 2099 2099 2099 2099 2099 2099	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 27 03 CF2E 7E D6 16 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D 00 CF38 39  CF3F 9E A6 CF41 34 10 CF43 8D 35 CF45 96 EB CF47 34 02 CF49 8D 2A CF48 B5 02 CF49 8D 2A CF48 B5 02 CF49 BC CF47 CF57 9F A6	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) 'I/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE SUFFER - RETURN THEM IN ACCD READ A BYTE, SAVE IT IN ACCB GET A CLARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK SYMTAX CHECK FOR 'TO' AND GET NEW FILENAME GET SOURCE DRIVE NUMBER COMPARE TO NEW FILE DRIVE NUMBER 'FC' ERROR IF FILES ON DIFFERENT DRIVES VERIFY THAT NEW FILE DOES NOT ALREADY EXIST * RESTORE INPUT POINTER  * * RESTORE INPUT POINTER  * * SETORE INPUT POINTER  * ON THE STACK SYMTAX CHECK FOR 'TO' AND GET NEW FILENAME GET SOURCE DRIVE NUMBER 'FC' ERROR IF FILES ON DIFFERENT DRIVES VERIFY THAT NEW FILE DOES NOT ALREADY EXIST  * RESTORE INPUT POINTER
2068 2069 2070 2071 2072 2073 2074 2075 2076 2077 2078 2079 2080 2081 2082 2083 2084 2085 2086 2087 2088 2090 2091 2092 2093 2091 2092 2093 2094 2095 2096 2097 2098 2099 2100 2101 2102 2103 2104 2105	CF0A 34 02 CF0C 8D 29 CF0E 1F 02 CF10 8D 25 CF12 D3 D3 CF14 DD 9D CF16 1F 01 CF18 A6 E0 CF1A 10 26 D5 0F  CF1E BD C5 97 CF21 D6 70 CF23 27 03 CF25 7E C3 34 CF28 A7 84 CF2A A1 80 CF2C 27 03 CF2C 27 03 CF25 7E C5 3 44 CF2A A1 80 CF2C 27 03 CF25 7E C5 20 CF31 31 3F CF33 26 E9 CF35 20 D0  CF37 8D 00 CF37 8D 00 CF37 8D 00 CF37 8D 00 CF38 BD CC E2 CF36 1E 89 CF36 56 EB CF47 34 10 CF43 8D 35 CF45 96 EB CF47 34 02 CF49 8D 2A CF48 35 02 CF49 8D 2A CF48 35 02 CF49 8D 2A CF45 8D 2B CF45 35 10	* GET R LCF1E LCF28 LCF31 * READ LCF37 LCF39	JSR	GET FIRST BYTE SAVE IT ON THE STACK GET FIRST ARGUMENT SAVE IT IN Y GET THE SECOND ARGUMENT ADD IT TO THE OFFSET STORE IT IN THE JUMP ADDRESS OF THE EXEC COMMAND SAVE IT IN X GET THE FIRST BYTE OFF OF THE STACK CLOSE FILE IF POSTAMBLE (EOF)  GET BYTE FROM BUFFER GET STATUS OF CONSOLE IN BUFFER BRANCH IF BUFFER NOT EMPTY 'INPUT PAST END OF FILE' ERROR STORE BYTE IN MEMORY *TEST TO SEE IF IT STORED PROPERLY AND *BRANCH IF PROPER STORE (NOT IN ROM OR BAD RAM) '1/O ERROR' IF BAD STORE DECREMENT BYTE COUNT GET NEXT BYTE IF NOT DONE READ ANOTHER PRE/POST AMBLE SUFFER - RETURN THEM IN ACCD READ A BYTE, SAVE IT IN ACCB GET A CHARACTER FROM INPUT BUFFER, RETURN IT IN ACCA SWAP ACCA, ACCB  * SAVE CURRENT INPUT POINTER * ON THE STACK GET FILENAME OF SOURCE FILE * SAVE DRIVE NUMBER * ON THE STACK SYNTAX CHECK FOR 'TO' AND GET NEW FILENAME GET SOURCE DRIVE NUMBER COMPARE TO NEW FILE DRIVE NUMBER 'FC' ERROR IF FILES ON DIFFERENT DRIVES VERIFY THAT NEW FILE DOES NOT ALREADY EXIST * RESTORE IMPUT POINTER

2113 2114 2115 2116 2117 2118 2119 2120 2121	CF5E BD C6 B8 CF61 8D 12 CF63 8E Ø9 4C CF66 FE Ø9 74 CF69 C6 ØB CF6B BD A5 9A CF6E C6 Ø3 CF7Ø D7 EA CF7Z 7E D5 FF		JSR LC6B8 BSR LCF75 LDX #DNAMBF LDU V974 LDB #11 JSR LA59A LDB #\$03 STB DC0PC JMP LD5FF	'NE' ERROR IF NOT FOUND SYNTAX CHECK FOR 'TO' AND GET NEW FILENAME POINT X TO FILENAME POINT U TO DIRECTORY ENTRY OF SOURCE FILE 11 CHARACTERS IN FILENAME AND EXTENSION COPY NEW FILENAME TO SOURCE FILE DIRECTORY RAM IMAGE * GET WRITE OP CODE AND * SAVE IN DSKCON VARIABLE WRITE NEW DIRECTORY SECTOR
2122 2123 2124 2125 2126 2127 2128 2129 2130 2131 2132	CF75 C6 A5 CF77 BD B2 6F CF7A 7E C8 87 CF7D BD C6 5F CF80 C6 42 CF82 7D 09 73 CF85 10 26 DC BD CF89 39	LCF75 LCF7A LCF7D	VNTAX CHECK FOR 'TO AND STRIP A FILEN. LDB #\$A5 JSR LB26F JMP LC887 JSR LC65F LDB #33*2 TST V973 LBNE LAC46 RTS	AME FROM BASIC 'TO' TOKEN SYNTAX CHECK FOR 'TO' GET FILEMAME FROM BASIC SCAN DIRECTORY FOR FILEMAME 'FILE ALREADY EXISTS' ERROR CHECK FOR A MATCH 'AE' ERROR IF FILE IN DIRECTORY
2133 2134 2135 2136 2137	CF8A 10 27 E9 CA CF8E 8D 03 CF90 0F 6F CF92 39	LCF92	LBEQ LB958 BSR LCF93 CLR DEVNUM RTS	PRINT CARRIAGE RETURN TO CONSOLE OUT IF END OF LINE GO WRITE AN ITEM LIST SET DEVICE NUMBER TO SCREEN
2138 2139 2140 2141 2142 2143 2144	CF93 81 23 CF95 26 ØF CF97 BD A5 A5 CF9A BD A4 Ø6 CF9D 9D A5 CF9F 1Ø 27 E9 B5 CFA3 BD B2 6D	LCFA3	CMPA #'#'  JSR LCFA6  JSR LA5A5  JSR LA496  JSR GETCCH  LBEQ LB958  JSR SYNCOMMA	CHECK FOR DEVICE NUMBER FLAG DEFAULT TO CURRENT DEVICE NUMBER IF NONE GIVEN SET DEVICE NUMBER; CHECK VALIDITY MAKE SURE SELECTED FILE IS AN OUTPUT FILE GET CURRENT INPUT CHARACTER PRINT CR TO CONSOLE OUT IF END OF LINE SYNTAX CHECK FOR COMMA
2145 2146 2147 2148 2149 2150 2151	CFA6 BD B1 56 CFA9 96 06 CFAB 26 1E CFAD BD BD D9 CFB0 BD B5 16 CFB3 BD B9 9F		JSR LB156 LDA VALTYP BNE LCFCB JSR LBDD9 JSR LB516 JSR LB99F	EVALUATE EXPRESSION GET VARIABLE TYPE BRANCH IF STRING CONVERT FP NUMBER TO ASCII STRING PUT ON TEMPORARY STRING STACK PRINT STRING TO CONSOLE OUT
2152 2153 2154 2155 2156 2157 2158 2159	CFB6 9D A5 CFB8 10 27 E9 9C CFBC 86 2C CFBE BD A3 5F CFC1 0D 6E CFC3 27 02 CFC5 86 0D	LCFB6	ITEM SEPARATOR TO CONSOLE OUT JSR GETCCH LBEQ LB958 LDA #',' JSR LA35F TST PRTDEV BEQ LCFC7 LDA #CR	GET CURRENT CHARACTER PUT CR TO CONSOLE OUT IF END OF LINE COMMA: NON-CASSETTE SEPARATOR SET PRINT PARAMETERS * GET CONSOLE PRINT DEVICE AND * BRANCH IF NOT CASSETTE GET CARRIAGE RETURN - CASSETTE ITEM SEPARATOR
2160 2161 2162 2163	CFC7 8D 14 CFC9 20 D8	LCFC7  * PRINT A	BSR LCFDD BRA LCFA3 A STRING TO CONSOLE OUT	SEND SEPARATOR TO CONSOLE OUT GET NEXT ITEM
2164 2165 2166 2167 2168	CFCB 8D 07 CFCD BD B9 9F CFD0 8D 02 CFD2 20 E2		BSR LCFD4 JSR LB99F BSR LCFD4 BRA LCFB6	PRINT LEADING STRING DELIMITER (") PRINT STRING TO CONSOLE OUT PRINT ENDING STRING DELIMITER (") GO PRINT SEPARATOR
2169 2170 2171 2172 2173	CFD4 BD A3 5F CFD7 ØD 6E CFD9 26 B7 CFDB 86 22	LCFD4	STRING DELIMITER (") TO CONSOLE OUT JSR LA35F ST PRIDEV BNE LCF92 LDA #'"'	SET PRINT PARAMETERS  * GET CONSOLE PRINT DEVICE AND  * RETURN IF CASSETTE QUOTE: NON-CASSETTE STRING DELIMITER
2174 2175 2176	CFDD 7E A2 82	* FIELD C		SEND TO CONSOLE OUT
2177 2178 2179 2180	CFEØ BD C7 FE CFE3 4F CFE4 5F CFE5 34 16		JSR LC7FE CLRA CLRB PSHS X,B,A	EVALUATE DEVICE NUMBER & VERIFY RANDOM FILE OPEN  * CLEAR TOTAL FIELD LENGTH COUNTER SAVE FCB POINTER & INITIALIZE TOTAL FIELD LENGTH TO ZERO
2181 2182 2183 2184 2185 2186	CFE7 9D A5 CFE9 26 02 CFEB 35 96 CFED BD B7 38 CFFØ 34 14	LCFED	JSR GETCCH BNE LCFED PULS A,B,X,PC JSR LB738 PSHS X,B	GET CURRENT IMPUT CHARACTER BRANCH IF NOT END OF LINE CLEAN UP STACK AND RETURN SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION *SAVE FIELD LENGTH (ACCB) ON STACK, X IS A DUMMY WHICH WILL *RESERVE 2 BYTES FOR THE ADDRESS WHICH WILL BE CALCULATED BELOW
2187 2188 2189 2190	CFF2 4F	* ,S = FI * 3 4,S =	S POINT THE STACK WILL HAVE THE FOLLOW IELD LENGTH 12,S = RANDOM FILE BUFF! = TOTAL FIELD LENGTH 56,S = FCD POIN CLRA	ER ADDRESS
2191 2192 2193 2194 2195 2196	CFF3 E3 63 CFF5 25 Ø7 CFF7 AE 65 CFF9 1Ø A3 Ø9 CFFC 23 Ø5 CFFE C6 44		ADDD \$03,S BLO LCFFE LDX \$05,S CMPD FCBRLN,X BLS LD003 LDB #34+2	ADD FIELD LENGTH TO TOTAL FIELD LENGTH COUNTER  'FO' ERROR IF SUM > \$FFFF POINT X TO FCB  * COMPARE TO RECORD LENGTH & BRANCH IF *TOTAL FIELD LENGTH < RECORD LENGTH 'FIELD OVERFLOW' ERROR
2197 2198 2199 2200 2201 2202	D000 7E AC 46 D003 EE 63 D005 ED 63 D007 EC 0B D009 33 CB D008 EF 61	LDØØ3	JMP LAC46 LDU \$03,S STD \$03,S LDD FCBBUF,X LEAU D,U STU \$01.S	JUMP TO ERROR DRIVER LOAD U WITH OLD TOTAL LENGTH OF ALL FIELDS SAVE NEW TOTAL FIELD LENGTH POINT ACCD TO START OF RANDOM FILE BUFFER *POINT U TO THIS FIELD'S SLOT IN THE RANDOM *FILE BUFFER AND SAVE IT ON THE STACK
2203 2204 2205 2206 2207 2208	DØØD C6 FF DØØF BD B2 6F DØ12 C6 A7 DØ14 BD B2 6F DØ17 BD B3 57 DØ1A BD B1 46		LDB #\$FF JSR LB26F LDB #\$A7 JSR LB26F JSR LB357 JSR LB146	SECONDARY TOKEN SYNTAX CHECK FOR SECONDARY TOKEN 'AS' TOKEN SYNTAX CHECK FOR 'AS' TOKEN EVALUATE VARIABLE 'TM' ERROR IF NUMERIC VARIABLE

0000	2012 25 11	211.0	A RIVE STRANG ADDRESS AND LENGTH
22Ø9 221Ø	DØ1D 35 44 DØ1F E7 84	PULS B,U STB ,X	* PULL STRING ADDRESS AND LENGTH * OFF OF THE STACK AND SAVE THEM
2211	DØ21 EF Ø2	STU \$02,X	* IN STRING DESCRIPTOR
2212	DØ23 20 C2	BRA LCFE7	CHECK FOR ANOTHER FIELD SPECIFICATION
2213 2214		* RSET COMMAND	
2214	DØ25 86 4F	RSET LDA #\$4F	SKIP ONE BYTE
2216			
2217		* LSET COMMAND	
2218 2219	DØ26 4F DØ27 34 Ø2	LSET CLRA PSHS A	LSET FLAG = Ø SAVE RSET(\$4F),LSET(ØØ) FLAG ON THE STACK
2220	DØ27 34 02 DØ29 BD B3 57	JSR LB357	EVALUATE FIELD STRING VARIABLE
2221	DØ2C BD B1 46	JSR LB146	'TM' ERROR IF NUMERIC VARIABLE
2222	DØ2F 34 1Ø	PSHS X	SAVE STRING DESCRIPTOR ON STACK
2223	DØ31 AE Ø2	LDX \$Ø2,X	POINT X TO ADDRESS OF STRING
2224 2225	DØ33 8C Ø9 89 DØ36 25 Ø5	CMPX #DFLBUF BLO LDØ3D	* COMPARE STRING ADDRESS TO START OF RANDOM * FILE BUFFER: 'SE' ERROR IF < RANDOM FILE BUFFER
2226	DØ38 BC Ø9 4A	CMPX FCBADR	= COMPARE STRING ADDRESS TO TOP OF RANDOM FILE BUFFER
2227	DØ3B 25 Ø5	BLO LDØ42	= AREA - BRANCH IF STRING IN RANDOM FILE BUFFER
2228	DØ3D C6 46	LDØ3D LDB #2*35	'SET TO NON-FIELDED STRING' ERROR
2229 223Ø	DØ3F 7E AC 46 DØ42 C6 B3	JMP LAC46 LDØ42 LDB #\$B3	JUMP TO ERROR HANDLER
2231	DØ44 BD B2 6F	JSR LB26F	* SYNTAX CHECK FOR '=' TOKEN
2232	DØ47 BD 87 48	JSR L8748	=EVALUATE DATA STRING EXPRESSION; RETURN WITH X
2233		*	=POINTING TO STRING; ACCB = LENGTH
2234	DØ4A 35 2Ø	PULS Y	POINT Y TO FIELD STRING DESCRIPTOR
2235 2236	DØ4C A6 A4 DØ4E 27 2E	LDA ,Y BEQ LDØ7E	GET LENGTH OF FIELD STRING RETURN IF NULL STRING
2237	DØ5Ø 34 Ø4	PSHS B	SAVE LENGTH OF DATA STRING ON STACK
2238	DØ52 C6 20	LDB #SPACE	PREPARE TO FILL DATA STRING WITH BLANKS
2239	DØ54 EE 22	LDU \$02,Y	POINT U TO FIELD STRING ADDRESS
224Ø 2241	DØ56 E7 CØ	* FILL THE FIELDED STRING WITH BLANKS LDØ56 STB ,U+	STORE A SPACE IN FIELDED STRING
2242	DØ58 4A	DECA	DECREMENT LENGTH COUNTER
2243	DØ59 26 FB	BNE LDØ56	KEEP FILLING W/SPACES IF NOT DONE
2244	DØ5B E6 EØ	LDB ,S+	*GET THE LENGTH OF THE DATA STRING AND
2245 2246	DØ5D 27 1F DØ5F E1 A4	BEQ LDØ7E CMPB ,Y	*RETURN IF IT IS NULL (ZERO) =COMPARE LENGTH OF DATA STRING TO LENGTH OF FIELD
2247	DØ61 25 Ø4	BLO LDØ67	=STRING, BRANCH IF FIELD STRING > DATA STRING
2248	DØ63 E6 A4	LDB ,Y	*GET THE LENGTH OF THE FIELD STRING AND FORCE THE
2249	DØ65 6F E4	CLR ,S	*RSET/LSET FLAG TO LSET (Ø) IF DATA STRING LENGTH IS
225Ø 2251		*	<pre>*&gt;= THE FIELD STRING LENGTH. THIS WILL CAUSE THE RIGHT *SIDE OF THE DATA STRING TO BE TRUNCATED</pre>
2252	DØ67 EE 22	 LDØ67 LDU \$Ø2,Y	LOAD U WITH THE ADDRESS OF THE FIELD STRING
2253	DØ69 6D EØ	TST ,S+	* GET THE RSET/LSET FLAG FROM THE STACK
2254	DØ6B 27 ØE	BEQ LDØ7B	* AND BRANCH IF LSET
2255	DGCD 24 G4	* RSET ROUTINE	CAVE THE NUMBER OF DATES TO MOVE INTO THE EVELD STRING
2256 2257	DØ6D 34 Ø4 DØ6F 4F	PSHS B CLRA	SAVE THE NUMBER OF BYTES TO MOVE INTO THE FIELD STRING = TAKE THE 2'S COMPLEMENT OF AN UNSIGNED
2258	DØ7Ø 5Ø	NEGB	= NUMBER IN ACCB - LEAVE THE DOUBLE BYTE SIGNED
2259	DØ71 82 ØØ	SBCA #\$ØØ	= RESULT IN ACCD
2260	DØ73 EB A4	ADDB ,Y	* ADD THE LENGTH OF THE FIELD STRING TO THE INVERSE
2261 2262	DØ75 89 ØØ DØ77 33 CB	ADCA #\$ØØ LEAU D,U	* OF THE NUMBER OF BYTES TO BE MOVED =ADD RESULT TO START OF FIELD STRING. NOW U
2263	5577 66 65	22.10 3,0	=WILL POINT TO (-NUMBER OF BYTES TO MOVE)
2264			=FROM THE RIGHT SIDE OF THE FIELD STRING
2265	DØ79 35 Ø4	PULS B	GET THE NUMBER OF BYTES TO MOVE
2266 2267	DØ7B 7E A5 9A DØ7E 35 82	LDØ7B JMP LA59A LDØ7E PULS A,PC	MOVE ACCB BYTES FROM X TO U (DATA TO FIELD STRING) PULL LSET/RSET FLAG OFF OF STACK AND RETURN
2268	5572 00 02	1020 7,10	TOTAL EDETY HOLF TEND OF OTHER MID RETORN
2269		* FILES COMMAND	
2270	DØ8Ø BD 95 AC	FILES JSR L95AC	RESET SAM DISPLAY PAGE AND VDG MODE
2271 2272	DØ83 FC Ø9 4A DØ86 83 Ø9 89	LDD FCBADR SUBD #DFLBUF	GET START OF FILE BUFFERS SUBTRACT THE START OF RANDOM FILE BUFFER SPACE
2273	DØ89 34 Ø6	PSHS B,A	SAVE DEFAULT VALUE OF RANDOM FILE BUFFER SPACE ON STACK
2274	DØ8B F6 Ø9 5B	LDB FCBACT	* GET CURRENT NUMBER OF FCBS
2275	DØ8E 34 Ø4	PSHS B	* AND SAVE ON THE STACK (DEFAULT VALUE)
2276 2277	DØ9Ø 9D A5 DØ92 81 2C	JSR GETCCH CMPA #','	GET CURRENT INPUT CHAR CHECK FOR COMMA
2278	DØ94 27 ØF	BEQ LDØA5	BRANCH IF COMMA - NO BUFFER NUMBER PARAMETER GIVEN
2279	DØ96 BD B7 ØB	JSR EVALEXPB	EVALUATE EXPRESSION (BUFFER NUMBER)
2280	DØ99 C1 ØF	CMPB #15	15 FCBS MAX
2281 2282	DØ9B 1Ø 22 E3 AB DØ9F E7 E4	LBHI LB44A STB ,S	BRANCH IF > 15 - 'ILLEGAL FUNCTION CALL' SAVE NUMBER OF FCBS ON STACK
2283	DØA1 9D A5	JSR GETCCH	CHECK CURRENT INPUT CHAR
2284	DØA3 27 ØB	BEQ LDØBØ	BRANCH IF END OF LINE
2285	DØA5 BD B2 6D	LDØA5 JSR SYNCOMMA	SYNTAX CHECK FOR COMMA
2286 2287	DØA8 BD B3 E6 DØAB C3 ØØ Ø1	JSR LB3E6 ADDD #\$0001	EVALUATE EXPRESSION, RETURN VALUE IN ACCD ADD ONE BYTE
2288	DØAE ED 61	STD \$01,S	SAVE RANDOM FILE BUFFER SIZE ON STACK
2289	DØBØ BD CA 3B	LDØBØ JSR DVEC7	CLOSE FILES
2290	DØB3 E6 E4	LDB ,S	* GET THE NUMBER OF BUFFERS TO MAKE AND
2291 2292	DØB5 34 Ø4 DØB7 CC Ø9 89	PSHS B LDD #DFLBUF	* INITIALIZE A BUFFER COUNTER ON THE STACK GET START OF RANDOM FILE BUFFERS
2292	DØBA E3 62	ADDD \$02,S	ADD THE NEWLY SPECIFIED RANDOM FILE BUFFER SPACE
2294	DØBC 25 5D	BLO LD11B	'OUT OF MEMORY' ERROR IF > \$FFFF
2295	DØBE ED 62	STD \$02,S	SAVE START OF FCBS
2296	חמרמ רי מי יי	* RESERVE SPACE FOR FCBS	ECDIEN DEGITTED FOR EACH DIFFEED
2297 2298	DØCØ C3 Ø1 19 DØC3 25 56	LDØCØ ADDD #FCBLEN BLO LD11B	FCBLEN REQUIRED FOR EACH BUFFER 'OUT OF MEMORY' ERROR IF > \$FFFF
2299	DØC5 6A E4	DEC ,S	DECREMENT BUFFER COUNTER
2300	DØC7 2A F7	BPL LDØCØ	*BRANCH IF NOT DONE - THE BPL WILL SET UP ONE MORE BUFFER
2301		*	*THAN THE NUMBER REQUESTED. THIS EXTRA BUFFER IS THE SYSTEM BUFFER
23Ø2 23Ø3		*	*AND IS LOCATED AT THE END OF THE NORMAL FCBS. ONLY SYSTEM ROUTINES *(COPY, BACKUP, MERGE ETC.) MAY ACCESS THIS BUFFER.
2304	DØC9 5D	TSTB	AT AN EXACT 256 BYTE BOUNDARY?

23Ø5						
	DØCA 27 Ø3		BEQ	LDØCF		YES
23Ø6	DØCC 4C		INCA			NO - ADD 256
2307	DØCD 27 4C	LDGCE		LD11B		'OUT OF MEMORY' ERROR IF PAST \$FFFF
23Ø8 23Ø9	DØCF A7 E4 DØD1 DC 1B	LDØCF	STA LDD	,S VARTAB		SAVE MS BYTE OF NEW GRAPHIC RAM START GET START OF VARIABLES
2310	DØD3 9Ø BC			GRPRAM		*SUBTRACT THE OLD GRAPHIC RAM START - ACCD CONTAINS LENGTH
2311	5550 55 50	*	005/	an rour		*OF PROGRAM PLUS RESERVED GRAPHIC RAM
2312	DØD5 AB E4		ADDA	,\$		ADD IN THE AMOUNT OF RAM CALCULATED ABOVE
2313	DØD7 25 42		BLO	LD11B		'OUT OF MEMORY' ERROR IF > \$FFFF
2314	DØD9 1F Ø1		TFR	D,X		SAVE NEW VARTAB IN X
2315	DØDB 4C	*	INCA			*ADD 256 - TO GUARANTEE ENOUGH ROOM SINCE ALL CALCULATIONS USE
2316 2317	DØDC 27 3D	*	DEO	LD11B		*ONLY THE MSB OF THE ADDRESS 'OUT OF MEMORY' ERROR IF PAST \$FFFF
2318	DØDE 10 93 21			FRETOP		IS IT GREATER THAN THE START OF STRING SPACE
2319	DØE1 24 38			LD11B		'OUT OF MEMORY' IF > START OF STRING SPACE
2320	DØE3 4A		DECA			SUBTRACT 256 - COMPENSATE FOR INCA ABOVE
2321	DØE4 93 1B			VARTAB		SUBTRACT START OF VARIABLES
2322	DØE6 D3 19			TXTTAB		ADD START OF BASIC
2323 2324	DØE8 1F Ø2 DØEA A6 E4		TFR LDA			Y HAS NEW START OF BASIC * GET THE GRAPHIC RAM START, SUBTRACT
2325	DØEC 9Ø BC			GRPRAM		* THE OLD GRAPHIC RAN START, SUBTRACT
2326	DØEE 1F 89		TFR			* THE DIFFERENCE IN ACCA AND ACCB
2327	DØFØ 9B BA			BÉGGRP		= ADD THE OLD GRAPHIC PAGE START AND
2328	DØF2 97 BA			BEGGRP		= STORE THE NEW START OF GRAPHICS RAM
2329	DØF4 DB B7			ENDGRP		* ADD THE OLD GRAPHIC RAM END ADDRESS AND
2330	DØF6 D7 B7			ENDGRP		* STORE THE NEW END OF GRAPHICS RAM
2331 2332	DØF8 35 46	*	PULS	A,B,U		= ACCA=MSB OF START OF GRAPHIC RAM; ACCB=NUMBER OF FILE BUFFERS = U=START OF FILE BUFFERS
2332	DØFA 97 BC		STA	GRPRAM		SAVE NEW START OF GRAPHIC RAM
2334	DØFC F7 Ø9 5B			FCBACT		NUMBER OF FILE BUFFERS
2335	DØFF FF Ø9 4A		STU	FCBADR		START OF FILE BUFFERS
2336	D102 DE 1B	LD102		VARTAB		POINT U TO OLD START OF VARIABLES
2337	D104 9F 1B			VARTAB		SAVE NEW START OF VARIBLES
2338 2339	D106 11 93 1B D109 22 13			VARTAB LD11E		* COMPARE OLD START OF VARIABLES TO NEW START OF * VARIABLES & BRANCH IF OLD > NEW
2339	5107 LL 13	* MOVF		PROGRAM IF OLD START		
2341	D1ØB A6 C2			,-U		GET A BYTE
2342	D1ØD A7 82		STA	, - X		MOVE 1T
2343	D1ØF 11 93 19			TXTTAB		AT START OF BASIC PROGRAM?
2344	D112 26 F7			LD1ØB		NO
2345 2346	D114 10 9F 19 D117 6F 3F			TXTTAB -1,Y		STORE NEW START OF BASIC PROGRAM RESET START OF PROGRAM FLAG
2347	D119 20 13		BRA	LD12E		CLOSE ALL FILES
2348	D11B 7E AC 44	LD11B		LAC44		'OUT OF MEMORY' ERROR
2349		* MOVE	BASIC	PROGRAM IF OLD START	ADDRESS > NEW	START ADDRESS
2350	D11E DE 19	LD11E	LDU	TXTTAB		POINT U TO OLD START OF BASIC
2351	D120 10 9F 19		STY	TXTTAB		SAVE NEW START OF BASIC
2352 2353	D123 6F 3F D125 A6 CØ	LD125	CLR LDA	-1,Y ,U+		RESET START OF BASIC FLAG GET A BYTE
2354	D127 A7 AØ	LDIZS	STA	, Y+		MOVE IT
2355	D129 10 9C 1B			VARTAB		AT START OF VARIABLES
2356	D12C 26 F7		BNE	LD125		NO MOVE ANOTHER BYTE
						NO - MOVE ANOTHER BYTE
2357				000 400 050410011475		
2358			E ALL F	CBS AND RECALCULATE		ESSES
2358 2359	D12E CE Ø9 28	* CLOSI LD12E	E ALL F LDU	#FCBV1		ESSES POINT U TO FILE BUFFER POINTERS
2358			E ALL F LDU			ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS
2358 2359 236Ø	D12E CE Ø9 28 D131 BE Ø9 4A		E ALL F LDU LDX	#FCBV1	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS
2358 2359 2360 2361 2362 2363	D12E CE Ø9 28 D131 BE Ø9 4A D134 5F D135 AF C1 D137 6F ØØ	LD12E	E ALL F LDU LDX CLRB STX CLR	#FCBV1 FCBADR ,U++ FCBTYP,X	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED
2358 2359 2360 2361 2362 2363 2364	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19	LD12E	E ALL F LDU LDX CLRB STX CLR LEAX	#FCBV1 FCBADR ,U++	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB
2358 2359 2360 2361 2362 2363 2364 2365	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C	LD12E	E ALL F LDU LDX CLRB STX CLR LEAX INCB	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER
2358 2359 2360 2361 2362 2363 2364 2365 2366	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B	LD12E	E ALL F LDU LDX CLRB STX CLR LEAX INCB CMPB	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB
2358 2359 2360 2361 2362 2363 2364 2365	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C	LD12E	E ALL F LDU LDX CLRB STX CLR LEAX INCB CMPB BLS	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2	LD12E	E ALL F LDU LDX CLRB STX CLR LEAX INCB CMPB BLS	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D13D 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB	LD12E LD135 * UNLO/	E ALL F LDU LDX CLRB STX CLR LEAX INCB CMPB BLS JMP	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371	D12E CE Ø9 28 D131 BE Ø9 4A D134 5F D135 AF C1 D137 6F ØØ D139 30 89 Ø1 19 D130 5C D13E F1 Ø9 5B D141 23 F2 D143 7E 96 CB	LD12E	E ALL F LDU LDX CLRB STX CLR LEAX INCB BLS JMP AD COMM BSR	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D138 F1 09 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F	LD12E LD135  * UNLO/ UNLOAD	E ALL F LDU LDX CLRB STX CLR LEAX INCB CMPB BLS JMP AD COMM BSR CLRB	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2369 2370 2371 2372 2373	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB D146 8D 1A D148 5F D149 5C	LD12E LD135 * UNLO/	E ALL F LDU LDX CLRB STX CLR LEAX INCB CMPB BLS JMP AD COMM BSR CLRB INCB	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB AND LD162	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLOSE FILE COUNTER INCREMENT FILE COUNTER
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2370 2371 2372 2373 2374	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D138 F1 09 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F	LD12E LD135  * UNLO/ UNLOAD	E ALL F LDU LDX CLRB STX CLR LEAX INCB BLS JMP AD COMM BSR CLRB INCB JSR BEO	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB AND LD162	FCB START ADDR	ESSES POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER
2358 2359 2360 2361 2362 2363 2364 2365 2367 2368 2369 2371 2372 2373 2374 2375	D12E CE Ø9 28 D131 BE Ø9 4A D134 5F D135 AF C1 D137 6F ØØ D139 3Ø 89 Ø1 19 D130 5C D13E F1 Ø9 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D144 BD C7 19	LD12E LD135  * UNLO/ UNLOAD	E ALL F LDU LDX CLRB STX CLR LEAX INCB BLS JMP AD COMM BSR CLRB INCB JSR BEO	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB AND LD162	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2370 2371 2372 2373 2374 2375 2375 2376	D12E CE Ø9 28 D131 BE Ø9 4A D134 5F D135 AF C1 D137 6F ØØ D139 3Ø 89 Ø1 19 D130 5C D13E F1 Ø9 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 Ø0 D14F A6 Ø1 D151 91 EB	LD12E LD135  * UNLO/ UNLOAD	E ALL F LDU LDX CLRB STX CLR LEAX INCB BLS JMP AD COMM BSR CLRB INCB JSR BEO	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB AND LD162	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER?
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2369 2370 2371 2372 2373 2374 2375 2376 2377 2378	D12E CE Ø9 28 D131 BE Ø9 4A D134 5F D135 AF C1 D137 6F ØØ D139 3Ø 89 Ø1 19 D130 5C D13E F1 Ø9 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D144 BD C7 19 D147 A6 Ø1 D151 91 EB D153 26 Ø7	LD12E LD135  * UNLO/ UNLOAD	E ALL F LDU LDX CLRB STX CLR LEAX INCB BLS JMP AD COMM BSR CLRB INCB JSR BEQ LDA CMPA BNE	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB AND LD162 LC719 LD15C FCBBRV,X DCDRV LD15C	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER ONE START OF THE START
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2378 2371 2372 2373 2374 2375 2376 2377 2377 2378 2377	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 0D D14F A6 01 D151 91 EB D153 26 07 D155 34 04	LD12E LD135  * UNLO/ UNLOAD	E ALL F LDU LDX CLRB STX CLR LEAX INCB BLS JMP AD COMM BSR CLRB INCB JSR EQ LDA CMPA BNE PSHS	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB AND LD162 LC719 LD15C FCBDRV,X DCDRV LD15C B	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF, NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK
2358 2359 2360 2361 2362 2363 2364 2366 2367 2372 2373 2374 2375 2377 2378 2377 2378 2378 2379 2378	D12E CE Ø9 28 D131 BE Ø9 4A D134 5F D135 AF C1 D137 6F ØØ D139 3Ø 89 Ø1 19 D130 5C D13E F1 Ø9 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 ØD D14F A6 Ø1 D151 91 EB D153 26 Ø7 D155 34 Ø4 D157 8D CA 58	LD12E LD135  * UNLO/ UNLOAD	E ALL F LDU LDX CLRB STX CLR LEAX INCB CMPB BLS JMP AD COMM BSR CLRB INCB JSR BEQ CMPA BNE PSHS	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X  FCBACT LD135 L96CB  AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B B B B B B B B B B B B B B B B B B B	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2381	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 0D D14F A6 01 D151 91 EB D153 26 07 D155 34 04	LD12E LD135  * UNLO, UNLOAD LD149	E ALL F LDU LDX CLRB STX CLR LEAX INCB BLS JMP AD COMM BSR CLRB JSR BEQ LDA CMPA BNE PSHS JSR PULS	#FCBV1 FCBADR ,U++ FCBTYP, X FCBLEN, X FCBACT LD135 L96CB AND LD162 LC719 LD15C FCBDRV, X DCDRV LD15C B LCA58 B	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF, NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK
2358 2359 2360 2361 2362 2363 2364 2366 2367 2371 2372 2373 2374 2375 2376 2377 2378 2377 2378 2379 2380 2381 2381 2382	D12E CE Ø9 28 D131 BE Ø9 4A D134 5F D135 AF C1 D137 6F ØØ D139 3Ø 89 Ø1 19 D130 5C D13E F1 Ø9 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 ØD D14F A6 Ø1 D151 91 EB D153 26 Ø7 D155 34 Ø4 D15C F1 Ø9 5B D156 23 E8	LD12E LD135  * UNLO, UNLOAD LD149	E ALL F LDU LDX CLRB CLR CLR LEAX INCB CMPB BLS LDS CMPB BLS LDS CMPB BLS CMPA BNE PSHS JSR PULS CMPA BNE PSHS BLS	#FCBV1 FCBADR ,U++ FCBTYP, X FCBLEN, X FCBACT LD135 L96CB AND LD162 LC719 LD15C FCBDRV, X DCDRV LD15C B LCA58 B	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER OLOSE FCB RESTORE FILE COUNTER
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2370 2371 2372 2373 2374 2375 2376 2377 2378 2378 2379 2381 2382 2382 2383 2383	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 00 D147 A6 01 D151 91 EB D153 26 07 D155 34 04 D157 BD CA 58 D15A 35 04 D15C F1 09 5B D15G F1 09 5B D15C F1 09 5B D16C F1 09 5B D16C F1 09 5B	* UNLO/ UNLOAD LD149	E ALL F LDU LDX CLRB STX CLR LEAX CLR LEAX JNCB BLS JMP AD COMM BSR CLRB BLS JSR BEQ LDA CMPA BEQ PSHS JSR PULS CMPB BLS	#FCBV1 FCBADR ,U++ FCBTYP, X FCBLEN, X  FCBACT LD135 L96CB  AND LD162  LC719 LD15C FCBDRV, X DCDRV LD15C B LCA58 B FCBACT LD149	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO
2358 2358 2360 2361 2362 2363 2364 2365 2366 2367 2371 2372 2373 2374 2375 2376 2377 2378 2378 2379 2380 2380 2381 2382 2383 2383 2384 2383	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D14A BD C7 19 D14D 27 0D D14F A6 01 D151 91 EB D153 26 07 D155 34 04 D157 BD CA 58 D156 1 09 5B D157 23 E8 D161 39	* UNLO/ UNLOAD LD149 LD15C * GET I	E ALL F LDU LDX CLRB STX CLR LEAX STX CLR LEAX INCB CMPB BLS CMPB BLS CLRB BINCB EQ LORD JSR BEQ LORD JSR PULS JSR PULS JSR PULS JSR PULS JSR BEQ LORD JSR BEQ LORD JSR BEQ LORD JSR BER BER ST	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X  FCBACT LD135 L96CB  AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B FCBACT LD149  UMBER FROM BASIC - U	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN
2358 2359 2360 2361 2362 2363 2364 2366 2366 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2380 2380 2381 2382 2383 2384 2385 2385	D12E CE 09 28 D131 BE 09 4A D134 FF D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D148 BD C7 19 D140 27 0D D14F A6 01 D151 91 EB D153 26 07 D155 34 04 D157 BD CA 58 D156 23 E8 D161 39 D162 F6 09 5A	* UNLO/ UNLOAD LD149 LD15C * GET I LD162	E ALL F LDU LDX CLRB STX CLR STX CLR LEAX INCB LMB BLS CMPB BSR CLRB BINCB JSR BEQ LDA BNE CMPA BNE CM	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B LCA58 B FCBACT LD149  UMBER FROM BASIC - U DEFDRV	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER
2358 2359 2360 2361 2362 2363 2364 2366 2366 2370 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2380 2380 2381 2382 2383 2384 2385 2385	D12E CE 09 28 D131 BE 09 4A D134 FF D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D148 BD C7 19 D140 27 0D D14F A6 01 D151 91 EB D153 26 07 D155 34 04 D157 BD CA 58 D156 23 E8 D161 39 D162 F6 09 5A	* UNLO/ UNLOAD LD149 LD15C * GET I LD162	E ALL F LDU LDX CLRB STX CLR STX CLR LEAX INCB LMB BLS CMPB BSR CLRB BINCB JSR BEQ LDA BNE CMPA BNE CM	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X FCBACT LD135 L96CB AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B LCA58 B FCBACT LD149  UMBER FROM BASIC - U DEFDRV	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER GET DEFAULT DRIVE NUMBER GET DEFAULT DRIVE NUMBER
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2370 2371 2372 2373 2374 2375 2376 2377 2378 2378 2379 2380 2381 2382 2383 2384 2383 2384 2385 2386 2387 2388	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 00 D147 A6 01 D151 91 EB D153 26 07 D155 34 04 D157 BD CA 58 D15A 35 04 D15C F1 09 5B D15F 23 EB D16C F1 09 5B D16T 27 09 D162 F6 09 5A D165 90 A5 D167 27 09 D169 R0 R7 0B	* UNLO/ UNLOAD LD149  LD15C  * GET [ LD162	E ALL F LDU LDX CLRB STX CLR LEAX CLR LEAX JMP AD COMM BSR CLRB BLS JMP BLS JMP LDA LDA CMPA BEQ LDA CMPA BEQ LDA CMPA BCR RTS RTS RTS BEQ LDA LDB	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X  FCBACT LD135 L96CB  AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B FCBACT LD149 UMBER FROM BASIC - U DEFDRV GETCCH LD172 EYALFYBR	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2370 2371 2372 2373 2374 2375 2376 2377 2378 2378 2379 2380 2381 2382 2383 2384 2383 2384 2385 2386 2387 2388	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 00 D147 A6 01 D151 91 EB D153 26 07 D155 34 04 D157 BD CA 58 D15A 35 04 D15C F1 09 5B D15F 23 EB D16C F1 09 5B D16T 27 09 D162 F6 09 5A D165 90 A5 D167 27 09 D169 R0 R7 0B	* UNLO/ UNLOAD LD149  LD15C  * GET [ LD162	E ALL F LDU LDX CLRB STX CLR LEAX CLR LEAX JMP AD COMM BSR CLRB BLS JMP BLS JMP LDA LDA CMPA BEQ LDA CMPA BEQ LDA CMPA BCR RTS RTS RTS BEQ LDA LDB	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X  FCBACT LD135 L96CB  AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B FCBACT LD149 UMBER FROM BASIC - U DEFDRV GETCCH LD172 EYALFYBR	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FGB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FGB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FGB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER GET NEXT INPUT CHAR USE DEFAULT DRIVE NUMBER IF NONE GIVEN EVALUATE EXPRESSION 4 DRIVES MAX
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2370 2371 2372 2373 2374 2375 2376 2377 2378 2378 2379 2380 2381 2382 2383 2384 2383 2384 2385 2386 2387 2388	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 00 D147 A6 01 D151 91 EB D153 26 07 D155 34 04 D157 BD CA 58 D15A 35 04 D15C F1 09 5B D15F 23 EB D16C F1 09 5B D16T 27 09 D162 F6 09 5A D165 90 A5 D167 27 09 D169 R0 R7 0B	* UNLO/ UNLOAD LD149  LD15C  * GET [ LD162	E ALL F LDU LDX CLRB STX CLR LEAX CLR LEAX JMP AD COMM BSR CLRB BLS JMP BLS JMP LDA LDA CMPA BEQ LDA CMPA BEQ LDA CMPA BCR RTS RTS RTS BEQ LDA LDB	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X  FCBACT LD135 L96CB  AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B FCBACT LD149 UMBER FROM BASIC - U DEFDRV GETCCH LD172 EYALFYBR	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER GET NEXT INPUT CHAR USE DEFAULT DRIVE NUMBER IF NONE GIVEN EVALUATE EXPRESSION 4 DRIVES MAX 'DEVICE NUMBER ERROR' IF > 3
2358 2359 2360 2361 2362 2363 2364 2366 2367 2369 2371 2372 2373 2374 2375 2376 2377 2378 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2389 2389 2389 2390 2390 2390 2390	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D146 BD C7 19 D140 27 00 D14F A6 01 D151 91 EB D153 26 07 D155 34 04 D15C F1 09 5B D157 23 E8 D161 39  D162 F6 09 5A D169 5D A7 D169 BD B7 0B D166 10 22 D4 AD D172 D7 EB	* UNLO/ UNLOAD LD149  LD15C  * GET [ LD162	E ALL F LDU LDX CLRB LDU LDX CLRB LEAX CLR LEAX INCB BLS CMPB BLS CLRB BEQ LDA BNE PSHS RTS RTS RTS RTS RTS RTYE LDB LST LDB LST LDB LBHI STB STB STB	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X  FCBACT LD135 L96CB  AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B FCBACT LD149 UMBER FROM BASIC - U DEFDRV GETCCH LD172 EYALFYBR	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FGB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FGB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FGB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER GET NEXT INPUT CHAR USE DEFAULT DRIVE NUMBER IF NONE GIVEN EVALUATE EXPRESSION 4 DRIVES MAX
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2370 2371 2372 2373 2374 2375 2376 2377 2378 2378 2379 2381 2382 2383 2384 2385 2386 2387 2388 2389 2381 2382 2383 2384 2385 2386 2387 2388 2389 2389 2381 2382 2383 2384 2385 2386 2387 2388 2389 2381 2382 2383 2384 2385 2386 2387 2388 2389 2381 2382 2383 2384 2385 2386 2387 2388 2389 2381 2382 2383 2384 2385 2386 2387 2387 2388 2389 2381 2382 2383 2384 2385 2386 2387 2387 2388 2389 2381 2386 2387 2388 2389 2381 2386 2387 2388 2389 2381 2381 2382 2383 2384 2385 2386 2387 2388 2389 2381 2386 2387 2388 2388 2389 2381 2386 2387 2388 2388 2389 2389 2381 2386 2387 2388 2388 2389 2389 2381 2386 2387 2388 2388 2389 2389 2389 2389 2389 2389	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 00 D147 A6 01 D151 91 EB D153 26 07 D155 34 04 D157 BD CA 58 D15A 35 04 D15C F1 09 5B D15F 23 EB D16C F1 09 5B D16T 27 09 D162 F6 09 5A D165 90 A5 D167 27 09 D169 R0 R7 0B	* UNLO/ UNLOAD LD149  LD15C  * GET [ LD162	E ALL F LDU LDX CLRB STX CLR LEAX CLR LEAX JMP AD COMM BSR CLRB BLS JMP BLS JMP LDA LDA CMPA BEQ LDA CMPA BEQ LDA CMPA BCR RTS RTS RTS BEQ LDA LDB	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X  FCBACT LD135 L96CB  AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B FCBACT LD149 UMBER FROM BASIC - U DEFDRV GETCCH LD172 EYALFYBR	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER GET NEXT INPUT CHAR USE DEFAULT DRIVE NUMBER IF NONE GIVEN EVALUATE EXPRESSION 4 DRIVES MAX 'DEVICE NUMBER ERROR' IF > 3
2358 2359 2360 2361 2362 2363 2364 2366 2367 2369 2371 2372 2373 2374 2375 2376 2377 2378 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2388 2389 2389 2389 2389 2390 2390 2390 2390	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D146 BD C7 19 D140 27 00 D14F A6 01 D151 91 EB D153 26 07 D155 34 04 D15C F1 09 5B D157 23 E8 D161 39  D162 F6 09 5A D169 5D A7 D169 BD B7 0B D166 10 22 D4 AD D172 D7 EB	* UNLO/ UNLOAD LD149  LD15C  * GET [ LD162	E ALL F LDU LDX CLRB STX CLR LEAX INCB CMPB BLS RT LDB	#FCBV1 FCBADR ,U++ FCBTYP, X FCBLEN, X FCBLEN, X FCBACT LD135 L96CB AND LD162  LC719 LD15C FCBDRV, X DCDRV LD15C B LCA58 B FCBACT LD149  UMBER FROM BASIC - U DEFDRV GETCCH LD172 EVALEXPB #\$03 LA61F DCDRV	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER GET NEXT INPUT CHAR USE DEFAULT DRIVE NUMBER IF NONE GIVEN EVALUATE EXPRESSION 4 DRIVES MAX 'DEVICE NUMBER ERROR' IF > 3
2358 2359 2360 2361 2362 2363 2364 2366 2367 2368 2371 2372 2373 2374 2375 2376 2377 2378 2379 2380 2381 2382 2383 2384 2385 2386 2387 2389 2389 2390 2391 2392 2393 2393 2394 2395	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D146 BD C7 19 D140 27 00 D14F A6 01 D151 91 EB D153 26 07 D155 34 04 D15C F1 09 5B D157 23 E8 D161 39  D162 F6 09 5A D169 5D A7 D169 BD B7 0B D166 10 22 D4 AD D172 D7 EB	* UNLO/ UNLOAD LD149 LD15C * GET I LD162 LD169	E ALL F LDU LDX CLRB LDU LDX CLRB LSTX CLR LEAX INCB CMPB BLS CLRB BSR CLRB BSR CLRB BEQ LDA BNE PSHS STS DRIVE N BEQ LDA	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X  FCBACT LD135 L96CB AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B FCBACT LD149  UMBER FROM BASIC - U DEFORV GETCCH LD172 EVALEXPB #\$03 LA61F DCDRV AND	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER GET NEXT INPUT CHAR USE DEFAULT DRIVE NUMBER IF NONE GIVEN EVALUATE EXPRESSION 4 DRIVES MAX 'DEVICE NUMBER ERROR' IF > 3
2358 2358 2359 2360 2361 2362 2363 2364 2366 2366 2371 2372 2373 2374 2375 2376 2377 2378 2377 2378 2379 2380 2381 2381 2382 2383 2384 2385 2386 2387 2389 2391 2392 2393 2394 2395 2395 2397	D12E CE Ø9 28 D131 BE Ø9 4A D134 5F D135 AF C1 D137 6F ØØ D139 3Ø 89 Ø1 19 D130 5C D13E F1 Ø9 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D14A BD C7 19 D140 27 ØD D14F A6 Ø1 D151 91 EB D153 26 Ø7 D155 34 Ø4 D157 BD CA 58 D15A 35 Ø4 D15C F1 Ø9 5B D15F 23 E8 D161 39 D162 F6 Ø9 5A D165 9D A5 D167 27 Ø9 D169 BD 87 ØB D16C C1 Ø3 D17C C1 C4 D17C C7 C8 D17C C8 D17C C7 C8 D17C C	* UNLO/ UNLOAD LD149 LD15C * GET I LD162 LD169 LD172	E ALL F LDU LDX CLRB STX CLR INCB CMPB BLS GLRB INCB CMPA BNE CLRB BNS CLRB BNS CMPA BNE CMPA BNE CMPA BNE CMPA BNE CMPA BNE GMPA GMPA GMPA GMPA GMPA GMPA GMPA GMPA	#FCBV1 FCBADR ,U++ FCBTYP, X FCBLEN, X FCBACT LD135 L96CB AND LD162  LC719 LD15C FCBDRV, X DCDRV LD15C B LCA58 B FCBACT LD149  UMBER FROM BASIC - U DEFORV GETCCH LD172 EVALEXPB #\$03 LA61F DCDRV  AND LA61F L95AC	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER GET NEXT INPUT CHAR USE DEFAULT DRIVE NUMBER IF NONE GIVEN EVALUATE EXPRESSION 4 DRIVES MAX 'DEVICE NUMBER ERROR' IF > 3 STORE IN DSKCON VARIABLE  DEVICE NUMBER ERROR IF NO DRIVE NUMBERS GIVEN RESET SAM DISPLAY PAGE AND VOG MODE
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2371 2372 2373 2374 2375 2376 2377 2378 2378 2381 2382 2383 2384 2385 2389 2390 2391 2392 2393 2394 2395 2396 2397	D12E CE 09 28 D131 BE 09 4A D134 5F D135 AF C1 D137 6F 00 D139 30 89 01 19 D130 5C D13E F1 09 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D144 BD C7 19 D140 27 0D D147 A6 01 D151 91 EB D153 26 07 D155 34 04 D157 BD CA 58 D156 23 EB D157 23 EB D161 39 D162 F6 09 5A D162 70 09 D162 F6 09 5A D165 9D A5 D167 27 09 D169 BD B7 0B D160 C1 03 D172 D7 EB D174 39 D175 10 27 D4 A6 D179 BD 95 AC D175 10 27 D4 A6 D179 BD 95 AC D175 10 27 D4 A6 D179 BD 95 AC D175 10 27 D4 A6 D179 BD 95 AC	* UNLO/ UNLOAD LD149 LD15C * GET I LD162 LD169 LD172	E ALL F LDU LDX CLRB STX CLR LEAX CLR LEAX JMP LDV LDAX LEAX JMP LDA	#FCBV1 FCBADR ,U++ FCBTYP,X FCBLEN,X  FCBACT LD135 L96CB  AND LD162  LC719 LD15C FCBDRV,X DCDRV LD15C B LCA58 B FCBACT LD149  UMBER FROM BASIC - U DEFORV GETCCH LD172 EVALEXPB #\$03 LA61F DCDRV  AND LA61F L95AC LD169	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER IF NONE GIVEN EVALUATE EXPRESSION 4 DRIVES MAX 'DEVICE NUMBER ERROR' IF > 3 STORE IN DSKCON VARIABLE  DEVICE NUMBER ERROR IF NO DRIVE NUMBERS GIVEN RESET SAM DISPLAY PAGE AND VOG MODE * GET SOURCE DRIVE NUMBER AND SAVE
2358 2359 2360 2361 2362 2363 2364 2365 2366 2367 2368 2371 2372 2373 2374 2375 2376 2377 2378 2378 2381 2382 2383 2384 2385 2389 2390 2391 2392 2393 2394 2395 2396 2397	D12E CE Ø9 28 D131 BE Ø9 4A D134 5F D135 AF C1 D137 6F ØØ D139 3Ø 89 Ø1 19 D130 5C D13E F1 Ø9 5B D141 23 F2 D143 7E 96 CB  D146 8D 1A D148 5F D149 5C D14A BD C7 19 D140 27 ØD D14F A6 Ø1 D151 91 EB D153 26 Ø7 D155 34 Ø4 D157 BD CA 58 D15A 35 Ø4 D15C F1 Ø9 5B D15F 23 E8 D161 39 D162 F6 Ø9 5A D165 9D A5 D167 27 Ø9 D169 BD 87 ØB D16C C1 Ø3 D17C C1 C4 D17C C7 C8 D17C C8 D17C C7 C8 D17C C	* UNLO/ UNLOAD LD149 LD15C * GET I LD162 LD169 LD172	E ALL F LDU LDX CLRB STX CLR LEAX INCB BLS ST LGAX JMP AD COMM BSR CLRB JSR BEQ LDA CMPA BNE LDA LDB BLS STB BLS ST	#FCBV1 FCBADR ,U++ FCBTYP, X FCBLEN, X FCBACT LD135 L96CB AND LD162  LC719 LD15C FCBDRV, X DCDRV LD15C B LCA58 B FCBACT LD149  UMBER FROM BASIC - U DEFORV GETCCH LD172 EVALEXPB #\$03 LA61F DCDRV  AND LA61F L95AC	FCB START ADDR	POINT U TO FILE BUFFER POINTERS POINT U TO FILE BUFFER POINTERS POINT X TO START OF BUFFERS RESET FILE COUNTER STORE FILE ADDRESS IN VECTOR TABLE RESET FILE TYPE TO CLOSED GO TO NEXT FCB INCREMENT FILE COUNTER CLOSE ALL ACTIVE BUFFERS AND SYSTEM FCB BRANCH IF NOT DONE READJUST LINE NUMBERS, ETC.  GET DRIVE NUMBER CLEAR FILE COUNTER INCREMENT FILE COUNTER POINT X TO FCB BRANCH IF FILE NOT OPEN CHECK DRIVE NUMBER DOES IT MATCH THE 'UNLOAD' DRIVE NUMBER? NO MATCH - DO NOT CLOSE THE FILE SAVE FILE COUNTER ON THE STACK CLOSE FCB RESTORE FILE COUNTER CHECKED ALL FILES? NO  DRIVE IF NONE GIVEN GET DEFAULT DRIVE NUMBER GET NEXT INPUT CHAR USE DEFAULT DRIVE NUMBER IF NONE GIVEN EVALUATE EXPRESSION 4 DRIVES MAX 'DEVICE NUMBER ERROR' IF > 3 STORE IN DSKCON VARIABLE  DEVICE NUMBER ERROR IF NO DRIVE NUMBERS GIVEN RESET SAM DISPLAY PAGE AND VOG MODE

2401	D184 27 Ø8		BEQ LD18E	BRANCH IF END OF LINE
	D186 C6 A5		LDB #\$A5	TOKEN FOR 'TO'
	D188 BD B2 6F		JSR LB26F	SYNTAX CHECK FOR 'TO'
	D18B BD D1 69		JSR LD169	GET DESTINATION DRIVE NUMBER
		I D18F	LDS #DBUFØ+255	PUT STACK AT TOP OF DBUFØ
	D192 34 Ø4	20102	PSHS B	SAVE DESTINATION DRIVE NUMBER ON STACK
	D194 BD A5 C7			SYNTAX ERROR IF NOT END OF LINE
	D197 BD CA 3B		JSR DVEC7	CLOSE ALL FILES
	D19A 6F E2		CLR ,-S	CLEAR A TRACK COUNTER ON STACK
	D19C 8E Ø9 88		LDX #DFLBUF-1	POINT X TO TOP OF DISK RAM VARIABLES
2411		LD19F	INC ,S	SYNTAX ERROR IF NOT END OF LINE CLOSE ALL FILES CLEAR A TRACK COUNTER ON STACK POINT X TO TOP OF DISK RAM VARIABLES INCREMENT TRACK COUNTER INCREMENT X BY ONE TRACK COMPARE TO TOP OF NON RESERVED RAN KEEP GOING IF MORE FREE RAM LEFT DECREMENT TRACK COUNTER 'OM' ERROR IF < 1 TRACK OF FREE RAM GET MAXIMUM NUMBER OF TRACKS INITIALIZE REMAINING TRACKS CTR INITIALIZE TRACKS WRITTEN COUNTER TO ZERO
			LEAX SECMAX*SECLEN,X	INCREMENT X BY ONE TRACK
	D1A5 9C 27		CMPX MEMSIZ	COMPARE TO TOP OF NON RESERVED RAN
	D1A7 23 F6		BLS LD19F	KEEP GOING IF MORE FREE RAM LEFT
	D1A9 6A E4		DEC ,S	DECREMENT TRACK COUNTER
2416	D1AB 10 27 DA 95		LBEQ LAC44 LDA #TRKMAX	'OM' ERROR IF < 1 TRACK OF FREE RAM
2417	D1AF 86 23		LDA #TRKMAX	GET MAXIMUM NUMBER OF TRACKS INITIALIZE REMAINING TRACKS CTR
2418	D1B1 5F		CLRB	INITIALIZE TRACKS WRITTEN COUNTER TO ZERO
2419	D1B2 34 Ø6		PSHS B,A	SAVE TRACKS WRITTEN AND REMAINING COUNTERS ON STACK
2420				
2421			S POINT THE STACK HAS THE FO	
2422			RACKS REMAINING COUNTER; 1,S	
2423				N RAM; 3,S = DESTINATION DRIVE NUMBER
2424		* 4,S =	SOURCE DRIVE NUMBER	
	D1B4 73 Ø9 5C		COM DRESFL	SET THE DISK RESET FLAG TO CAUSE A RESET
	D1B7 5F	LD1B7	CLRB	INITIALIZE WRITE TRACK COUNTER TO ZERO
	D1B8 5C	LD1B8	INCB	ADD ONE TO WRITE TRACK COUNTER
	D1B9 6A E4		DEC ,S	* DECREMENT REMAINING TRACKS COUNTER
	D1BB 27 Ø4		BEQ LD1C1	* AND BRANCH IF NO TRACKS LEFT
	D1BD E1 62		CMPB \$02,S	= COMPARE WRITE TRACK COUNTER TO NUMBER OF TRACKS THAT
	D1BF 26 F7	10101	BNE LD1B8	= WILL FIT IN RAM AND BRANCH IF ROOM FOR MORE TRACKS IN RAM
		LD1C1	STB TMPLOC	SAVE THE NUMBER OF TRACKS TO BE TRANSFERRED
	D1C3 E6 64 D1C5 8D 48		LDB \$Ø4,S BSR LD2ØF	GET SOURCE DRIVE NUMBER FILL RAM BUFFER WITH TMPLOC TRACKS OF DATA
	D1C7 86 FF		LDA #\$FF	
	D1C9 BD D2 35		JSR LD235	SET SOURCE/DESTINATION FLAG TO DESTINATION PRINT PROMPT MESSAGE IF NEEDED
	D1CC E6 63		LDB \$03,S	GET DESTINATION DRIVE NUMBER
	D1CE 8D 42		RSD I D212	WRITE TMPLOC TRACKS FROM BUFFER
	D1DØ 6D E4		DSR LD212 TST ,S BEQ LD1EØ CLRA JSR LD235 LDB \$Ø1,S	TEST TRACKS REMAINING FLAG
	D1D2 27 ØC		BEO IDIEØ	BRANCH IF BACKUP DONE
	D1D4 4F		CLRA	SET SOURCE/DESTINATION FLAG TO SOURCE
	D1D5 BD D2 35		JSR LD235	PRINT PROMPT MESSAGE IF NEEDED
	D1D8 E6 61		LDB \$01,S	* GET THE TRACKS WRITTEN COUNTER, ADD THE NUMBER OF
	D1DA DB Ø3		ADDB TMPLOC	* TRACKS MOVED THIS TIME THROUGH LOOP AND
2445	D1DC E7 61		STB \$01,S	* SAVE THE NEW TRACKS WRITTEN COUNTER
2446	D1DE 20 D7		BRA LD1B7	COPY SOME MORE TRACKS
2447				
2448	D1EØ 8D Ø3	LD1EØ	BSR LD1E5	CHECK FOR DOS INITIALIZATION
2449	D1E2 7E AC 73		JMP LAC73	JUMP BACK TO BASIC S MAIN LOOP
2450				
	D1E5 35 40	LD1E5	PULS U	PUT THE RETURN ADDRESS IN U
	D1E7 B6 Ø9 5C		LDA DRESFL	TEST DISK RESET FLAG
	D1EA 27 16		BEQ LD202	DON T RESET THE DOS IF FLAG NOT SET
	D1EC 8E Ø9 28		LDX #FCBV1	POINT X TO TABLE OF FCB ADDRESSES
	D1EF 4F	10150	CLRA	SET FILE COUNTER TO ZERO
	D1FØ 6F 91 D1F2 4C	LD1FØ	CLR [,X++] INCA	MARK FCB AS CLOSED ADD ONE TO FILE COUNTER
	D1F3 B1 Ø9 5B		CMPA FCBACT	COMPARE TO NUMBER OF RESERVED FILES
	D1F6 23 F8		BLS LD1FØ	BRANCH IF ANY FILES NOT SHUT DOWN
	D1F8 9E 19		LDX TXTTAB	LOAD X WITH THE START OF BASIC
	D1FA 6F 1F		CLR -1,X	SET FIRST BYTE OF BASIC PROGRAM TO ZERO
	D1FC BD AD 19		JSR LAD19	GO DO A 'NEW'
	D1FF 7F Ø9 5C		CLR DRESFL	RESET THE DOS RESET FLAG
2464	D202 B6 09 5D	LD2Ø2	LDA DLODFL	* CHECK THE LOAD RESET FLAG AND
	D2Ø5 27 Ø6		BEO LD2ØD	* BRANCH IF NOT SET
2466	D2Ø7 7F Ø9 5D		CLR DLODFL	CLEAR THE LOAD RESET FLAG
2467	D2ØA BD AD 19		JSR LAD19	GO DO A 'NEW'
	D2ØD 6E C4	LD2ØD	JMP ,U	JUMP BACK TO RETURN ADDRESS SAVED IN U ABOVE
2469				
	D2ØF 86 Ø2		LDA #\$Ø2	READ OP CODE
	D211 8C 86 Ø3	LD211	CMPX #\$86Ø3	SKIP TWO BYTES
	D212 86 33	LD212	LDA #\$Ø3	WRITE OP CODE
	D214 DD EA		STD DCOPC	SAVE IN DSKCON VARIABLE
	D216 A6 63		LDA \$03,S	* GET THE NUMBER OF THE TRACK BEING CURRENTLY
24/5	D218 97 EC		STA DCTRK	* WRITTEN AND SAVE IT IN DSKCON VARIABLE
24/6	D218 97 EC D21A 8E Ø9 89 D21D 9F EE		LDX #DFLBUF	= TRACK BUFFER STARTS AT DFLBUF = SAVE IT IN DSKCON VARIABLE
	D21E 06 02		STX DCBPT	
	D21F 96 Ø3 D221 C6 Ø1	I D221	LDA TMPLOC LDB #\$Ø1	GET NUMBER OF TRACKS TO MOVE INITIALIZE SECTOR COUNTER TO ONE
			STB DSEC	SAVE DSKCON SECTOR VARIABLE
	D225 BD D5 FF	LULLS	JSR LD5FF	READ/WRITE A SECTOR
	D228 ØC EE		INC DCBPT	MOVE BUFFER POINTER UP ONE SECTOR (256 BYTES)
2483	D22A 5C		INCB	INCREMENT SECTOR COUNTER
2484	D22B C1 12		CMPB #SECMAX	COMPARE TO MAXIMUM NUMBER OF SECTORS PER TRACK
2485	D22D 23 F4		BLS LD223	BRANCH IF ANY SECTORS LEFT
2486	D22B C1 12 D22D 23 F4 D22F ØC EC		INC DCTRK	INCREMENT TRACK COUNTER VARIABLE TO NEXT TRACK
2487	D231 4A		DECA	DECREMENT TRACKS TO MOVE COUNTER
	D232 26 ED		BNE LD221	READ MORE TRACKS IF ANY LEFT
	D234 39		RTS	
2490				
2491	D235 E6 65	LD235	LDB \$05,S	* GET THE DESTINATION DRIVE NUMBER AND
2492	D237 E1 66		CMPB \$06,S	* COMPARE IT TO THE SOURCE DRIVE NUMBER
2493				
2494	2000 05 05		SOURCE/DESTINATION DISK SWIT	
	D239 26 36	LD239		RETURN IF DRIVE NUMBERS NOT EQUAL
2496	D23B 7F Ø9 85		CLR RDYTMR	RESET THE READY TIMER

```
CLEAR DSKREG - TURN OFF ALL DISK MOTORS CLEAR DSKREG RAM IMAGE
2497
           D23E 7F FF 40
                                                            CIR
                                                                    DSKREG
2498
           D241 7F Ø9 86
                                                            CLR
                                                                    DRGRAM
                                                                                                                        SAVE SOURCE/DESTINATION FLAG ON STACK
CLEAR SCREEN
POINT X TO 'INSERT SOURCE' MESSAGE
13 BYTES IN MESSAGE
2499
          D244 34 Ø2
D246 BD A9 28
                                                            PSHS
                                                                    A
LA928
                                                            JSR
2500
          D249 8E D2 72
D24C C6 ØD
2501
                                                            I DX
                                                                    #LD272
2502
                                                                    #13
                                                            LDB
                                                                                                                        GET SOURCE/DESTINATION FLAG FROM THE STACK
BRANCH IF SOURCE
POINT X TO 'INSERT DESTINATION' MESSAGE
18 BYTES IN MESSAGE
25Ø3
           D24E A6 EØ
                                                            LDA
                                                                    LD257
           D25Ø 27 Ø5
25Ø4
                                                            BE0
2505
           D252 8E D2 7F
                                                                    #LD27F
2506
          D255 C6 12
                                                            I DR
                                                                    #18
           D257 BD B9 A2
                                                                    LB9A2
                                                                                                                         SEND MESSAGE TO CONSOLE OUT
2507
                                               LD257
                                                            JSR
2508
           D25A 8E D2 91
                                                            I DX
                                                                    #LD291
                                                                                                                        POINT X TO 'DISKETTE AND' MESSAGE
                                                                                                                        27 BYTES IN MESSAGE
2509
           D25D C6 1B
                                                                    #27
                                                            LDB
                                                                                                                        SEND MESSAGE TO CONSOLE OUT
* SET UP 'SOUND' PARAMETERS
2510
          D25F BD B9 A2
                                                            JSR
                                                                    LB9A2
           D262 CC 64 Ø5
                                                                    #$6405
                                                            LDD
2511
          D265 97 8C
D267 BD A9 51
                                                                                                                        * FOR A BEEP
JUMP TO 'SOUND' - DO A BEEP
2512
                                                                    SNDTON
2513
                                                            JSR
                                                                    LA951
                                                                                                                        GET A CHARACTER FROM CONSOLE IN
* KEEP LOOKING AT CONSOLE IN UNTIL
* YOU GET A CARRIAGE RETURN
2514
           D26A BD A1 71
                                              LD26A
                                                            JSR
                                                                    LA171
           D26D 81 ØD
2515
                                                            CMPA
                                                                    #CR
2516
           D26F 26 F9
                                                                    LD26A
                                              LD271
2517
           D271 39
                                                            RTS
2518
2519
          D272 49 4E 53 45 52 54 LD272
D278 2Ø 53 4F 55 52 43
                                                            FCC
                                                                    'INSERT SOURCE'
2520
2521
           D27E 45
           D27F 49 4E 53 45 52 54 LD27F
                                                                    'INSERT DESTINATION'
                                                            FCC
2522
2523
          D285 2Ø 44 45 53 54 49
D28B 4E 41 54 49 4F 4E
2524
2525
           D291 20 44 49 53 4B 45
                                              LD291
                                                            FCC
                                                                    ' DISKETTE AND'
           D297 54 54 45 20 41 4E
2526
2527
           D29D 44
2528
           D29E ØD
                                                            FCB
                                                                   CR
           D29F 5Ø 52 45 53 53 2Ø
                                                                    'PRESS 'ENTER''
2529
2530
           D2A5 27 45 4E 54 45 52
2531
2532
2533
                                               * PUSH FILENAME.EXT AND DRIVE NUMBER ONTO THE STACK
                                                           PULS Y
LDB #11
          D2AC 35 20
D2AE C6 0B
                                                                                                                        SAVE RETURN ADDRESS IN Y
11 CHARACTERS IN FILENAME AND EXTENSION
2534
                                              LD2AC
2535
                                                                                                                        POINT X TO TOP OF DISK NAME/EXT BUFFER
* GET A CHARACTER FROM FILENAME.
           D2BØ 8E Ø9 57
2536
                                                                    #DNAMBF+11
                                              LD2B3
                                                                    , - X
A
2537
           D2B3 A6 82
                                                            LDA
2538
           D2B5 34 Ø2
                                                            PSHS
                                                                                                                        * EXT BUFFER AND PUSH IT ONTO THE

* STACK - DECREMENT COUNTER AND
2539
           D2B7 5A
                                                            DECB
           D2B8 26 F9
                                                                    LD2B3
                                                                                                                        * KEEP LOOPING UNTIL DONE
                                                            BNE
                                                                                                                        = GET DRIVE NUMBER AND PUSH
2541
           D2BA 96 EB
                                                            LDA
                                                                    DCDRV
                                                                                                                        = IT ONTO THE STACK
PSEUDO - RETURN TO CALLING ROUTINE
2542
           D2BC 34 Ø2
                                                            PSHS
2543
           D2BE 6E A4
                                                            JMP
2544
2545
                                               * PULL FILENAME.EXT AND DRIVE NUMBER FROM (X) TO RAM
                                                                                                                        U RAM
* GET DRIVE NUMBER AND SAVE
* IT IN DSKCON VARIABLE
11 BYTES IN FILENAME AND EXTENSION
          D2CØ A6 8Ø
D2C2 97 EB
2546
                                              LD2CØ
                                                           LDA
2547
                                                                    DCDRV
2548
           D2C4 C6 ØB
                                                            LDB
                                                                    #11
                                                                    #DNAMBF
                                                                                                                        POINT U TO DISK NAME BUFFER
MOVE FILENANE.EXT FROM (X) TO DNAMBF
2549
           D2C6 CE Ø9 4C
                                                            LDU
2550
           D2C9 7E A5 9A
                                                            JMP
                                                                    LA59A
2552
                                               * COPY
                                              * THE COPY PROCESS IS PERFORMED BY COPYING DATA FROM THE SOURCE FILE
2553
                                                 TO RAM AND THEN COPYING IT TO THE DESTINATION FILE. THE SOURCE AND DESTINATION FILES ARE OPENED AS RANDOM FILES AND BOTH USE THE SYSTEM
2554
2555
2556
                                                 FCB ABOVE THE RESERVED FCBS. ALL OF AVAILABLE FREE RAM ABOVE THE VARIABLES IS USED AS A COPY BUFFER WHICH SPEEDS UP THE COPYING PROCESS
2557
                                                VARIABLES IS USED AS A CUPY BUFFER WHICH SPEEDS UP INE CUPYING PROCESS BUT UNFORTUNATELY THE METHOD USED WILL ALLOW AN ERROR ENCOUNTERED DURING THE COPY PROCESS TO 'HANG' THE SYSTEM. THIS IS CAUSED BY POINTING THE FCB'S RANDOM FILE BUFFER POINTER (FCBBUF,X) TO THE FREE RAM BUFFER. AN ERROR WILL THEN CAUSE THE OPEN FILE TO BE CLOSED WITH FCBBUF,X POINTING TO AN AREA IN RAM WHERE THE RANDOM FILE BUFFER CLOSE ROUTINE (LCAE2) WILL NEVER
2558
2559
2560
2561
2562
                                              * LOOK FOR IT
2563
                                                                                                                         * GET SOURCE FILENAME.EXT & DRIVE NUMBER FROM BASIC
                                                            JSR
2564
           D2CC BD C8 87
                                                                                                                        * AND SAVE THEM ON THE STACK
CLEAR A BYTE ON STACK - SINGLE DISK COPY (SDC) FLAG
2565
           D2CF 8D DB
                                                            BSR
                                                                    LD2AC
           D2D1 6F E2
                                                            CLR
2566
                                                                                                                        GET CURRENT INPUT CHARACTER
BRANCH IF END OF LINE - SINGLE DISK COPY
                                                                    GETCCH
2567
           D2D3 9D A5
                                                            JSR
2568
           D2D5 27 ØA
                                                            BEQ
                                                                    LD2E1
                                                                                                                        BRANCH IF END OF LINE - SINGLE DISK COPY)
TOKEN FOR 'TO'
SYNTAX CHECK FOR 'TO'
GET DESTINATION FILENAME.EXT AND DRIVE NUMBER
SAVE DESTINATION FILENAME.EXT & DRIVE NUMBER ON STACK
SYNTAX ERROR IF MORE CHARACTERS ON LINE
2569
           D2D7 63 E4
                                                                    ,S
#$A5
2570
           D2D9 C6 A5
                                                            LDB
2571
           D2DB BD B2 6F
                                                            JSR
                                                                    LB26F
2572
           D2DE BD C8 87
                                                            JSR
                                                                    LC887
2573
           D2E1 8D C9
                                              LD2F1
                                                            BSR
                                                                    LD2AC
2574
2575
           D2E3 BD A5 C7
                                                                    1 A5C7
                                                            .100
           D2E6 BD CA 3B
                                                            JSR
                                                                    DVEC7
                                                                                                                        CLOSE ALL FILES
2576
                                               * COUNT THE NUMBER OF SECTORS WORTH OF FREE RAM AVAILABLE
2577
2578
          D2E9 6F E2
D2EB 3Ø E8 9C
                                                            CLR ,-S
LEAX -100,S
                                                                                                                        CLEAR A SECTOR COUNTER ON THE STACK
** BUG ** THIS SHOULD BE $100 TO POINT X ONE SECTOR LENGTH BELOW STACK
2579
2580
           D2EE 6C E4
                                              LD2EE
                                                                                                                        INCREMENT SECTOR COUNTER
                                                                    ,S
-SECLEN,X
           D2FØ 3Ø 89 FF ØØ
                                                            LEAX
                                                                                                                        DECREMENT X BY ONE SECTOR
2581
                                                                                                                        COMPARE TO TOP OF ARRAYS
BRANCH IF NOT AT BOTTOM OF FREE RAM
           D2F4 9C 1F
                                                                   ARYEND
2582
                                                            CMPX
2583
           D2F6 24 F6
                                                            RHS
                                                                   ID2FF
           D2F8 6A E4
                                                                                                                        DECREMENT SECTOR COUNTER
2584
                                                            DEC
                                                            IREO ÍAC44
                                                                                                                        'OM' ERROR IF NOT AT LEAST ONE FULL SECTOR OF FREE RAM
POINT X TO START OF SOURCE DATA
2585
           D2FA 10 27 D9 46
           D2FE 3Ø 6E
2586
                                                            LEAX
                                                                    14,5
                                                                                                                        POINT A TO STARK OF SOURCE DATA
PUT SOURCE DATA INTO DNAMBE AND DSKCON
SCAN DIRECTORY FOR A MATCH
'NE' ERROR IF MATCH NOT FOUND
POINT X TO DIRECTORY RAM IMAGE OF FOUND FILE
* GET NUMBER OF BYTES IN LAST SECTOR AND
* SOURCE FILE TYPE AND ASCII FLAG
          D300 8D BE
D302 BD C6 5F
2587
                                                            RSR
                                                                    LD2CØ
LC65F
2588
                                                            JSR
          D3Ø5 BD C6 B8
D3Ø8 BE Ø9 74
                                                                    LC6B8
V974
2589
                                                            JSR
2590
                                                            LDX
           D3ØB EE ØE
                                                                    DIRLST, X
          D3ØD AF ØB
2592
                                                            LDX
                                                                    DIRTYP.X
```

2593	D3ØF 34 5Ø	PSHS U,X	* AND SAVE THEM ON THE STACK
2594	D311 BD C7 6D	JSR LC76D	GET VALID FAT DATA
2595	D314 F6 Ø9 76	LDB V976	GET NUMBER OF FIRST GRANULE IN FILE
2596	D317 BD CC 44	JSR LCC44	* GET THE NUMBER OF GRANULES IN FILE
2597	D31A 34 Ø2	PSHS A	* AND SAVE IT ON THE STACK
2598	D31C 4A	DECA	SUBTRACT OFF THE LAST GRANULE
2599	D31D C4 3F	ANDB #\$3F	* MASK OFF LAST GRANULE FLAG BITS AND SAVE THE
2600	D31F 34 Ø4	PSHS B	* NUMBER OF SECTORS IN LAST GRANULE ON STACK
2601	D321 1F 89 D323 4F	TFR A,B	SAVE THE NUMBER OF GRANULES IN ACCB
26Ø2 26Ø3	D323 4F D324 BD C7 49	CLRA JSR LC749	CLEAR THE MS BYTE OF ACCD MULTIPLY ACCD BY NINE
2604	D324 BD C7 49	ADDB ,S	* ADD THE NUMBER OF SECTORS IN THE LAST
2605	D329 89 ØØ	ADCA #\$ØØ	* GRANULE TO ACCD
2606	D32B 8E ØØ Ø1	LDX #\$0001	INITIALIZE RECORD COUNTER TO ONE
2607	D32E 34 16	PSHS X,B,A	INITIALIZE SECTOR AND RECORD COUNTERS ON THE STACK
2608	5022 0. 10	10110 11,0,11	111111111111111111111111111111111111111
2609		* AT THIS POINT THE (	ONTROL VARIABLES FOR COPY ARE STORED ON THE STACK.
2610			SECTORS COUNTER: 2 3.5 = RECORD COUNTER
2611		* 4,S = NUMBER OF	SECTORS TO BE COPIED. INITIALLY SET TO NUMBER OF
2612		* SECTORS I	N THE LAST GRANULE.
2613		* 5,S = GRAN TEST	FLAG. INITIALLY SET TO NUMBER OF GRANS IN FILE
2614		* 6,S = FILE TYPE	; 7,S = ASCII FLAG; 8 9,S = NUMBER OF BYTES IN LAST SECTOR
2615		* 10,S = NUMBER OF	SECTORS WHICH WILL FIT IN THE CURRENTLY AVAILABLE FREE RAM
2616		* 11-22,S = DESTINAT	ON FILENAME.EXT AND DRIVE NUMBER
2617		* 23,S = SINGLE D1	SK COPY FLAG; 24-35,S = SOURCE FILENAME.EXT AND DRIVE NUMBER
2618	D33Ø 5F	LD33Ø CLRB	SET SECTOR COUNTER TO ZERO
2619	D331 AE E4	LDX ,S	GET THE NUMBER OF SECTORS REMAINING IN THE FILE
2620	D333 27 Ø9	BEQ LD33E	BRANCH IF NO SECTORS LEFT
2621	D335 5C	LD335 INCB	ADD A SECTOR TO TEMPORARY SECTOR COUNTER
2622	D336 3Ø 1F	LEAX -1,X	DECREMENT REMAINING SECTORS COUNTER
2623	D338 27 Ø4	BEQ LD33E	BRANCH IF NO SECTORS LEFT *COMPARE TEMPORARY COUNTER TO NUMBER OF SECTORS WHICH MAY
2624 2625	D33A E1 6A	CMPB 10,S	*COMPARE TEMPORARY COUNTER TO NUMBER OF SECTORS WHICH MAY  *BE STORED IN FREE RAM
2625	D33C 26 F7	BNE LD335	*BE STURED IN FREE RAM  BRANCH IF STILL ROOM FOR MORE SECTORS
2627	D33C 26 F7	LD33E STX ,S	SAVE THE NUMBER OF UNCOPIED SECTORS REMAINING IN THE FILE
2628	D34Ø E7 64	STB \$04,S	SAVE THE NUMBER OF SECTORS TO BE COPIED THIS TIME THROUGH LOOP
2629	D340 E7 04 D342 8D 50	BSR \$D394	'GET' ACCB SECTORS TO RAM BUFFER
2630	D344 86 FF	LDA #\$FF	SET SOURCE/DESTINATION FLAG TO DESTINATION
2631	D346 8D 4Ø	BSR LD388	PRINT PROMPT MESSAGE IF REQUIRED
2632	D348 6D 65	TST \$05,S	* CHECK THE GRAN TEST FLAG. IF ⇔ Ø, IT CONTAINS THE
2633	D34A 27 25	BEQ LD371	* NUMBER OF GRANS IN THE FILE AND THE DESTINATION DISK
2634		*	* MUST BE CHECKED FOR ENOUGH ROOM. IF IT IS = Ø
2635		*	* THEN THE CHECK HAS ALREADY BEEN DONE
2636	D34C 3Ø 6B	LEAX 11,S	POINT TO DESTINATION FILE PARAMETERS
2637	D34E BD D2 CØ	JSR LD2CØ	GET DESTINATION FILE PARAMETERS FROM STACK
2638	D351 BD CF 7D	JSR LCF7D	SCAN DIRECTORY FOR FILE - 'AE' ERROR IF IT EXISTS
2639	D354 BD C7 6D	JSR LC76D	GET VALID FAT DATA
2640			
2641		* MAKE SURE THERE ARE	ENOUGH FREE GRANULES ON THE DESTINATION DISK
0.040			
2642	D357 BD C7 25	JSR LC725	POINT X TO FAT
2643	D35A 3Ø Ø6	JSR LC725 LEAX FATCON	POINT X TO FAT X SKIP PAST THE FAT CONTROL BYTES
2643 2644	D35A 3Ø Ø6 D35C A6 65	JSR LC725 LEAX FATCON LDA \$05,S	POINT X TO FAT X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE
2643 2644 2645	D35A 3Ø Ø6 D35C A6 65 D35E C6 44	JSR LC725 LEAX FATCON LDA \$05,S LDB #GRANM	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM
2643 2644	D35A 3Ø Ø6 D35C A6 65	JSR LC725 LEAX FATCON LDA \$05,S	POINT X TO FAT X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE
2643 2644 2645 2646	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84	JSR LC725 LEAX FATCON LDA \$05,S LDB #GRANM? LD360 COM ,X	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE
2643 2644 2645 2646 2647	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3	JSR LC725 LEAX FATCON LDA \$05,S LDB #GRANM: COM ,X BNE LD367	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM * CHECK TO SEE IF A BRAN IS FREE * AND BRANCH IF IT IS NOT FREE
2643 2644 2645 2646 2647 2648	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3 D364 4A	JSR LC725 LEAX FATCON LDA \$05,S LDB #GRANM: LD360 COM ,X BNE LD367 DECA	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM * CHECK TO SEE IF A BRAN IS FREE * AND BRANCH IF IT IS NOT FREE = DECREMENT COUNTER AND BRANCH IF = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER
2643 2644 2645 2646 2647 2648 2649	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3 D364 4A D365 27 Ø8	JSR LC725 LEAX FATCON LDA \$Ø5,S LDB #GRANM; LD360 COM ,X BNE LD367 DECA BEQ LD36F	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES
2643 2644 2645 2646 2647 2648 2649 265Ø 2651 2652	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D36Ø 26 Ø3 D364 4A D365 27 Ø8 D367 63 8Ø D369 5A D369 5A	JSR LC725 LEAX FATCON 405 496,5 LDB #GRANM: LD360 COM ,X BNE LD367 DECA BEQ LD36F LD367 COM ,X+ DECB BNE LD360	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED
2643 2644 2645 2646 2647 2648 2649 265Ø 2651 2652 2653	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3 D364 4A D365 27 Ø8 D367 63 8Ø D369 5A D36A 26 F4 D36C 7E C7 C8	DSR LC725 LEAX FATCON 495,S LDB #GRANM; LD360 COM ,X BNE LD367 DECA BEQ LD367 LD367 COM ,X+ DECB BNE LD369 JMP LC7C8	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM * CHECK TO SEE IF A BRAN IS FREE * AND BRANCH IF IT IS NOT FREE = DECREMENT COUNTER AND BRANCH IF = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR
2643 2644 2645 2646 2647 2648 2649 2650 2651 2652 2653 2654	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3 D364 4A D365 27 Ø8 D367 63 8Ø D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84	JSR LC725 LEAX FATCON 496,5 LDB #GRANM; LD36Ø COM ,X BNE LD367 DECA BEQ LD36F LD367 COM ,X+ DECB BNE LD368 JMP LC7C8 LD36F COM ,X	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENDUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR RESTORE FAT BYTE
2643 2644 2645 2646 2647 2648 2649 2650 2651 2652 2653 2654 2655	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3 D364 4A D365 27 Ø8 D367 63 8Ø D367 63 8Ø D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B	JSR   LC725	POINT X TO FAT  XX SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE
2643 2644 2645 2646 2647 2648 2650 2651 2652 2653 2654 2655 2656	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3 D364 4A D367 63 8Ø D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4	LD367 COM ,X LD367 COM ,X+ LD367 COM ,X+ LD367 COM ,X LD364 LD365 LD366 LD366 LD367 LD367 LD368 LD386 LD386 LD3 , S	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS
2643 2644 2645 2646 2647 2648 2650 2651 2652 2653 2654 2655 2656 2657	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D	LD360 LD364 LC725 LEAX FATCON  LD360 COM ,X BNE LD367 DECA BEQ LD36F LD367 COM ,X+ DECB BNE LD360 BNE LD360 LD367 COM ,X+ DECB SNE LD360 LD367 COM ,X LD371 BSR LD38E LDX ,S BEQ LD384	POINT X TO FAT  XX SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE
2643 2644 2645 2646 2647 2648 2649 2650 2651 2652 2653 2654 2655 2656 2657 2658	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3 D364 4A D365 27 Ø8 D367 63 8Ø D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D377 EC 62	JSR   LC725	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2654 2655 2656 2657 2658	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D366 7E C7 C8 D36F 63 84 D373 AE E4 D379 EC 62 D379 EB 64	LD367 COM ,X LD367	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  * * GET THE CURRENT RECORD COUNTER, ADD
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2655 2656 2657 2656 2659 2669	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D377 EC 62 D379 EB 64 D378 B9 00	LD360 LD367 LC725 LEAX FATCON  LD360 COM ,X BNE LD367 DECA BEQ LD367 LD367 COM ,X+ DECB BNE LD368 BNE LD368 LD367 COM ,X LD371 BSR LD384 LDD \$02,S ADDB \$04,S ADDB \$04,S	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENDUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED
2643 2644 2645 2646 2647 2659 2650 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2660	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3 D364 4A D365 27 Ø8 D367 63 8Ø D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 ØD D377 EC 62 D379 EB 64 D378 89 ØØ	LD360 COM ,X 905,S LC725 LEAX FATCON	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE AND BRANCH IF IT IS NOT FREE = DECREMENT COUNTER AND BRANCH IF = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER
2643 2644 2645 2646 2647 2648 2650 2651 2652 2653 2654 2655 2656 2657 2658 2659 2661 2661	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D37B 89 00 D37D ED 62 D37F 4F	LD367 COM ,X LD367	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE
2643 2644 2645 2646 2647 2659 2650 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2660	D35A 3Ø Ø6 D35C A6 65 D35E C6 44 D36Ø 63 84 D362 26 Ø3 D364 4A D365 27 Ø8 D367 63 8Ø D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 ØD D377 EC 62 D379 EB 64 D378 89 ØØ	LD360 COM ,X 905,S LC725 LEAX FATCON	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE = DECREMENT COUNTER AND BRANCH IF = THERE ARE ENDUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  * * GET THE CURRENT RECORD COUNTER, ADD * THE NUMBER OF SECTORS (RECORDS) MOVED * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663 2663	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D377 EC 62 D379 EB 64 D37B E9 00 D37F 4F D380 8D 06	JSR   LC725	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE
2643 2644 2645 2646 2647 2658 2659 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D377 EC 62 D379 EB 64 D37B E9 00 D37F 4F D380 8D 06	JSR   LC725	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE = DECREMENT COUNTER AND BRANCH IF = THERE ARE ENDUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  * * GET THE CURRENT RECORD COUNTER, ADD * THE NUMBER OF SECTORS (RECORDS) MOVED * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2654 2655 2656 2657 2658 2659 2661 2662 2663 2664 2665	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D378 EB 64 D378 B9 00 D379 EB 64 D378 B9 00 D377 EC 62 D378 B9 00 D378 ED 62 D378 ED 62 D377 4F D380 8D 06 D382 20 AC	LD367 COM ,X LD367	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED KEEP COPYING SECTORS
2643 2644 2645 2646 2647 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663 2664 2664 2665	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D377 EC 62 D379 EB 64 D37B 89 00 D37D ED 62 D37F 4F D380 8D 06 D382 20 AC D384 32 E8 24	LD360 LD367 LC725 LEAX FATCON  LD360 COM ,X BNE LD367 DECA BEQ LD367 LD367 COM ,X+ DECB BNE LD360 JMP LC7C8 LD367 COM ,X LD371 BSR LD384 LDD \$92,S ADDB \$94,S ADCA #\$00 STD \$02,S CLRA BSR LD388 BRA LD388 BRA LD388 BRA LD388	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM * CHECK TO SEE IF A BRAN IS FREE * AND BRANCH IF IT IS NOT FREE = DECREMENT COUNTER AND BRANCH IF = THERE ARE ENDUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT * * GET THE CURRENT RECORD COUNTER, ADD * THE NUMBER OF SECTORS (RECORDS) MOVED * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK
2643 2644 2645 2646 2647 2651 2652 2653 2654 2655 2656 2661 2662 2663 2664 2665 2666 2667 2666 2667	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D377 EC 62 D379 EB 64 D37B 89 00 D37D ED 62 D37F 4F D380 8D 06 D382 20 AC D384 32 E8 24	LD360 LD367 LC725 LEAX FATCON  LD360 COM ,X BNE LD367 DECA BEQ LD367 LD367 COM ,X+ DECB BNE LD360 JMP LC7C8 LD367 COM ,X LD371 BSR LD384 LDD \$92,S ADDB \$94,S ADCA #\$00 STD \$02,S CLRA BSR LD388 BRA LD388 BRA LD388 BRA LD388	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENDUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DONE *****  *CHECK SINGLE DISK COPY FLAG - IF  SZERO, THEN DON'T
2643 2644 2645 2646 2647 2658 2651 2652 2653 2655 2656 2657 2658 2659 2660 2661 2662 2663 2664 2665 2666 2667 2668 2669 2669 2668 2669 267 2688	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D37B 89 00 D37D ED 62 D37F 4F D380 8D 06 D382 20 AC D384 39 D388 6D E8 19	LD367 COM ,X LD368 LDX ,S BEQ LD386 LDX ,S BEQ LD384 LDD \$92,S ADDB \$94,S ADCA #\$90 STD \$92,S CLRA BSR LD388 BRA LD338 LD338 LD338 LD338 LD338 S94,S ADCA #\$90 STD \$92,S CLRA BSR LD338 LD388 LD338 LD388	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  ***** COPY DONE *****  *CHECK SINGLE DISK COPY FLAG - IF   ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE
2643 2644 2645 2646 2647 2659 2651 2652 2653 2654 2655 2656 2657 2668 2661 2662 2663 2665 2666 2667 2668 2669 2669 2671	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D367 63 80 D367 63 80 D367 76 80 D367 76 80 D367 76 80 D367 76 80 D377 80 18 D373 AE E4 D375 27 00 D377 EC 62 D379 EB 64 D37B 89 00 D37D ED 62 D37F 4F D380 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39	LD360 LD367 LD368 LD368 LD368 LD368 LD368 LD339 LD364 LD368 LD339 LD384 LEAS LD339 LD384 LD389 L	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENDUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DONE *****  *CHECK SINGLE DISK COPY FLAG - IF  SZERO, THEN DON'T
2643 2644 2645 2646 2647 2652 2651 2652 2653 2654 2655 2656 2667 2668 2661 2662 2663 2664 2666 2667 2666 2667 2669 2670 2670 2671 2672	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D37B 89 00 D37D ED 62 D37F 4F D380 8D 06 D382 20 AC D384 39 D388 6D E8 19	LD367 COM ,X LD368 LDX ,S BEQ LD386 LDX ,S BEQ LD384 LDD \$92,S ADDB \$94,S ADCA #\$90 STD \$92,S CLRA BSR LD388 BRA LD338 LD338 LD338 LD338 LD338 S94,S ADCA #\$90 STD \$92,S CLRA BSR LD338 LD388 LD338 LD388	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  ***** COPY DONE *****  *CHECK SINGLE DISK COPY FLAG - IF   ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2654 2655 2656 2657 2658 2661 2662 2663 2661 2662 2663 2664 2665 2667 2668 2667 2668 2667 2668 2667 2672 2673	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D367 63 80 D367 63 80 D367 76 80 D367 76 80 D367 76 80 D367 76 80 D377 80 18 D373 AE E4 D375 27 00 D377 EC 62 D379 EB 64 D37B 89 00 D37D ED 62 D37F 4F D380 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39	LD367 COM ,X LD368 LD367 LD367 COM ,X BEQ LD367 LD367 COM ,X BEQ LD366 LD367 COM ,X LD367 COM ,X LD367 COM ,X LD368 LD368 LD368 LD386 LD336 LD387 LD388 TST 25,S  LD388 TST 25,S  LD388 LD339 LD388 LD388 LD339 LD388 LD339 LD388 LD388 LD339 LD388 LD388 LD388 LD388 LD398 LD388 LD388 LD398 LD388 LD388 LD398 LD388 LD388 LD388 LD398 LD388 LD388 LD398 LD388 LD388 LD	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DONE ****  * CCHECK SINGLE DISK COPY FLAG - IF   ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE  PRINT THE PROMPT MESSAGE
2643 2644 2645 2646 2647 2651 2652 2653 2654 2655 2656 2657 2658 2669 2661 2662 2663 2664 2665 2666 2667 2668 2669 2671 2672 2672 2673	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D367 63 80 D367 63 80 D367 63 84 D367 76 C7 C8 D367 76 C7 C8 D377 8D 18 D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D37B 89 00 D37D ED 62 D37F 4F D380 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 7E D2 39 D388 7E D2 39 D388 7E D2 39	LD368	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DONE ****  **  **CHECK SINGLE DISK COPY FLAG - IF  SERO, THEN DON'T  **PRINT THE PROMPT MESSAGE IF REQUIRED
2643 2644 2645 2646 2647 2652 2651 2652 2653 2654 2655 2656 2657 2658 2669 2661 2662 2663 2664 2666 2667 2666 267 2669 2670 2671 2672 2673 2673 2674 2675	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D377 EC 62 D379 EB 64 D37B 89 00 D37D ED 62 D37F 4F D38B 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 7E D2 39 D388 7E D2 39 D388 7E D2 39	JSR	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE HOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DONE ****  *CHECK SINGLE DISK COPY FLAG - IF   ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE  PRINT THE PROMPT MESSAGE IF REQUIRED
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2654 2655 2656 2657 2668 2661 2662 2663 2664 2665 2667 2668 2667 2668 2667 2671 2673 2674 2673 2674 2673	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D378 89 00 D379 EB 64 D378 89 00 D370 ED 62 D377 4F D380 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 86 FF	LD366	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  ***** COPY DONE *****  * CHECK SINGLE DISK COPY FLAG - IF   ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE PRINT THE PROMPT MESSAGE IF REQUIRED  **OM THE DESTINATION/SOURCE I'PUT' FLAG FOINT X TO DESTINATION FILENAME DATA GO 'PUT' SOME DATA
2643 2644 2645 2646 2647 2652 2653 2654 2655 2656 2657 2658 2669 2661 2662 2663 2664 2665 2666 2667 2671 2672 2674 2675 2674 2675	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D367 63 80 D367 63 80 D367 63 84 D367 7E C7 C8 D378 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D378 89 00 D370 ED 62 D378 4F D380 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 7E D2 39 D388 86 FF	LD36#   LC725   LEAX   FATCON   A96, S   LD8   #GRANM;   LD36#   LD3	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DONE ****  **  **CHECK SINGLE DISK COPY FLAG - IF  SERO, THEN DON'T  *PRINT THE PROMPT MESSAGE IF REQUIRED  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  *PRINT THE PROMPT MESSAGE IF REQUIRED  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  **  **PRINT THE PROMPT MESSAGE IF REQUIRED  **  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  **  **  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  **  **  **  **  **  **  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  **  **  **  **  **  **  **  **  **
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2654 2655 2656 2657 2668 2661 2662 2663 2664 2665 2667 2668 2667 2668 2667 2671 2673 2674 2673 2674 2673	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D378 89 00 D379 EB 64 D378 89 00 D370 ED 62 D377 4F D380 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 86 FF	LD369	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DONE ****  **CHECK SINGLE DISK COPY FLAG - IF   ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE  PRINT THE PROMPT MESSAGE IF REQUIRED  **CHECK SINGLE DISK COPY FLAG - IF PRINT THE PROMPT MESSAGE IF REQUIRED  ****** COPY DONE ****  **CHECK SINGLE DISK COPY FLAG - IF PRINT THE PROMPT MESSAGE IF REQUIRED  ***************  ******************
2643 2644 2645 2646 2647 2652 2651 2652 2653 2654 2655 2656 2667 2668 2661 2662 2663 2664 2666 2667 2666 267 2669 2670 2671 2672 2673 2674 2675 2675 2676 2677 2678	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D377 EC 62 D379 EB 64 D376 E 64 D378 89 00 D370 ED 62 D378 4F D388 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 6F D390 30 6D D392 20 04 D394 4F D394 4F D395 30 E8 1A	LD369	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DONE ****  **  **CHECK SINGLE DISK COPY FLAG - IF  SERO, THEN DON'T  *PRINT THE PROMPT MESSAGE IF REQUIRED  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  *PRINT THE PROMPT MESSAGE IF REQUIRED  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  **  **PRINT THE PROMPT MESSAGE IF REQUIRED  **  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  **  **  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  **  **  **  **  **  **  **  **  **CHECK SINGLE DISK COPY FLAG - IF SERO, THEN DON'T  **  **  **  **  **  **  **  **  **
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2654 2655 2656 2657 2668 2661 2662 2663 2664 2665 2667 2668 2667 2668 2667 2671 2672 2673 2674 2677 2677 2677	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D378 EB 64 D378 89 00 D37D ED 62 D37F 4F D380 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 86 FF  D390 30 6D D392 20 04 D394 4F D395 30 E8 1A D398 97 D8	LD369	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  ***** COPY DONE *****  * CHECK SINGLE DISK COPY FLAG - IF   ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE PRINT THE PROMPT MESSAGE PRINT THE PROMPT MESSAGE IF REQUIRED  **OM THE DESTINATION/SOURCE I'PUT' FLAG O' 'PUT' SOME DATA ZERO IS THE 'GET' 'FLOG POINT X TO THE SOURCE FILENAME DATA SAVE THE 'GET' 'PUT' FLAG POINT X TO THE SOURCE FILENAME DATA SAVE THE 'GET' 'PUT' FLAG
2643 2644 2645 2646 2647 2652 2653 2654 2655 2655 2656 2657 2658 2669 2661 2662 2663 2664 2665 2666 2667 2667 2671 2672 2674 2675 2674 2675 2674 2675 2676 2677 2678 2677 2678 2677 2678 2679 2679 2679 2679 2679 2679 2679 2679	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D367 63 80 D367 63 84 D366 7E C7 C8 D367 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D378 89 00 D37D ED 62 D378 4F D380 8D 06 D382 20 AC D384 32 E8 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 6F D390 30 6D D392 20 04 D394 4F D399 30 6B D394 4F D399 30 6B D392 20 04 D394 4F D395 30 E8 1A D398 97 D8 D398 97 D8 D398 DD D2 C0	LD36#   LC725   LEAX   FATCON   A 96, S   LD36#   LD	POINT X TO FAT  SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DONE ****  **  **CHECK SINGLE DISK COPY FLAG - IF  IMP ZERO, THEN DON'T  **PRINT THE PROMPT MESSAGE IF REQUIRED  **  **CHECK SINGLE DISK COPY FLAG - IF IP IN THE PROMPT MESSAGE IF REQUIRED  **  **  **CHECK SINGLE DISK COPY FLAG - IF IP IN THE PROMPT MESSAGE IF REQUIRED  **  **  **  **CHECK SINGLE DISK COPY FLAG - IF IP IN THE PROMPT MESSAGE IF REQUIRED  **  **  **  **  **  **  **  **  **
2643 2644 2645 2646 2647 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663 2664 2665 2667 2667 2671 2672 2674 2675 2674 2675 2674 2675 2674 2677 2678 2677 2678 2677 2678 2677 2688 2689 2681 2681 2681 2682 2683	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D367 63 80 D367 63 84 D366 7E C7 C8 D366 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D378 89 00 D370 ED 62 D378 4F D380 8D 06 D382 20 AC D384 32 EB 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 6F F D390 30 6D D392 20 04 D394 4F D395 30 EB 1A D398 97 D8 D390 AE 68 D397 BF 09 57 D378 E6 95 57 D342 8E 01 00	LD366	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES  GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE  GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE  PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  ***** COPY DONE ****  * CHECK SINGLE DISK COPY FLAG - IF   * ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE IF REQUIRED  OM THE DESTINATION/SOURCE I'PUT' FLAG  POINT X TO DESTINATION FILENAME DATA  GO 'PUT' SOME DATA  ZERO IS THE 'GET' 'PLAG  POINT X TO DESTINATION FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE THE 'GET' FLAG  POINT X TO THE SOURCE FILENAME DATA  SAVE ONE SECTOR LENGTH IN
2643 2644 2645 2646 2647 2648 2651 2652 2653 2655 2656 2657 2658 2657 2668 2661 2662 2663 2664 2667 2666 267 2670 2671 2672 2673 2674 2675 2676 2677 2678 2679 2679 2679 2679 2679 2679 2679 2679	D35A 30 06 D35C A6 65 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D371 8D 1B D373 AE E4 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D378 EP 06 D382 20 AC D388 3D 06 D382 20 AC D388 3D 06 D382 20 AC D388 6D E8 19 D388 86 FF  D390 30 6D D392 20 04 D394 4F D394 4F D394 4F D394 4F D398 97 D8 D399 97 D8 D399 BP 69 57 D342 8E 01 00 D345 BP 09 7C	LD366	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  ***** COPY DONE ****  * CHECK SINGLE DISK COPY FLAG - IF   ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE IF REQUIRED  OM THE DESTINATION FILENAME DATA GO 'PUT' SOME DATA ZERO IS THE 'GET' FLAG POINT X TO THE SOURCE FILENAME DATA SAVE THE 'GET' /'PUT' FLAG GET FILENAME AND DRIVE DATA FROM THE STACK  * GET ASCII FLAG AND FILE TYPE AND SAVE  * THEM IN THE DISK RAM VARIABLES  = SAVE ONE SECTOR LENGTH IN  = RAM RECORD LENGTH VARIABLE
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2654 2655 2658 2669 2661 2662 2663 2664 2665 2666 2667 2668 2669 2671 2672 2673 2674 2674 2675 2676 2677 2678 2679 2679 2679 2679 2679 2679 2679 2679	D35A 30 06 D35C A6 65 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D378 89 00 D37D ED 62 D378 4F D380 8D 06 D382 20 AC D384 32 EB 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 86 FF  D390 30 6D D392 20 04 D394 4F D395 30 EB 1A D398 97 D8 D398 BD D2 C0 D399 AE 68 D399 AE 68 D399 AE 68 D399 AE 69 D399 AE 68 D399 AE 68 D399 AE 68 D399 AE 69 D399 AE 68 D399 AE 69 D399 AE 68 D399 BF 09 57 D3A2 BE 01 00 D3A5 BF 09 7C D3A8 BF 09 7C D3A8 BF 09 7C D3A8 BF 09 7C D3A8 BF 69 57	LD367	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DOME ****  * CHECK SINGLE DISK COPY FLAG - IF <> ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE IF REQUIRED  OM THE DESTINATION/SOURCE I'PUT' FLAG POINT X TO DESTINATION FILENAME DATA ZERO IS THE 'GET' FLAG POINT X TO THE SOURCE FILENAME DATA SAVE THE 'GET' FLAG POINT X TO THE SOURCE FILENAME DATA SAVE THE 'GET', FUT' FLAG GET FILENAME AND DRIVE DATA FROM THE STACK  * GET ASCII FLAG AND FILE TYPE AND SAVE  * THEM IN THE DISK RAM VARIABLE  SAVE ONE SECTOR LENGTH IN  = RAM RECORD LENGTH VARIABLE RANDOM FILE TYPE FLAG RANCH SARIBBLE RANCH SARIBBLE RANDOM FILE TYPE FLAG RANCH SARIBBLE RANCH SARIBB
2643 2644 2645 2646 2647 2651 2652 2653 2654 2655 2656 2657 2658 2659 2660 2661 2662 2663 2664 2665 2667 2667 2671 2672 2673 2674 2675 2674 2675 2674 2675 2676 2677 2678 2677 2678 2677 2678 2677 2678 2677 2678 2677 2678 2677 2678 2677 2678 2677 2678 2679 2678 2679 2671 2672 2673 2674 2675 2676 2677 2678 2677 2678 2679 2679 2679 2679 2679 2679 2679 2679	D35A 30 06 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D376 63 84 D371 8D 1B D373 AE E4 D377 EC 62 D379 EB 64 D378 EB 64 D388 EB 19 D398 EB 60 D399 D30 EB 1A D399 BT 60 D390 AE 68 D397 EB 68 D397 EB 69 D342 EB 61 00 D345 EF 09 D348 EB 61 D346 EB 7C D348 EB 61 D348 EB 7C D348 EB 61 D348 EB 7C D348 EB 61 D348 EB 61 D358 EB 69 D368 EB 7C D348 EB 61 D348 EB 61 D358 EB 69 D358 EB 69 D368 EB 69 D368 EB 69 D368 EB 69 D368 EB 69 D376 EB 69 D348 EB 68 D349 EB 68 D358 EB 69 D348 EB 69 D348 EB 69 D348 EB 69 D348 EB 68 D358 EB	LOS	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF = THERE ARE ENOUGH FREE GRANULES RESTORE FAT BYTE AND INCREMENT POINTER DECREMENT GRAN COUNTER BRANCH IF ALL GRANS NOT CHECKED 'DISK FULL' ERROR 'DISK FULL' ERROR RESTORE FAT BYTE 'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS EXIT ROUTINE IF NO SECTORS LEFT  *  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK ***** COPY DONE *****  * CHECK SINGLE DISK COPY FLAG - IF
2643 2644 2645 2646 2647 2648 2659 2651 2652 2653 2654 2655 2658 2669 2661 2662 2663 2664 2665 2666 2667 2668 2669 2671 2672 2673 2674 2674 2675 2676 2677 2678 2679 2679 2679 2679 2679 2679 2679 2679	D35A 30 06 D35C A6 65 D35C A6 65 D35E C6 44 D360 63 84 D362 26 03 D364 4A D365 27 08 D367 63 80 D369 5A D36A 26 F4 D36C 7E C7 C8 D36F 63 84 D371 8D 1B D373 AE E4 D375 27 0D D377 EC 62 D379 EB 64 D378 89 00 D37D ED 62 D378 4F D380 8D 06 D382 20 AC D384 32 EB 24 D387 39 D388 6D E8 19 D388 7E D2 39 D388 86 FF  D390 30 6D D392 20 04 D394 4F D395 30 EB 1A D398 97 D8 D398 BD D2 C0 D399 AE 68 D399 AE 68 D399 AE 68 D399 AE 69 D399 AE 68 D399 AE 68 D399 AE 68 D399 AE 69 D399 AE 68 D399 AE 69 D399 AE 68 D399 BF 09 57 D3A2 BE 01 00 D3A5 BF 09 7C D3A8 BF 09 7C D3A8 BF 09 7C D3A8 BF 09 7C D3A8 BF 69 57	LD367	POINT X TO FAT  X SKIP PAST THE FAT CONTROL BYTES GET THE NUMBER OF GRANS IN THE FILE  SET GRAN COUNTER TO MAXIMUM  * CHECK TO SEE IF A BRAN IS FREE  * AND BRANCH IF IT IS NOT FREE  = DECREMENT COUNTER AND BRANCH IF  = THERE ARE ENOUGH FREE GRANULES  RESTORE FAT BYTE AND INCREMENT POINTER  DECREMENT GRAN COUNTER  BRANCH IF ALL GRANS NOT CHECKED  'DISK FULL' ERROR  RESTORE FAT BYTE  'PUT' DATA FROM RAM BUFFER TO DESTINATION FILE GET THE NUMBER OF REMAINING SECTORS  EXIT ROUTINE IF NO SECTORS LEFT  * GET THE CURRENT RECORD COUNTER, ADD  * THE NUMBER OF SECTORS (RECORDS) MOVED  * AND SAVE THE NEW RECORD COUNTER  SET SOURCE/DESTINATION FLAG TO SOURCE PRINT PROMPT MESSAGE IF REQUIRED  KEEP COPYING SECTORS  REMOVE TEMPORARY STORAGE VARIABLES FROM STACK  **** COPY DOME ****  * CHECK SINGLE DISK COPY FLAG - IF <> ZERO, THEN DON'T  *PRINT THE PROMPT MESSAGE IF REQUIRED  OM THE DESTINATION/SOURCE I'PUT' FLAG POINT X TO DESTINATION FILENAME DATA ZERO IS THE 'GET' FLAG POINT X TO THE SOURCE FILENAME DATA SAVE THE 'GET' FLAG POINT X TO THE SOURCE FILENAME DATA SAVE THE 'GET', FUT' FLAG GET FILENAME AND DRIVE DATA FROM THE STACK  * GET ASCII FLAG AND FILE TYPE AND SAVE  * THEM IN THE DISK RAM VARIABLE  SAVE ONE SECTOR LENGTH IN  = RAM RECORD LENGTH VARIABLE RANDOM FILE TYPE FLAG RANCH SARIBBLE RANCH SARIBBLE RANDOM FILE TYPE FLAG RANCH SARIBBLE RANCH SARIBB

0.000				
2689	D3B1 9E F1		LDX FCBTMP	POINT X TO THE 'SYSTEM' FCB
2690	D3B3 CC Ø1 ØØ		LDD #SECLEN	* SET THE NUMBER OF BYTES IN THE LAST SECTOR
2691	D3B6 ED 88 13		STD FCBLST,X	* OF THE FILE EQUAL TO ONE SECTOR LENGTH
2692	D3B9 E6 66		LDB \$06,S	=GET THE NUMBER OF SECTORS TO MOVE AND
2693	D3BB 27 29		BEQ LD3E6	=BRANCH IF NONE LEFT
2694	D3BD D6 D8		LDB VD8	*GRAB THE 'GET'/'PUT' FLAG, 'AND' IT WITH THE
2695	D3BF E4 67		ANDB \$07,S	*GRAN TEST FLAG - BRANCH IF 'GET'ING DATA OR THIS IS
2696	D3C1 27 Ø9		BEQ LD3CC	*NOT THE FIRST TIME THROUGH THE LOOP
2697 2698	D3C3 EC 62 D3C5 EB 66		LDD \$02,S	=GET THE NUMBER OF SECTORS REMAINING TO BE COPIED AND
2699	D3C7 89 ØØ		ADDB \$06,S ADCA #\$00	=ADD THE NUMBER TO BE COPIED THIS TIME THROUGH LOOP
2700	D3C9 BD C2 CC		JSR LC2CC	*'PUT' THE LAST RECORD IN THE FILE TO THE SYSTEM FCB.
2701	D3CC 9E F1	LD3CC	LDX FCBTMP	*THE RECORD NUMBER IS IN ACCD.
2702		*		POINT X TO THE SYSTEM FCB
27Ø3	D3CE EE 64		LDU \$Ø4,S	* GET THE CURRENT RECORD NUMBER
2704	D3DØ EF Ø7		STU FCBREC,X	* AND SAVE IT IN THE FCB
2705	D3D2 E6 66		LDB \$06,S	GET THE NUMBER OF THE RECORD (SECTOR) TO MOVE
27Ø6	D3D4 DE 1F		LDU ARYEND	END OF ARRAYS IS THE START OF THE COPY FREE RAM BUFFER
2707	D3D6 34 44	LD3D6	PSHS U,B	SAVE SECTOR COUNTER AND BUFFER POINTER ON THE STACK
27Ø8	D3D8 9E F1		LDX FCBTMP	POINT X TO SYSTEM FCB
2709	D3DA EF ØB		STU FCBBUF,X	*SET THE RANDOM FILE BUFFER POINTER TO THE 'COPY' RAM BUFFER
2710	D3DC BD C2 DØ	*	JSR LC2DØ	*THIS WILL CAUSE THE SYSTEM TO 'HANG' IF AN ERROR OCCURS DURING COPY.
2711 2712	D3DF 6C 61	•	INC \$Ø1,S	GO 'GET' OR 'PUT' DATA TO THE SYSTEM FCB ADD 256 (ONE SECTOR) TO THE BUFFER POINTER
2712	D3E1 35 44		PULS B,U	GET THE SECTOR COUNTER AND BUFFER POINTER
2714	D3E3 5A		DECB	DECREMENT SECTOR COUNTER
2715	D3E4 26 FØ		BNE LD3D6	BRANCH IF ALL SECTORS NOT DONE
2716	D3E6 9E F1	LD3E6	LDX FCBTMP	POINT X TO SYSTEM FCB
2717	D3E8 CE Ø9 89		LDU #DFLBUF	* RESET THE RANDOM FILE BUFFER POINTER FOR THE SYSTEM
2718	D3EB EF ØB		STU FCBBUF,X	* FCB TO THE BOTTOM OF RANDOM FILE BUFFER AREA
2719	D3ED D6 D8		LDB VD8	=GRAB THE 'GET'/'PUT' FLAG, 'AND' IT WITH THE GRAN
2720	D3EF E4 67		ANDB \$07,S	=TEST FLAG - CLOSE THE FILE IF 'GET'ING DATA AND
2721	D3F1 27 Ø9		BEQ LD3FC	=THIS IS NOT THE FIRST TIME THROUGH THE LOOP
2722	D3F3 6F 67		CLR \$07,S	RESET THE GRAN TEST FLAG IF FIRST TIME THROUGH LOOP
2723	D3F5 EC 6A		LDD 10,S	*GET THE NUMBER OF BYTES IN THE LAST SECTOR,
2724	D3F7 8A 8Ø		ORA #\$8Ø	*'OR' IN THE PRE-SAVED FLAG AND
2725	D3F9 ED 88 13	10050	STD FCBLST,X	*SAVE THE NUMBER OF BYTES IN THE LAST SECTOR IN THE FCB
2726	D3FC 7E CA 58	LD3FC	JMP LCA58	CLOSE THE FILE
2727 2728		* UCA1¢	COMMAND	
2729	D3FF 8D 38	DSKI	BSR LD439	GET THE DRIVE, TRACK AND SECTOR NUMBERS
2730	D4Ø1 8D 2B	DUKI	BSR LD42E	* EVALUATE STRING VARIABLE 1 AND SAVE
2731	D403 34 10		PSHS X	* THE DESCRIPTOR ADDRESS ON THE STACK
2732	D4Ø5 8D 27		BSR LD42E	= EVALUATE STRING VARIABLE 2 AND SAVE
2733	D407 34 10		PSHS X	= THE DESCRIPTOR ADDRESS ON THE STACK
2734	D4Ø9 C6 Ø2		LDB #\$Ø2	DSKCON READ OP CODE
2735	D4ØB BD D4 A1		JSR LD4A1	REAO A SECTOR INTO DBUFØ
2736	D4ØE CE Ø6 8Ø		LDU #DBUFØ+128	POINT U TO TOP HALF OF DBUFØ
2737	D411 35 10		PULS X	GET STRING 2 DESCRIPTOR ADDRESS
2738	D413 8D Ø5		BSR LD41A	PUT STRING 2 INTO STRING SPACE
2739	D415 CE Ø6 ØØ		LDU #DBUFØ	POINT U TO BOTTOM HALF OF DBUFØ
2740	D418 35 10		PULS X	GET STRING 1 DESCRIPTOR ADDRESS
2741	D41A 34 50	LD41A	PSHS U,X	PUT STRING DESCRIPTOR & SOURCE POINTER ON THE STACK *
2742 2743	D41C C6 8Ø		LDB #128	
2744	D41E BD B5 ØF D421 33 84		JSR LB5ØF LEAU ,X	* RESERVE 128 BYTES IN STRING SPACE POINT U TO RESERVED STRING SPACE
2745	D423 35 10		PULS X	GET STRING DESCRIPTOR ADDRESS
2746	D425 E7 84		STB ,X	* SAVE DESCRIPTOR DATA (LENGTH AND ADDRESS)
2747	D427 EF Ø2		STU \$02,X	* OF THE NEW STRING
2748	D429 35 10		PULS X	GET THE SOURCE (DBUFØ) POINTER
2749	D42B 7E A5 9A	LD42B	JMP LA59A	MOVE SECTOR DATA FROM DBUFØ TO STRING SPACE
2750				
2751	D42E BD B2 6D			
2752		LD42E	JSR SYNCOMMA	SYNTAX CHECK FOR A COMMA
	D431 8E B3 57	LD42E	LDX #LB357	POINT X TO EVALUATE VARIABLE ROUTINE
2753	D431 8E B3 57 D434 8D 2F		LDX #LB357 BSR LD465	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE
2753 2754	D431 8E B3 57	LD42E LD436	LDX #LB357	POINT X TO EVALUATE VARIABLE ROUTINE
2753 2754 2755	D431 8E B3 57 D434 8D 2F	LD436	LDX #LB357 BSR LD465 JMP LB146	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE
2753 2754 2755 2756	D431 8E B3 57 D434 8D 2F D436 7E B1 46	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE SECTOR NUMBERS
2753 2754 2755	D431 8E B3 57 D434 8D 2F	LD436	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S JSR EVALEXPB	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE SECTOR NUMBERS EVALUATE EXPRESSION, RETURN VALUE IN ACCB
2753 2754 2755 2756 2757	D431 8E B3 57 D434 8D 2F D436 7E B1 46	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE SECTOR NUMBERS
2753 2754 2755 2756 2757 2758	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$Ø3	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) -
2753 2754 2755 2756 2757 2758 2759	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 IC	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB  * COMPARE TO 3 (HIGHEST DRIVE NUMBER) -  * 'FC' ERROR IF IT S > 3
2753 2754 2755 2756 2757 2758 2759 2760	D431 8E B3 57 D434 8D 2F D436 7E B1 46 D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3 SAVE DRIVE NUMBER ON THE STACK
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER * 'FC' ERROR IF TRACK NUMBER > 34
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER)
2753 2754 2755 2756 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38 D446 D7 ED	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE
2753 2754 2755 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2766	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38 D448 BD B7 38 D448 D7 ED D45Ø 5A	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2768	D431 8E 83 57 D434 8D 2F D436 7E 81 46  D439 BD 87 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD 87 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD 87 38 D44E D7 ED D45Ø 5A D45I C1 11	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146 ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX)
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2768 2768	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38 D446 D7 ED D45Ø 5A D451 C1 11 D453 22 07	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER TOOD BIG
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2768 2769	D431 8E 83 57 D434 8D 2F D436 7E 81 46  D439 BD 87 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD 87 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD 87 38 D445 D7 ED D45Ø 5A D45I C1 11 D453 22 Ø7 D455 35 Ø6	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER (SECMAX)  * GET TRACK AND DRIVE NUMBER TOO BIG  * GET TRACK AND DRIVE NUMBER OFF OF
2753 2754 2755 2756 2757 2758 2760 2761 2762 2763 2764 2765 2766 2767 2768 2768 2769 2770	D431 8E 83 57 D434 8D 2F D436 7E 81 46  D439 BD 87 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 8D 87 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD 87 38 D446 D7 ED D45Ø 5A D45I C1 11 D453 22 Ø7 D455 35 Ø6 D457 97 EC	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  EVALUATE EXPRESSION, RETURN VALUE IN ACCB  * COMPARE TO 3 (HIGHEST DRIVE NUMBER) -  * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER)  * CHECK FOR MAXIMUM TRACK NUMBER  * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER)  SAVE SECTOR NUMBER IN DSKCON VARIABLE  *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST  *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX)  'FC' ERROR IF SECTOR NUMBER TOO BIG  * GET TRACK AND SAVE IN DSKCON  * THE STACK AND SAVE IN DSKCON
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2768 2769	D431 8E 83 57 D434 8D 2F D436 7E 81 46  D439 BD 87 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD 87 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD 87 38 D445 D7 ED D45Ø 5A D45I C1 11 D453 22 Ø7 D455 35 Ø6	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER (SECMAX)  * GET TRACK AND DRIVE NUMBER TOO BIG  * GET TRACK AND DRIVE NUMBER OFF OF
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2768 2769 2770 2771 2772	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38 D446 D7 ED D45Ø 5A D451 C1 11 D453 22 Ø7 D455 35 Ø6 D457 97 EC D459 D7 EB	LD436 * EVALU	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK STB DCDRV	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  EVALUATE EXPRESSION, RETURN VALUE IN ACCB  * COMPARE TO 3 (HIGHEST DRIVE NUMBER) -  * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER)  * CHECK FOR MAXIMUM TRACK NUMBER  * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER)  SAVE SECTOR NUMBER IN DSKCON VARIABLE  *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST  *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX)  'FC' ERROR IF SECTOR NUMBER TOO BIG  * GET TRACK AND SAVE IN DSKCON  * THE STACK AND SAVE IN DSKCON
2753 2754 2755 2756 2757 2758 2759 2761 2762 2763 2764 2765 2766 2767 2768 2770 2771 2771 2772 2773	D431 8E 83 57 D434 8D 2F D436 7E 81 46  D439 8D 87 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 8D 87 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 8D 87 38 D445 5A D451 C1 11 D453 22 Ø7 D455 35 Ø6 D457 97 EC D459 D7 EB D458 39	LD436 * EVALU, LD439	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK STB DCDRV RTS	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FFC' ERROR IF SECTOR NUMBER FOR DIG  * GET TRACK AND DRIVE NUMBER OFF OF  * THE STACK AND SAVE IN DSKCON * VARIABLES
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2778 2779 2771 2772 2773 2774 2775 2776	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38 D456 5A D451 C1 11 D453 22 Ø7 D453 35 Ø6 D457 97 EC D459 D7 EB D459 39 D45C 7E B4 4A D45F BD B2 6D	LD436 * EVALU, LD439	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK STB DCDRV RTS JMP LB44A  JSR SYNCOMMA	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER FOR DIG * GET TRACK AND DRIVE NUMBER FOR DIG * THE STACK AND SAVE IN DSKCON * VARIABLES  JUMP TO 'FC' ERROR  SYNTAX CHECK FOR COMMA
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2778 2771 2772 2773 2774 2775 2776 2777	D431 8E 83 57 D434 8D 2F D436 7E 81 46  D439 BD 87 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 8D 87 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD 87 38 D449 D7 ED D45Ø 5A D45C 11 D453 22 Ø7 D457 35 Ø6 D457 97 EC D459 D7 EB D458 39 D45C 7E 84 4A D45F BD 82 6D D462 8E 81 56	LD436 * EVALU, LD439  LD45C LD45F	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK STB DCDRV RTS JMP LB44A  JSR SYNCOMMA LDX #LB156	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE * USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST * CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER TOO BIG * GET TRACK AND DRIVE NUMBER OFF OF * THE STACK AND DRIVE NUMBER OFF OF * THE STACK AND SAVE IN DSKCON * VARIABLES  JUMP TO 'FC' ERROR  SYNTAX CHECK FOR COMMA POINT X TO 'EVALUATE EXPRESSION' ROUTINE ADDRESS
2753 2754 2755 2756 2757 2758 2759 2761 2762 2763 2764 2765 2766 2767 2768 2779 2771 2772 2773 2774 2775 2776 2777 2778	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38 D44E D7 ED D45Ø 5A D451 C1 11 D453 22 Ø7 D455 35 Ø6 D457 97 EC D459 D7 EB D458 39 D45C 7E B4 4A D45F BD B2 6D D466 BE B1 56 D465 D6 EB	LD436 * EVALU, LD439	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK STB DCDRV RTS JMP LB44A JSR SYNCOMMA LDX #LB156 LDB DCDRV	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER TOO BIG * GET TRACK AND DRIVE NUMBER OFF OF * THE STACK AND DRIVE NUMBER OFF OF * THE STACK AND SAVE IN DSKCON * VARIABLES  JUMP TO 'FC' ERROR  SYNTAX CHECK FOR COMMA POINT X TO 'EVALUATE EXPRESSION' ROUTINE ADDRESS * GET THE DSKCON DRIVE, TRACK AND * VERTABLES
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2778 2779 2771 2772 2773 2774 2775 2776 2777 2778 2777	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38 D455 TED D45Ø 5A D451 C1 11 D453 22 Ø7 D455 35 Ø6 D457 79 EC D459 D7 EB D459 D7 EB D459 B7 EB D450 B8 B8 56 D457 DE EB D466 BB B8 60 D467 DE EC	LD436 * EVALU, LD439  LD45C LD45F	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK STB DCDRV RTS JMP LB44A  JSR SYNCOMMA LDX #LB156 LDB DCDRV LDU DCTRK	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE * USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST * CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER TOO BIG * GET TRACK AND DRIVE NUMBER OFF OF * THE STACK AND DRIVE NUMBER OFF OF * THE STACK AND SAVE IN DSKCON * VARIABLES  JUMP TO 'FC' ERROR  SYNTAX CHECK FOR COMMA POINT X TO 'EVALUATE EXPRESSION' ROUTINE ADDRESS
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2778 2779 2771 2772 2773 2774 2775 2777 2778 2777 2778 2777 2778	D431 8E 83 57 D434 8D 2F D436 7E 81 46  D439 BD 87 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 8D 87 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD 87 38 D446 D7 ED D45Ø 5A D45E 11 D453 22 Ø7 D455 35 Ø6 D457 97 EC D459 D7 EB D458 39 D45C 7E 84 4A D45C 7E 84 94 D462 8E 81 56 D465 D6 EB D467 DE EC D469 34 44	LD436 * EVALU, LD439  LD45C LD45F	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S USR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRXMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK STB DCDRV RTS JMP LB44A  JSR SYNCOMMA LDX #LB156 LDB DCDRV LDU DCTRK PSHS U,B	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * CHECK FOR MAXIMUM TRACK NUMBER * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER)  SAVE SECTOR NUMBER IN DSKCON VARIABLE * USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST * CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER TOO BIG * GET TRACK AND DRIVE NUMBER OFF OF * THE STACK AND DRIVE NUMBER OFF OF * THE STACK AND SAVE IN DSKCON  * VARIABLES  JUMP TO 'FC' ERROR  SYNTAX CHECK FOR COMMA POINT X TO 'EVALUATE EXPRESSION' ROUTINE ADDRESS * GET THE DSKCON DRIVE, TRACK AND * SECTOR VALUES AND SAVE THEM ON THE STACK *
2753 2754 2755 2756 2757 2758 2759 2761 2762 2763 2764 2765 2766 2767 2778 2770 2771 2772 2773 2774 2775 2776 2777 2778 2777 2778 2777 2778 2777 2778 2777 2778 2779 2781	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38 D44E D7 ED D45Ø 5A D451 C1 11 D453 22 Ø7 D455 35 Ø6 D457 97 EC D459 D7 EB D458 39 D45C 7E B4 4A D45F BD B2 6D D462 BE B1 56 D465 D6 EB D467 DE EC D469 34 44 D468 AD 84	LD436 * EVALU, LD439  LD45C LD45F	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK STB DCDRV RTS JMP LB44A  JSR SYNCOMMA LDX #LB156 LDB DCDRV LDU DCTRK PSHS U,B JSR ,X	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER TOO BIG * GET TRACK AND DRIVE NUMBER OFF OF * THE STACK AND DRIVE NUMBER OFF OF * THE STACK AND SAVE IN DSKCON * VARIABLES  JUMP TO 'FC' ERROR  SYNTAX CHECK FOR COMMA POINT X TO 'EVALUATE EXPRESSION' ROUTINE ADDRESS * GET THE DSKCON DRIVE, TRACK AND * SECTOR VALUES AND SAVE THEM ON THE STACK * GO EVALUATE AN EXPRESSION OR A VARIABLE
2753 2754 2755 2756 2757 2758 2759 2760 2761 2762 2763 2764 2765 2766 2767 2778 2771 2772 2773 2774 2775 2776 2777 2778 2777 2778 2779 2780 2780 2780 2780	D431 8E 83 57 D434 8D 2F D436 7E 81 46  D439 BD 87 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 8D 87 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD 87 38 D44E D7 ED D45Ø 5A D45E 11 D453 22 Ø7 D45Ø 5A D45E 7E 84 4A  D45F 8D 82 6D D462 8E 81 56 D465 D6 EB D467 DE EC D469 34 44 D468 AD 84 D468 AD 84 D468 AB 84	LD436 * EVALU, LD439  LD45C LD45F	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB JSR LB738 STB DSEC DECB UECB A,B STA DCTRK STB DCDRV RTS JMP LB44A  JSR SYNCOMMA LDX #LB156 LDB DCDRV LDU DCTRK PSHS U,B JSR ,X PULS B,U	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  EVALUATE EXPRESSION, RETURN VALUE IN ACCB  * COMPARE TO 3 (HIGHEST DRIVE NUMBER) -  * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER)  * CHECK FOR MAXIMUM TRACK NUMBER  * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER)  SAVE SECTOR NUMBER IN DSKCON VARIABLE  *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST  *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER TOO BIG  * GET TRACK AND DRIVE NUMBER OFF OF  * THE STACK AND SAVE IN DSKCON  * VARIABLES  JUMP TO 'FC' ERROR  SYNTAX CHECK FOR COMMA POINT X TO 'EVALUATE EXPRESSION' ROUTINE ADDRESS  * GET THE DSKCON DRIVE, TRACK AND  * SECTOR VALUES AND SAVE THEM ON THE STACK  * GO EVALUATE AN EXPRESSION OR A VARIABLE  * GET THE DRIVE, TRACK AND SAVE THEM ON THE STACK  * GO EVALUATE AN EXPRESSION OR A VARIABLE  * GET THE DRIVE, TRACK AND SECTOR
2753 2754 2755 2756 2757 2758 2759 2761 2762 2763 2764 2765 2766 2767 2778 2770 2771 2772 2773 2774 2775 2776 2777 2778 2777 2778 2777 2778 2777 2778 2777 2778 2779 2781	D431 8E B3 57 D434 8D 2F D436 7E B1 46  D439 BD B7 ØB D43C C1 Ø3 D43E 22 1C D44Ø 34 Ø4 D442 BD B7 38 D445 C1 22 D447 22 13 D449 34 Ø4 D448 BD B7 38 D44E D7 ED D45Ø 5A D451 C1 11 D453 22 Ø7 D455 35 Ø6 D457 97 EC D459 D7 EB D458 39 D45C 7E B4 4A D45F BD B2 6D D462 BE B1 56 D465 D6 EB D467 DE EC D469 34 44 D468 AD 84	LD436 * EVALU, LD439  LD45C LD45F	LDX #LB357 BSR LD465 JMP LB146  ATE DRIVE, TRACK AND S JSR EVALEXPB CMPB #\$03 BHI LD45C PSHS B JSR LB738 CMPB #TRKMAX-1 BHI LD45C PSHS B JSR LB738 STB DSEC DECB CMPB #SECMAX-1 BHI LD45C PULS A,B STA DCTRK STB DCDRV RTS JMP LB44A  JSR SYNCOMMA LDX #LB156 LDB DCDRV LDU DCTRK PSHS U,B JSR ,X	POINT X TO EVALUATE VARIABLE ROUTINE EVALUATE A VARIABLE 'TM' ERROR IF NUMERIC VARIABLE  SECTOR NUMBERS  EVALUATE EXPRESSION, RETURN VALUE IN ACCB * COMPARE TO 3 (HIGHEST DRIVE NUMBER) - * 'FC' ERROR IF IT S > 3  SAVE DRIVE NUMBER ON THE STACK SYNTAX CHECK FOR COMMA. EVALUATE EXPRESSION (TRACK NUMBER) * 'FC' ERROR IF TRACK NUMBER > 34  SAVE TRACK NUMBER ON THE STACK SYNTAX CHECK FOR COMMA, EVALUATE EXPRESSION (SECTOR NUMBER) SAVE SECTOR NUMBER IN DSKCON VARIABLE *USELESS INSTRUCTION. NEXT INSTRUCTION SHOULD JUST *CHECK FOR MAXIMUM SECTOR NUMBER (SECMAX) 'FC' ERROR IF SECTOR NUMBER TOO BIG * GET TRACK AND DRIVE NUMBER OFF OF * THE STACK AND DRIVE NUMBER OFF OF * THE STACK AND SAVE IN DSKCON * VARIABLES  JUMP TO 'FC' ERROR  SYNTAX CHECK FOR COMMA POINT X TO 'EVALUATE EXPRESSION' ROUTINE ADDRESS * GET THE DSKCON DRIVE, TRACK AND * SECTOR VALUES AND SAVE THEM ON THE STACK * GO EVALUATE AN EXPRESSION OR A VARIABLE

2785	D473 39		RTS	
2786	0473 33		KIS	
2787	D474 OD C2		COMMAND	OFT THE DRIVE TRACK AND OFFICED NUMBERS
2788 2789	D474 8D C3 D476 8D E7	DSK0	BSR LD439 BSR LD45F	GET THE DRIVE, TRACK AND SECTOR NUMBERS GET THE DESCRIPTOR OF STRING 1
279Ø	D478 8D BC		BSR LD436	'TM' ERROR IF NUMERIC EXPRESSION
2791	D47A 9E 52		LDX FPAØ+2	* GET STRING 1 DESCRIPTOR ADDRESS
2792 2793	D47C 34 10 D47E 8D DF		PSHS X BSR LD45F	* AND SAVE IT ON THE STACK GET THE DESCRIPTOR OF STRING 2
2793	D48Ø BD B6 54		JSR LB654	*GET LENGTH AND ADDRESS OF STRING 2 AND
2795	D483 34 14		PSHS X,B	*SAVE THEM ON THE STACK
2796	D485 5F		CLRB	SET CLEAR COUNTER TO 256 (FULL SECTOR BUFFER)
2797 2798	D486 8E Ø6 ØØ D489 6F 8Ø	LD489	LDX #DBUFØ CLR ,X+	USE DBUFØ AS THE DSKO\$ I/O BUFFER CLEAR A BYTE IN I/O BUFFER
2799	D48B 5A		DECB	DECREMENT CLEAR COUNTER
2800	D48C 26 FB		BNE LD489	BRANCH IF ALL 256 BYTES NOT CLEARED
28Ø1 28Ø2	D48E 35 14 D490 CE 06 80		PULS B,X LDU #DBUFØ+128	GET THE LENGTH AND ADDRESS OF STRING 2 POINT X TO STRING 2 DESTINATION
28Ø3	D493 8D 96		BSR LD42B	MOVE STRING 2 DATA INTO DBUFØ
2804	D495 35 10		PULS X	POINT X TO STRING 1 DESCRIPTOR
28Ø5 28Ø6	D497 BD B6 59 D49A CE Ø6 ØØ		JSR LB659 LDU #DBUFØ	GET THE LENGTH AND ADDRESS OF STRING 1 POINT U TO STRING 1 DESTINATION
2807	D49D 8D 8C		BSR LD42B	MOVE STRING 1 DATA INTO DBUFØ
28Ø8	D49F C6 Ø3		LDB #\$Ø3	DSKCON WRITE OP CODE
28Ø9 281Ø	D4A1 8E 06 00 D4A4 9F EE	LD4A1	LDX #DBUFØ STX DCBPT	POINT X TO I/O BUFFER (DBUFØ) *
2811	D4A6 D7 EA		STB DCOPC	* SAVE NEW DSKCON BUFFER POINTER AND OP CODE VARIABLES
2812	D4A8 7E D5 FF		JMP LD5FF	GO WRITE OUT A SECTOR
2813 2814		* DSKIN	I COMMAND	
2815	D4AB 10 27 D1 70		LBEQ LA61F	BRANCH TO 'DN' ERROR IF NO DRIVE NUMBER SPECIFIED
2816	D4AF BD D1 69		JSR LD169	CALCULATE DRIVE NUMBER
2817 2818	D4B2 C6 Ø4 D4B4 9D A5		LDB #\$Ø4 JSR GETCCH	SKIP FACTOR DEFAULT VALUE GET CURRENT INPUT CHAR FROM BASiC
2819	D4B4 3D A3 D4B6 27 ØC		BEQ LD4C4	BRANCH IF END OF LINE
2820	D4B8 BD B7 38		JSR LB738	SYNTAX CHECK FOR COMMA AND EVALUATE EXPRESSION
2821 2822	D4BB C1 11 D4BD 10 24 DF 89		CMPB #17 LBHS LB44A	MAX VALUE OF SKIP FACTOR = 16
2823	D4C1 BD A5 C7		JSR LA5C7	'ILLEGAL FUNCTION CALL' IF BAD SKIP FACTOR SYNTAX ERROR IF MORE CHARACTERS ON THE LINE
2824	D4C4 34 Ø4	LD4C4	PSHS B	SAVE SKIP FACTOR ON THE STACK
2825	D4C6 8E Ø7 12		LDX #DBUF1+SECMAX	POINT TO END OF LOGICAL SECTOR NUMBER STORAGE AREA
2826 2827	D4C9 C6 12 D4CB 6F 82	LD4CB	LDB #SECMAX CLR ,-X	18 SECTORS PER TRACK CLEAR A BYTE IN THE BUFFER
2828	D4CD 5A		DECB	CLEARED ALL 18?
2829	D4CE 26 FB		BNE LD4CB	KEEP GOING IF NOT
283Ø 2831	D4DØ 4F D4D1 2Ø ØD		CLRA BRA LD4EØ	RESET PHYSICAL SECTOR COUNTER START WITH FIRST PHYSICAL SECTOR = 1
2832				
2833 2834	D4D3 EB E4	* CALCU LD4D3	ILATE LOGICAL SECTOR NUMBERS ADDB ,S	ADD SVID EACTOR TO LOCICAL SECTOR COUNTER
	D4D5 EB E4	LD4D5	INCB	ADD SKIP FACTOR TO LOGICAL SECTOR COUNTER ADD ONE TO LOGICAL SECTOR COUNTER
2836	D4D6 CØ 12	LD4D6	SUBB #SECMAX	SUBTRACT MAX NUMBER OF SECTORS
2837 2838	D4D8 24 FC D4DA CB 12		BHS LD4D6 ADDB #SECMAX	BRANCH UNTIL Ø > ACCB >= -18 ADD 18, NOW ACCB IS Ø-17
2839	D4DC 6D 85		TST B,X	IS ANYTHING STORED HERE ALREADY?
2840	D4DE 26 F5		BNE LD4D5	YES - GET ANOTHER SECTOR
2841 2842	D4EØ 4C D4E1 A7 85	LD4EØ	INCA STA B,X	* INCREMENT PHYSICAL SECTOR NUMBER AND * SAVE IT IN THE RAM BUFFER
2843	D4E3 81 12		CMPA #SECMAX	FINISHED WITH ALL SECTORS?
2844	D4E5 25 EC		BLO LD4D3	NO - KEEP GOING
2845 2846	D4E7 32 61 D4E9 8E 22 ØF		LEAS \$01,S LDX #DFLBUF+\$1888-2	REMOVE SKIP FACTOR FROM STACK GET TOP OF RAM USED BY DSKINI
2847	D4EG 9C 27		CMPX MEMSIZ	IS IT > CLEARED AREA?
2848	D4EE 10 22 D7 52		LBHI LAC44	'OUT OF MEMORY' ERROR IF > CLEARED AREA
2849 285Ø	D4F2 BD CA 3B D4F5 73 Ø9 5C		JSR DVEC7 COM DRESFL	CLOSE ALL FILES SET RESET FLAG TO \$FF - THIS WILL CAUSE A DOS RESET
2851	D4F8 10 CE 08 00		LDS #DBUF1+SECLEN	SET STACK TO TOP OF DBUF1
2852	D4FC BD 95 AC		JSR L95AC	RESET SAM TO DISPLAY PAGE ZERO AND ALPHA GRAPHICS
2853 2854	D4FF 86 ØØ D5Ø1 97 EA		LDA #\$ØØ STA DCOPC	YOU COULD DELETE THIS INSTRUCTION AND CHANGE FOLLOWING STA TO CLR RESTORE HEAD TO TRACK ZERO DSKCON OP CODE
2855	D501 97 EA D503 BD D5 FF		JSR LD5FF	RESTORE HEAD TO TRACK ZERO
2856	D5Ø6 7F Ø9 85		CLR RDYTMR	RESET THE READY TIMER
2857 2858	D5Ø9 86 CØ D5ØB B7 FF 48		LDA #\$CØ STA FDCREG	* FOC READ ADDRESS CODE
2859	D5ØE BD D6 DE		JSR LD6DE	CHECK DRIVE READY - WAIT UNTIL READY
2860	D511 10 26 00 86		LBNE LD59B	BRANCH IF NOT READY - ISSUE AN ERROR
2861 2862	D515 ØF EC D517 20 1A		CLR DCTRK BRA LD533	RESET TRACK NUMBER START THE FORMATTING PROCESS
2863	D519 81 16	LD519	CMPA #22	= CHECK FOR TRACK 22 (PRECOMPENSATION)
2864	D51B 25 Ø8		BLO LD525	= AND BRANCH IF < TRACK 22 - NO PRECOMP
2865 2866	D51D B6 Ø9 86 D52Ø 8A 1Ø		LDA DRGRAM ORA #\$10	* GET THE RAM IMAGE OF DSKREG, 'OR' * IN THE PRECOMPENSATION FLAG AND
2867	D522 B7 FF 40		STA DSKREG	* SEND IT TO DSKREG
2868	D525 86 53	LD525	LDA #\$53	= GET STEP IN COMMAND
2869 287Ø			STA FDCREG EXG A,A	= AND SEND IT TO THE 1793 * DELAY AFTER ISSUING COMMAND TO 1793
	D527 B7 FF 48 D52A 1E 88			
2871	D52A 1E 88 D52C 1E 88		EXG A,A	*
2871 2872	D52A 1E 88 D52C 1E 88 D52E BD D6 DE		EXG A,A JSR LD6DE	CHECK DRIVE READY
2871 2872 2873	D52A 1E 88 D52C 1E 88 D52E BD D6 DE D531 26 68	LD533	EXG A,A JSR LD6DE BNE LD59B	BRANCH IF NOT READY - ISSUE AN ERROR
2871 2872 2873 2874 2875	D52A 1E 88 D52C 1E 88 D52E BD D6 DE D531 26 68 D533 BD D6 FD D536 8D 6C	LD533	EXG A,A JSR LD6DE BNE LD59B JSR LD6FD BSR \$D5A4	BRANCH IF NOT READY – ISSUE AN ERROR WAIT A WHILE BUILD A FORMATTED TRACK IN RAM
2871 2872 2873 2874 2875 2876	D52A 1E 88 D52C 1E 88 D52E BD D6 DE D531 26 68 D533 BD D6 FD D536 8D 6C D538 1Ø 8E FF 4B	LD533	EXG A,A JSR LD6DE BNE LD59B JSR LD6FD BSR \$D5A4 LDY #FDCREG+3	BRANCH IF NOT READY - ISSUE AN ERROR WAIT A WHILE BUILD A FORMATTED TRACK IN RAM Y POINTS TO 1793 DATA REGISTER
2871 2872 2873 2874 2875	D52A 1E 88 D52C 1E 88 D52E BD D6 DE D531 26 68 D533 BD D6 FD D536 8D 6C D538 1Ø 8E FF 4B D53C 1A 5Ø	LD533	EXG A,A JSR LD6DE BNE LD59B JSR LD6FD BSR \$D5A4 LDY #FDCREG+3 ORCC #\$50	BRANCH IF NOT READY - ISSUE AN ERROR WAIT A WHILE BUILD A FORMATTED TRACK IN RAM Y POINTS TO 1793 DATA REGISTER DISABLE INTERRUPTS
2871 2872 2873 2874 2875 2876 2877	D52A 1E 88 D52C 1E 88 D52E BD D6 DE D531 26 68 D533 BD D6 FD D536 8D 6C D538 1Ø 8E FF 4B	LD533	EXG A,A JSR LD6DE BNE LD59B JSR LD6FD BSR \$D5A4 LDY #FDCREG+3	BRANCH IF NOT READY - ISSUE AN ERROR WAIT A WHILE BUILD A FORMATTED TRACK IN RAM Y POINTS TO 1793 DATA REGISTER

2881					
700T	D547 B6 FF 48		LDA	FDCREG	RESET STATUS OF THE 1793
2882	D54A 86 FF		LDA	#\$FF	* ENABLE THE NMI FLAG TO VECTOR
2883	D54C B7 Ø9 82		STA	NMIFLG	* OUT OF AN I/O LOOP UPON AN NMI INTERRUPT
2884	D54F C6 F4		LDB	#\$F4	= GET WRITE TRACK COMMAND AND
2885	D551 F7 FF 48		STB	FDCREG	= SEND TO 1793
2886 2887	D554 B6 Ø9 86 D557 8A 8Ø		LDA ORA	DRGRAM #\$80	* GET THE DSKREG RAM IMAGE AND 'OR' IN THE * FLAG WHICH WILL ENABLE THE 1793 TO HALT
2888	D557 BA 60 D559 B7 FF 40		STA	DSKREG	* THE 6809. SEND RESULT TO DSKREG
2889	D55C E6 8Ø	LD55C	LDB	,X+	= GET A BYTE FROM THE FORMATTED TRACK
2890	D55E E7 A4		STB	,Y	= RAM IMAGE, SEND IT TO THE 1793 AND
2891	D560 20 FA		BRA	ĹD55C	= LOOP BACK TO GET ANOTHER BYTE
2892					
2893	D562 B6 FF 48	LD562		FDCREG	GET STATUS
2894	D565 1C AF		ANDCO		ENABLE INTERRUPTS
2895	D567 84 44 D569 97 FØ			#\$44 DCSTA	* KEEP ONLY WRITE PROTECT & LOST DATA
2896 2897	D56B 26 2E		STA BNE	DCSTA LD59B	* AND SAVE IT IN THE DSKCON STATUS BYTE BRANCH IF ERROR
2898	D56D ØC EC		INC	DCTRK	SKIP TO THE NEXT TRACK
2899	D56F 96 EC		LDA	DCTRK	GET THE TRACK NUMBER
2900	D571 81 23		CMPA	#TRKMAX	WAS IT THE LAST TRACK
2901	D573 26 A4		BNE	LD519	NO - KEEP GOING
2902					
2903	DE7E OC 00	* VERIF		ALL SECTORS ARE READABLE	- CET THE DEVECON DEAD OR CODE
29Ø4 29Ø5	D575 86 Ø2 D577 97 EA		STA	#\$Ø2 DCOPC	= GET THE DSKCON READ OP CODE = AND SAVE IT IN THE DSKCON VARIABLE
2906	D579 8E Ø6 ØØ		LDX	#DBUFØ	* POINT THE DSKCON BUFFER POINTER
2907	D57C 9F EE		STX	DCBPT	* TO DBUFØ
29Ø8	D57E CE Ø7 ØØ		LDU	#DBUF1	POINT U TO THE LOGICAL SECTOR NUMBERS
2909	D581 4F		CLRA	DOTRU	RESET THE TRACK COUNTER TO ZERO
2910	D582 97 EC	LD582	STA	DCTRK	SET THE DSKCON TRACK VARIABLE
2911 2912	D584 5F D585 A6 C5	LD585	CLRB LDA	B,U	RESET THE SECTOR COUNTER GET THE PHYSICAL SECTOR NUMBER
2912	D587 97 ED	FD303	STA	DSEC	SAVE DSKCON SECTOR VARIABLE
	D589 BD D5 FF		JSR	LD5FF	READ A SECTOR
2915	D58C 5C		INCB		* INCREMENT THE SECTOR COUNTER
2916	D58D C1 12		CMPB	#SECMAX	* AND COMPARE IT TO MAXIMUM SECTOR NUMBER
2917	D58F 25 F4		BLO	LD585	* AND KEEP LOOPING IF MORE SECTORS LEFT
2918	D591 96 EC		LDA	DCTRK	= GET THE CURRENT TRACK NUMBER
2919 2920	D593 4C D594 81 23		INCA	#TRKMAX	= ADD ONE TO IT, COMPARE TO THE MAXIMUM TRACK = NUMBER AND KEEP LOOPING IF
2921	D594 61 23 D596 25 EA			LD582	= THERE ARE STILL TRACKS TO DO
2922	D598 7E D1 EØ		JMP	LD1EØ	GO CHECK FOR A DOS RESET
2923	D59B 7F Ø9 86	LD59B	CLR	DRGRAM	CLEAR RAM IMAGE OF DSKREG
2924	D59E 7F FF 40		CLR	DSKREG	CLEAR DSKREG - TURN DISK MOTORS OFF
	· D5A1 7E D6 ØE		JMP	LD6ØE	PROCESS DRIVES NOT READY ERROR
2926 2927		* BIITI D	V EUDI	MATTED TRACK OF DATA IN RAM START:	ING AT DELBUE
2928		DOILD	7 1010	TATLES TRACK OF BATA IN RAIT START.	THE AT DIEDOT.
2929	D5A4 8E Ø9 89		LDX	#DFLBUF	START TRACK BUFFER AT DFLBUF
2930	D5A7 CC 20 4E		LDD	#\$204E	GET SET TO WRITE 32 BYTES OF \$4E
2931	D5AA 8D 29		BSR	LD5D5	GO WRITE GAP IV
2932	D5AC 5F	LDEAD	CLRB	D.	RESET SECTOR COUNTER
2933			PSHS		SAVE SECTOR COUNTER
	D5AD 34 Ø4	LD5AD			
2934 2935	D5AF CE Ø7 ØØ	LUJAU	LDU	#DBUF1 B.U	POINT U TO THE TABLE OF LOGICAL SECTORS  * GET LOGICAL SECTOR NUMBER FROM TABLE AND
2934		LUJAU		#DBOF1 B,U DSEC	* GET LOGICAL SECTOR NUMBER FROM TABLE AND  * SAVE IT IN THE DSKCON VARIABLE
2934 2935 2936 2937	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7	LUJAU	LDU LDB STB LDU	B,U	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA
2934 2935 2936 2937 2938	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3	LUJAU	LDU LDB STB LDU LDB	B,U DSEC #LD5E7 #\$Ø3	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND
2934 2935 2936 2937 2938 2939	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5BB 8D 1E	LUSAU	LDU LDB STB LDU LDB BSR	B,U DSEC #LD5E7 #\$03 LD5DB	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER
2934 2935 2936 2937 2938 2939 2940	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 BD 1E D5BD 96 EC	LUSAU	LDU LDB STB LDU LDB BSR LDA	B,U DSEC #LD5E7 #\$Ø3 LD5DB DCTRK	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE 1T
2934 2935 2936 2937 2938 2939 2940 2941	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5BB 8D IE D5BD 96 EC D5BF A7 8Ø	LUJAU	LDU LDB STB LDU LDB BSR LDA STA	B,U DSEC #LDSE7 #\$83 LD5DB DCTRK ,X+	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER
2934 2935 2936 2937 2938 2939 2940	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 BD 1E D5BD 96 EC	LUJAU	LDU LDB STB LDU LDB BSR LDA	B,U DSEC #LD5E7 #\$Ø3 LD5DB DCTRK	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE 1T
2934 2935 2936 2937 2938 2939 2940 2941 2942 2943 2944	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 BD 1E D5BD 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø	LUJAU	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA	B,U DSEC #LD5E7 #\$Ø3 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER
2934 2935 2936 2937 2938 2939 2940 2941 2942 2943 2944 2945	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 BD 1E D5BD 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C5 A7 8Ø	EBSAU	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDA	B,U DSEC #\$03 LD5DB DCTRK ,X+ ,X+ DSEC ,X+ #\$09	* GET LOGICAL SECTOR NUMBER FROM TABLE AND  * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA  * GET FIRST 3 DATA BLOCKS AND  * WRITE THEM TO BUFFER  = GET TRACK NUMBER AND STORE IT  = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER  * GET SECTOR NUMBER AND  * STORE IT IN THE BUFFER  = GET THE LAST NINE DATA BLOCKS AND
2934 2935 2936 2937 2938 2939 2940 2941 2942 2943 2944 2945 2946	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5BD 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø	LUJAU	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR	B,U DSEC #LD5E7 #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LD5DB	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER
2934 2935 2936 2937 2938 2939 2940 2941 2942 2943 2944 2945 2946 2947	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 8D 1E D5BD 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C3 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5CB 35 Ø4	LUJAU	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR PULS	B,U DSEC #LDSE7 #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ ,X+ BSEC ,X+ #\$09 LD5DB B	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER
2934 2935 2936 2937 2938 2939 2940 2941 2942 2943 2944 2945 2946 2947 2948	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5BD 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C3 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 35 Ø4 D5C8 B5 Ø4	LUJAU	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR PULS INCB	B,U DSEC ##803 LD5DB DCTRK ,X+ ,X+ DSEC ,X+ #\$09 LD5DB B	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER
2934 2935 2936 2937 2938 2939 2940 2941 2942 2943 2944 2945 2946 2947 2948 2949	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 8D 1E D5BD 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C3 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5CB 35 Ø4	LUJAU	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR PULS INCB CMPB	B,U DSEC #LDSE7 #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ ,X+ BSEC ,X+ #\$09 LD5DB B	* GET LOGICAL SECTOR NUMBER FROM TABLE AND  * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA  * GET FIRST 3 DATA BLOCKS AND  * WRITE THEM TO BUFFER  = GET TRACK NUMBER AND STORE IT  = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER  * GET SECTOR NUMBER AND  * STORE IT IN THE BUFFER  = GET THE LAST NINE DATA BLOCKS AND  = WRITE THEM TO THE BUFFER  GET SECTOR COUNTER  NEXT SECTOR COUNTER
2934 2935 2936 2937 2938 2939 2941 2942 2943 2944 2945 2946 2947 2948 2949 2950 2950	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5BB B0 1E D5BB FA7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 35 Ø4 D5C0 5C D5CE C1 12	LUJAU	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR PULS INCB CMPB BLO	B,U DSEC #LD5E7 #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LD5DB B #SECMAX	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK
2934 2935 2936 2937 2938 2939 2940 2941 2942 2943 2944 2945 2946 2946 2948 2949 2951 2951 2952	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5BB 8D IE D5BD 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D IØ D5C8 35 Ø4 D5CB 5C D5CE C1 12 D5D0 25 DB		LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR LDB BSR PULS INCB CMPB BLO LDD	B, U DSEC #LD5E7 #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LD5DB B #SECMAX LD5AD #\$C84E	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE
2934 2935 2936 2937 2938 2940 2941 2942 2943 2944 2945 2946 2947 2948 2950 2950 2951 2952	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 IE D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 10 D5C8 B5 Ø4 D5C0 5C D5CE C1 12 D5DØ 25 DB D5DØ 25 DB	* WRITE	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR PULS INCB CMPB BL0 LDD	B,U DSEC #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ BS09 LD5DB B #\$ECMAX LD5DB #\$C84E	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR
2934 2935 2936 2937 2938 2940 2941 2942 2943 2944 2945 2947 2948 2949 2950 2950 2951 2952 2953 2954	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5BD 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C5 A7 8Ø D5C5 BA 10 D5C8 35 Ø4 D5C8 35 Ø4 D5C0 5C D5CE C1 12 D5DØ 25 DB D5D2 CC C8 4E		LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR PULS INCB CMPB BLO LDD  ACCA STB	B,U DSEC #LD5E7 #\$83 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$89 LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NIME DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALLS SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK STORE A BYTE IN THE BUFFER
2934 2935 2936 2937 2938 2940 2941 2942 2943 2944 2945 2946 2947 2948 2949 2951 2952 2953 2953 2955	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 35 Ø4 D5C0 5C D5CE C1 12 D5D0 25 D8 D5D0 25 C8 4E	* WRITE	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR LDA STA LDB BSR LDA STA LDB STB BLO LDD ACCA	B,U DSEC ##LD5E7 #\$Ø3 LD5DB DCTRK ,X+ ,X+ ,X+ ,X+ #\$Ø9 LD5DB B #SECMAX LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER DECREMENT COUNTER
2934 2935 2936 2937 2938 2939 2940 2941 2942 2943 2944 2945 2946 2947 2948 2951 2951 2953 2954 2955 2956	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5BD 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C5 A7 8Ø D5C5 BA 10 D5C8 35 Ø4 D5C8 35 Ø4 D5C0 5C D5CE C1 12 D5DØ 25 DB D5D2 CC C8 4E	* WRITE	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR LDA STA LDB BSR LDA STA LDB STB BLO LDD ACCA	B,U DSEC #LD5E7 #\$83 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$89 LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NIME DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALLS SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK STORE A BYTE IN THE BUFFER
2934 2935 2936 2937 2938 2940 2941 2942 2943 2944 2946 2947 2947 2951 2952 2953 2954 2955 2956 2957 2958	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 35 Ø4 D5C0 5C D5CE C1 12 D5D0 25 D8 D5D0 25 C8 4E	* WRITE	LDU LDB STB LDU LDB BSR LDA STA CLR LDA STA LDB BSR PULS INCB CMPB BLO LDD  ACCA STB DECA BNE	B,U DSEC ##405 #\$83 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$89 LD5DB B #SECMAX LD5AD #\$4584 BSYTES OF ACCB INTO BUFFER ,X+ LD5D5	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER DECREMENT COUNTER
2934 2935 2936 2937 2938 2949 2941 2942 2943 2944 2945 2946 2947 2948 2950 2951 2953 2954 2953 2954 2957 2956 2957	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 BD 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 10 D5C8 B5 Ø4 D5C0 5C D5CE C1 12 D5DØ 25 DB D5D2 CC C8 4E	* WRITE LUSDS	LDU LDB STB LDU LDB BSR CLDA STA CLR LDA LDB BSR PULS STA LDB BSR PULS STB BLO LDD ACCA BNE RTS PSHS	B,U DSEC ##LD5E7 #\$93 LD5DB DCTRK ,X+ ,X+ ,X+ ,X+ B99 LD5DB B #SECMAX LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B #,U++	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR IN SECTOR BE SECTOR SECTOR SOND DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BCCREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET THO BYTES OF DATA FROM THE TABLE
2934 2935 2936 2937 2939 2940 2941 2942 2943 2944 2945 2946 2947 2951 2951 2952 2953 2954 2955 2955 2956 2957 2958 2958 2958	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 S6 D5C6 C1 12 D5D0 25 DB D5D0 25 CB D5D0 25 CB D5C5 C7 8Ø D5D0 25 CB D5D0 25	* WRITE LUSDS	LDU LDB LDU LDB BSR LDA STA LDB STA LDB STA LDB LDA STA LDB BSR PULS INCB CMPB BLO LDD ACCA STB ACCA LDD BNE RTS PSHS LDD BNE RTS PSHS LDD BSR	B,U DSEC ##105E7 #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LD5DB B #SECMAX LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B #U++ LD5D5	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET SECTOR COUNTER BUTTE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BECREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER
2934 2935 2936 2937 2938 2940 2941 2942 2943 2944 2946 2947 2951 2952 2953 2954 2955 2956 2956 2958 2958 2958 2959 2950 2950 2951	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 BD IE D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D IØ D5C8 35 Ø4 D5C0 5C D5CE C1 12 D5D0 25 DB D5D2 CC C8 4E	* WRITE LUSDS	LDU LDB LDA LDA STA LDB BSR ACLR LDB BSR BSR INCB LDD ACCA STB DECA BNE RT PSHS LDD BSR PSHS LDD	B,U DSEC #\$03 LD5DB DCTRK ,X+ ,X+ ,X+  #\$89 LD5DB B #\$ECMAX LD5DB B #\$ECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B #U++ LD5D5 B B	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BERANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT
2934 2935 2936 2937 2938 2949 2941 2942 2943 2945 2946 2947 2948 2950 2951 2952 2953 2954 2955 2956 2957 2958 2958 2958 2958 2958 2958 2958 2958	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 BD 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 35 Ø4 D5CD 5C D5CE C1 12 D5DØ 25 DB D5D2 CC C8 4E D5D5 E7 8Ø D5D3 39 D5D8 34 Ø4 D5D0 39 D5D9 34 Ø4 D5D0 35 W4 D5D0 39 D5D8 34 Ø4 D5D0 36 CC C1 D5DF 8D F4 D5DF 8D F4 D5E1 35 Ø4 D5E3 5A	* WRITE LUSDS	LDU LDB STB LDU LDB SSR LDA STA LDB BSR PULS INCB LDD ACCA STB LDD ACCA BNE RTS LDD BSR RTS LDD BSR RTS LDD BSR PULS LDD BSR PULS LDD BSR LDD	B,U DSEC ##LD5E7 #\$#93 LD5DB DCTRK ,X+ ,X+ ,X+ ,X+ #\$#99 LD5DB B #SECMAX LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B ,U++ LD5D5 B	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BCEREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCE INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS
2934 2935 2936 2937 2939 2940 2941 2942 2943 2944 2945 2946 2947 2951 2952 2953 2954 2955 2955 2955 2956 2957 2958 2959 2960 2961 2962 2961 2962 2963	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 35 Ø4 D5C0 5C D5CE C1 12 D5DØ 25 DB D5D0 25 CB D5C0 CC B 4E  D5D5 E7 8Ø D5D1 A4 D5D8 26 FB D5D0 39 D5D8 34 Ø4 D5D8 39 D5D8 34 Ø4 D5D8 36 FB D5D7 80 F4 D5D8 35 Ø4 D5D7 80 F4	* WRITE LUSDS	LDU LDB STB LDU LDB BSR LDA CLR LDA STB LDD LDD ACCA BNE RTS PSHS LDD BSR PULS BSR PULS BER BLD DECA BNE RTS PSHS LDD BCB BSR PULS BSR PULS BSR PULS BSR PULS BSR PULS BSR PULS BSR BNE	B,U DSEC #\$03 LD5DB DCTRK ,X+ ,X+ ,X+  #\$89 LD5DB B #\$ECMAX LD5DB B #\$ECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B #U++ LD5D5 B B	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BERANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT
2934 2935 2936 2937 2939 2940 2941 2942 2943 2944 2945 2946 2947 2951 2952 2953 2954 2955 2955 2955 2956 2957 2958 2959 2960 2961 2962 2961 2962 2963	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 BD 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 35 Ø4 D5CD 5C D5CE C1 12 D5DØ 25 DB D5D2 CC C8 4E D5D5 E7 8Ø D5D3 39 D5D8 34 Ø4 D5D0 39 D5D9 34 Ø4 D5D0 35 W4 D5D0 39 D5D8 34 Ø4 D5D0 36 CC C1 D5DF 8D F4 D5DF 8D F4 D5E1 35 Ø4 D5E3 5A	* WRITE LUSDS	LDU LDB STB LDU LDB SSR LDA STA LDB BSR PULS INCB LDD ACCA STB LDD ACCA BNE RTS LDD BSR RTS LDD BSR RTS LDD BSR PULS LDD BSR PULS LDD BSR LDD	B,U DSEC ##LD5E7 #\$#93 LD5DB DCTRK ,X+ ,X+ ,X+ ,X+ #\$#99 LD5DB B #SECMAX LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B ,U++ LD5D5 B	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BCEREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCE INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS
2934 2935 2936 2937 2938 2949 2941 2942 2943 2944 2945 2947 2948 2947 2950 2951 2952 2953 2954 2955 2956 2957 2957 2958 2959 2959 2961 2962 2962 2962 2963 2964	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 S6 D5C0 CC B4E D5D5 E7 8Ø D5D7 4A D5D8 26 FB D5D8 34 Ø4 D5D8 34 Ø4 D5D8 35 Ø4 D5D7 8D F4 D5E3 5A D5E4 26 F5	* WRITE LD5D5 LD5DB	LDU LDB STB LDU LDB BSR LDA LDB BSR LDA LDB BSR LDA ACCA LDB BNE RTS PSHS LDD BSR PULS BNE RTS PSHS LDB BNE RTS	B,U DSEC ##LD5E7 #\$#93 LD5DB DCTRK ,X+ ,X+ ,X+ ,X+ #\$#99 LD5DB B #SECMAX LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B ,U++ LD5D5 B	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER = GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BCEREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCE INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS
2934 2935 2936 2937 2938 2949 2941 2942 2943 2944 2945 2947 2948 2949 2951 2951 2952 2953 2954 2956 2957 2956 2957 2958 2959 2961 2962 2964 2964 2962 2964 2964 2965 2966 2966 2966 2966 2966 2967	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 S6 D5C0 CC B4E D5D5 E7 8Ø D5D7 4A D5D8 26 FB D5D8 34 Ø4 D5D8 34 Ø4 D5D8 35 Ø4 D5D7 8D F4 D5E3 5A D5E4 26 F5	* WRITE LD5D5 LD5DB	LDU LDB STB LDU LDB BSR LDA CLR LDA CLR LDB BSR LDA LDB BSR LDA LDB BSR LDA LDB BSR LDD LDD ACCA STB BNE RTS LDD BSR LDD BSR RTS RTS LDD BSR RTS RTS LDD BSR RTS RTS RTS RTS RTS RTS RTS RTS RTS R	B,U DSEC #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ #\$09 BSEC ,X+ #\$89 LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B B LD5DB B LD5DB D COMMAN A SECTOR ON THE DISK	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER DECREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS * NOT DONE
2934 2935 2936 2937 2938 2949 2941 2942 2943 2945 2946 2947 2950 2951 2952 2953 2954 2956 2957 2958 2959 2959 2959 2959 2959 2959 2959	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 S6 D5C0 CC B4E D5D5 E7 8Ø D5D7 4A D5D8 26 FB D5D8 34 Ø4 D5D8 34 Ø4 D5D8 35 Ø4 D5D7 8D F4 D5E3 5A D5E4 26 F5	* WRITE LD5D5 LD5DB * DATA (	LDU LDB STB LDU LDB BSR LDA LDB BSR LDA LDB LDD LDD ACCA STB DECA BNE RTS PSHS DECB BNE RTS STB LDB LDD LDBSR RTS STB LDB LDD LDBSR RTS STB LDB LDB LDB LDB LDB LDB LDB LDB LDB LD	B,U DSEC #LODET #\$83 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$99 LD5DB B #SECMAX LD5AD #\$C84E B #LD5D5 B LD5D5 B LD5D5 B LD5D5 B LD5D6 B LD5D6 B LD5D7 ACCB INTO BUFFER ACCB INTO BUFF	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET SECTOR COUNTER NEXT SECTOR COUNTER NEXT SECTOR COUNTER NEXT SECTOR COUNTER NEXT SECTOR OUNTER SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER DECREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET THO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS * NOT DONE
2934 2935 2936 2937 2939 2940 2941 2942 2943 2944 2945 2946 2947 2951 2952 2953 2954 2955 2955 2956 2957 2958 2959 2960 2961 2962 2963 2964 2966 2967 2966 2967 2966 2967 2967 2968 2967 2968	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 S6 D5C0 CC B4E D5D5 E7 8Ø D5D7 4A D5D8 26 FB D5D8 34 Ø4 D5D8 34 Ø4 D5D8 35 Ø4 D5D7 8D F4 D5E3 5A D5E4 26 F5	* WRITE LD5D5  LD5DB  * DATA   * THESE   * DOUBL	LDU LDB STB LDU LDB SSR LDA STA LDB BSR LDD LDD LDD ACCA STB LDD ACCA BNE RTS PSHS LDD DECB BSR RTS STB LDD LDD LDD LDD LDD LDD LDD LDD LDD LD	B,U DSEC #LODET #\$03 LD5DB DCTRK ,X+ ,X+ DSEC ,X+ #\$09 LD5DB B  #SECMAX LD5AD #\$C84E EYTES OF ACCB INTO BUFFER ,X+ LD5D5 B  LD5D5 B LD5DB D FORMAT A SECTOR ON THE DISK  ARE CLOSE TO THE IBM SYSTEM 34 FOILTY. THE FORMAT GENERALLY CONFORM.	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER DECREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS * NOT DONE  RRAT FOR 256 BYTE SECTORS. S TO THAT SPECIFIED ON THE
2934 2935 2936 2937 2938 2949 2941 2942 2943 2944 2945 2946 2947 2951 2951 2953 2954 2955 2956 2957 2958 2959 2961 2962 2963 2964 2965 2964 2965 2967 2968 2968 2968 2968 2968 2968 2968 2968	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 S6 D5C0 CC B4E D5D5 E7 8Ø D5D7 4A D5D8 26 FB D5D8 34 Ø4 D5D8 34 Ø4 D5D8 35 Ø4 D5D7 8D F4 D5E3 5A D5E4 26 F5	* WRITE LD5D5 LD5DB * DATA   * THESE * DOUBL! * 1793	LDU LDB STB LDU LDB BSR LDA LDB BSR LDA LDB BSR LDA LDB BSR LDA LDB LDD LDD LDD LDD LDD LDD LDD LDD LDD	B,U DSEC #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ ,X+ #\$09 LD5DB B  #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B  LD5DB  C DFORMAT A SECTOR ON THE DISK  ARE CLOSE TO THE IBM SYSTEM 34 FOIL TY. THE FORMAT GENERALLY CONFORMHEET. THE GAP SIZES HAVE BEEN REDI	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET THE LAST NINE DATA BLOCKS AND = WRITE THEM TO THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER DECREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS * NOT DONE  RMAT FOR 256 BYTE SECTORS. \$ TO THAT SPECIFIED ON THE JCED TO THE MINIMUM
2934 2935 2936 2937 2939 2940 2941 2942 2943 2944 2945 2946 2947 2951 2952 2953 2954 2955 2955 2956 2957 2958 2959 2960 2961 2962 2963 2964 2966 2967 2966 2967 2966 2967 2967 2968 2967 2968	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 S6 D5C0 CC B4E D5D5 E7 8Ø D5D7 4A D5D8 26 FB D5D8 34 Ø4 D5D8 34 Ø4 D5D8 35 Ø4 D5D7 8D F4 D5E3 5A D5E4 26 F5	* WRITE LD5D5 LD5DB * DATA ( * THESE * DOUBL * 1793 ( * ALLOW)	LDU LDB STB LDU LDB BSR LDA ACCA LDB BSR LDD ACCA LDD BSR RTS PSHS LDD BSR RTS PSHS LDC BSR RTS PSHS LDC BSR RTS PSHS LDC BSR RTS LDC BSR	B,U DSEC #LODET #\$03 LD5DB DCTRK ,X+ ,X+ DSEC ,X+ #\$09 LD5DB B  #SECMAX LD5AD #\$C84E EYTES OF ACCB INTO BUFFER ,X+ LD5D5 B  LD5D5 B LD5DB D FORMAT A SECTOR ON THE DISK  ARE CLOSE TO THE IBM SYSTEM 34 FOILTY. THE FORMAT GENERALLY CONFORM.	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BET SECTOR COUNTER NEXT SECTOR SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF SAE AT END OF TRACK  STORE A BYTE IN THE BUFFER DECREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET THO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS * NOT DONE  RMAT FOR 256 BYTE SECTORS. S TO THAT SPECIFIED ON THE JCCD TO THE MINHUM LL CHARACTER FOR THE DATA
2934 2935 2936 2937 2948 2949 2941 2942 2943 2944 2945 2946 2947 2951 2952 2953 2954 2955 2955 2956 2957 2958 2959 2960 2961 2962 2963 2964 2966 2967 2966 2967 2967 2967 2969 2971 2972	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 S6 D5C0 CC B4E D5D5 E7 8Ø D5D7 4A D5D8 26 FB D5D8 34 Ø4 D5D8 34 Ø4 D5D8 35 Ø4 D5D7 8D F4 D5E3 5A D5E4 26 F5	* WRITE LD5D5 LD5DB * DATA ( * THESE * DOUBL * 1793 ( * ALLOW)	LDU LDB STB LDU LDB BSR LDA ACCA LDB BSR LDD ACCA LDD BSR RTS PSHS LDD BSR RTS PSHS LDC BSR RTS PSHS LDC BSR RTS PSHS LDC BSR RTS LDC BSR	B,U DSEC #LOSE7 #\$03 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LD5DB B  #SECMAX LD5AD #\$CRACE INTO BUFFER ,X+ LD5D5 B  LD5D5 B LD5DB D FORMAT A SECTOR ON THE DISK ARE CLOSE TO THE IBM SYSTEM 34 FOI ITY. THE FORMAT GENERALLY CONFORM. HEET. THE GAP SIZES HAVE BEEN REDI THE IBM FORMAT USES \$40 AS THE FII E COLOR DOS USES AN \$FF AS THE FII	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BET SECTOR COUNTER NEXT SECTOR SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF SAE AT END OF TRACK  STORE A BYTE IN THE BUFFER DECREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET THO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS * NOT DONE  RMAT FOR 256 BYTE SECTORS. S TO THAT SPECIFIED ON THE JCCD TO THE MINHUM LL CHARACTER FOR THE DATA
2934 2935 2936 2937 2938 2949 2941 2942 2943 2944 2945 2946 2950 2951 2952 2953 2954 2955 2956 2957 2958 2959 2960 2961 2962 2963 2964 2965 2966 2967 2968 2968 2969 2971 2972 2973 2974	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 35 Ø4 D5C0 5C D5CE C1 12 D5DØ 25 DB D5D2 CC C8 4E D5DA 39 D5D8 34 Ø4 D5DA 39 D5DB 34 Ø4 D5DB 34 Ø4 D5DB 35 A D5E4 26 F5 D5E6 39 D5E7 Ø8 ØØ D5E3 99 D5E4 26 F5 D5E6 39	* WRITE LD5D5 LD5DB * DATA   * THESE * DOUBL! * 1793 ; * ALLOW * BLOCK	LDU LDB STB LDU LDB BSR LDA LDB LDD LDB LDD LDD LDD LDD LDD LDD LDD	B,U DSEC ##105E7 #\$93 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B LD5DB B B LD5DB	* GET LOGICAL SECTOR NUMBER FROM TABLE AND * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA * GET FIRST 3 DATA BLOCKS AND * WRITE THEM TO BUFFER = GET TRACK NUMBER AND STORE IT = IN THE RAM BUFFER CLEAR A BYTE (SIDE NUMBER) IN BUFFER * GET SECTOR NUMBER AND * STORE IT IN THE BUFFER GET SECTOR COUNTER NEXT SECTOR 18 SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER BET SECTOR COUNTER NEXT SECTOR SECTORS PER TRACK BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF A4E AT END OF TRACK  STORE A BYTE IN THE BUFFER DECREMENT COUNTER BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE WRITE ACCA BYTES OF ACCB INTO THE BUFFER * GET THE COUNTER BACK, DECREMENT * IT AND BRANCH IF ALL DATA BLOCKS * NOT DONE  RMAT FOR 256 BYTE SECTORS. S TO THAT SPECIFIED ON THE JUICED TO THE MINHUM LL CHARACTER FOR THE DATA LL CHARACTER. SYNC FIELD
2934 2935 2936 2937 2938 2940 2941 2942 2943 2944 2945 2946 2947 2951 2952 2953 2954 2955 2955 2956 2957 2958 2959 2960 2961 2962 2963 2964 2966 2967 2966 2967 2969 2971 2972 2973 2974 2977 2973 2974	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 IE D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 10 D5C8 B5 Ø4 D5C0 5C D5CE C1 12 D5D0 25 DB D5D2 CC C8 4E D5D5 E7 8Ø D5D3 A9 D5D3 A9 D5D3 A9 D5D4 AA D5D5 E7 B0 D5D8 A9 D5D8	* WRITE LD5D5 LD5DB * DATA (* THESE * DOUBL(* 1793) * ALLOW, * BLOCK(LD6D4)	LDU LDB STB LDU LDB BSR LDA LDB BSR LDA LDB BSR LDA LDB BSR LDA LDB LDD LDD LDD LDD LDD LDD LDD LDD LDD	B,U DSEC #LOSE7 #\$03 LDSDB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LDSDB B  #SECMAX LDSAD #\$CRACE BYTES OF ACCB INTO BUFFER ,X+ LDSD5 B LDSD5 B LDSD5 B LDSDB D FORMAT A SECTOR ON THE DISK ARE CLOSE TO THE IBM SYSTEM 34 FOI ITY. THE FORMAT GENERALLY CONFORM. HEET. THE GAP SIZES HAVE BEEN REDI THE IBM FORMAT USES \$40 AS THE FII 8,0 3,\$F5 1,\$FE	* GET LOGICAL SECTOR NUMBER FROM TABLE AND  * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA  * GET FIRST 3 DATA BLOCKS AND  * WRITE THEM TO BUFFER  GET TRACK NUMBER AND STORE IT  IN THE RAM BUFFER  CLEAR A BYTE (SIDE NUMBER) IN BUFFER  * GET SECTOR NUMBER AND  * STORE IT IN THE BUFFER  GET SECTOR COUNTER  WRITE THEM TO THE BUFFER  GET SECTOR COUNTER  NEXT SECTOR  18 SECTORS PER TRACK  BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER  DECREMENT COUNTER  BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE MRITE ACCA BYTES OF ACCB INTO THE BUFFER  * GET THE COUNTER BACK, DECREMENT  * IT AND BRANCH IF ALL DATA BLOCKS  * NOT DONE  RMAT FOR 256 BYTE SECTORS.  S TO THAT SPECIFIED ON THE JCED TO THE MINIMUM  L CHARACTER FOR THE DATA L CHARACTER.  SYNC FIELD  ID ADDRESS MARK (AM1)
2934 2935 2936 2937 2938 2949 2941 2942 2943 2944 2945 2946 2950 2951 2952 2953 2954 2955 2956 2957 2958 2959 2960 2961 2962 2963 2964 2965 2966 2967 2968 2968 2969 2971 2972 2973 2974	D5AF CE Ø7 ØØ D5B2 E6 C5 D5B4 D7 ED D5B6 CE D5 E7 D5B9 C6 Ø3 D5B8 B0 1E D5B0 96 EC D5BF A7 8Ø D5C1 6F 8Ø D5C3 96 ED D5C5 A7 8Ø D5C7 C6 Ø9 D5C9 8D 1Ø D5C8 35 Ø4 D5C0 5C D5CE C1 12 D5DØ 25 DB D5D2 CC C8 4E D5DA 39 D5D8 34 Ø4 D5DA 39 D5DB 34 Ø4 D5DB 34 Ø4 D5DB 35 A D5E4 26 F5 D5E6 39 D5E7 Ø8 ØØ D5E3 SA D5E4 26 F5 D5E6 39	* WRITE LD5D5 LD5DB * DATA (* THESE * DOUBL(* 1793) * ALLOW, * BLOCK(LD6D4)	LDU LDB STB LDU LDB BSR LDA LDB BSR LDA LDB BSR LDA LDB BSR LDA LDB LDD LDD LDD LDD LDD LDD LDD LDD LDD	B,U DSEC ##105E7 #\$93 LD5DB DCTRK ,X+ ,X+ ,X+ DSEC ,X+ #\$09 LD5DB B #SECMAX LD5AD #\$C84E BYTES OF ACCB INTO BUFFER ,X+ LD5D5 B LD5DB B B LD5DB	* GET LOGICAL SECTOR NUMBER FROM TABLE AND  * SAVE IT IN THE DSKCON VARIABLE POINT U TO TABLE OF SECTOR FORMATTING DATA  * GET FIRST 3 DATA BLOCKS AND  * WRITE THEM TO BUFFER  GET TRACK NUMBER AND STORE IT  IN THE RAM BUFFER  CLEAR A BYTE (SIDE NUMBER) IN BUFFER  * GET SECTOR NUMBER AND  * STORE IT IN THE BUFFER  GET SECTOR COUNTER  WRITE THEM TO THE BUFFER  GET SECTOR COUNTER  NEXT SECTOR  18 SECTORS PER TRACK  BRANCH IF ALL SECTORS NOT DONE WRITE 200 BYTES OF \$4E AT END OF TRACK  STORE A BYTE IN THE BUFFER  DECREMENT COUNTER  BRANCH IF ALL BYTES NOT MOVED  SAVE THE COUNTER ON THE STACK GET TWO BYTES OF DATA FROM THE TABLE MRITE ACCA BYTES OF ACCB INTO THE BUFFER  * GET THE COUNTER BACK, DECREMENT  * IT AND BRANCH IF ALL DATA BLOCKS  * NOT DONE  RMAT FOR 256 BYTE SECTORS.  S TO THAT SPECIFIED ON THE JCED TO THE MINIMUM  L CHARACTER FOR THE DATA L CHARACTER.  SYNC FIELD  ID ADDRESS MARK (AM1)

2977	D5ED Ø1 Ø1		FCB 1,1	CECTOD C17E (2EC DVTE CECTODE)
2978	D5EF Ø1 F7		FCB 1,1 FCB 1,\$F7	SECTOR SIZE (256 BYTE SECTORS) CRC REQUEST
2979	D5F1 16 4E		FCB 22,\$4E	GAP II (POST-ID GAP)
2980	D5F3 ØC ØØ		FCB 12,0	SYNC FIELD
2981 2982	D5F5 Ø3 F5 D5F7 Ø1 FB		FCB 3,\$F5 FCB 1,\$FB	DATA ADDRESS MARK (AM2)
2983	D5F9 ØØ FF		FCB Ø,\$FF	DATA FIELD (256 BYTES)
2984	D5FB Ø1 F7		FCB 1,\$F7	CRC REQUEST
2985	D5FD 18 4E		FCB 24,\$4E	GAP III (POST DATA GAP)
2986 2987				
2988	D5FF 34 Ø4	LD5FF	PSHS B	SAVE ACCB
2989	D6Ø1 C6 Ø5		LDB #\$Ø5	5 RETRIES
2990	D603 F7 09 88		STB ATTCTR	SAVE RETRY COUNT
2991	D606 35 04	I DCGO	PULS B	RESTORE ACCB
2992 2993	D608 8D 62 D60A 0D F0	LD6Ø8	BSR DSKCON TST DCSTA	GO EXECUTE COMMAND CHECK STATUS
2994			BEQ LD61B	BRANCH IF NO ERRORS
2995		LD6ØE	LDA DCSTA	GET DSKCON ERROR STATUS
2996	D610 C6 3C		LDB #2*3Ø	'WRITE PROTECTED' ERROR
2997 2998	D612 85 40 D614 26 02		BITA #\$40 BNE LD618	CHECK BIT 6 OF STATUS BRANCH IF WRITE PROTECT ERROR
2999	D616 C6 28	LD616 LD618 LD61B	LDB #2*2Ø	'I/O ERROR'
	D618 7E AC 46	LD618	JMP LAC46	JUMP TO ERROR DRIVER
3001	D61B 34 Ø2	LD61B	PSHS A	SAVE ACCA
	D61D 96 EA		LDA DCOPC	GET OPERATION CODE
3003 3004	D61F 81 Ø3 D621 35 Ø2		CMPA #\$Ø3 PULS A	CHECK FOR WRITE SECTOR COMMAND RESTORE ACCA
	D623 26 2A		BNE LD64F	RETURN IF NOT WRITE SECTOR
	D625 7D Ø9 87		TST DVERVL	CHECK VERIFY FLAG
	D628 27 25		BEQ LD64F PSHS U,X,B,A	RETURN IF NO VERIFY
3008 3009	D62A 34 56 D62C 86 Ø2		PSHS U,X,B,A LDA #\$Ø2	SAVE REGISTERS READ OPERATION CODE
3010	D62E 97 EA		STA DCOPC	STORE TO DSKCON PARAMETER
	D63Ø DE EE		LDU DCBPT	POINT U TO WRITE BUFFER ADDRESS
	D632 8E Ø7 ØØ		LDX #DBUF1	* ADDRESS OF VERIFY BUFFER
3Ø13 3Ø14	D635 9F EE D637 8D 33		STX DCBPT BSR DSKCON	* TO DSKCON VARIABLE GO READ SECTOR
3015	D639 DF EE		STU DCBPT	RESTORE WRITE BUFFER
3Ø16	D63B 86 Ø3		LDA #\$Ø3	WRITE OP CODE
3017	D63D 97 EA		STA DCOPC	SAVE IN DSKCON VARIABLE
3Ø18 3Ø19	D63F 96 FØ D641 26 ØD		LDA DCSTA BNE LD650	CHECK STATUS FOR THE READ OPERATION BRANCH IF ERROR
3020	D643 5F		CLRB	CHECK 256 BYTES
		LD644	LDA ,X+	GET BYTE FROM WRITE BUFFER
	D646 A1 CØ		CMPA ,U+	COMPARE TO READ BUFFER
	D648 26 Ø6		BNE LD65Ø	BRANCH IF NOT EQUAL
	D64A 5A D64B 26 F7		DECB BNE LD644	* DECREMENT BYTE COUNTER AND * BRANCH IF NOT DONE
	D64D 35 56		PULS A,B,X,U	RESTORE REGISTERS
	D64F 39	LD64F	RTS	
		LD650		RESTORE REGISTERS
3029 3030	D652 7A Ø9 88 D655 26 B1		DEC ATTCTR BNE LD608	DECREMENT THE VERIFY COUNTER BRANCH IF MORE TRIES LEFT
3031	D657 C6 48		LDB #2*336	'VERIFY ERROR'
3Ø32	D659 20 BD		BRA LD618	JUMP TO ERROR HANDLER
3Ø33				
3034	Deed ee		COMMAND	OFF FLAC - A
3Ø35	D65B 5F D65C 81 AA	* VERIFY	CLRB	OFF FLAG = Ø OFF TOKEN ?
	D65B 5F D65C 81 AA D65E 27 Ø7			OFF FLAG = Ø OFF TOKEN ? YES
3035 3036 3037 3038	D65C 81 AA D65E 27 Ø7 D66Ø 53		CLRB CMPA #\$AA BEQ LD667 COMB	OFF TOKEN ? YES ON FLAG = \$FF
3035 3036 3037 3038 3039	D65C 81 AA D65E 27 Ø7 D66Ø 53	VERIFY	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN
3035 3036 3037 3038 3039 3040	D65C 81 AA D65E 27 Ø7 D66Ø 53	VERIFY	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF
3035 3036 3037 3038 3039	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88	VERIFY	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN
3035 3036 3037 3038 3039 3040 3041 3042 3043	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87	VERIFY	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG
3035 3036 3037 3038 3039 3040 3041 3042 3043 3044	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 10 26 DC 10 D667 F7 09 87 D66A ØE 9F	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC
3035 3036 3037 3038 3039 3040 3041 3042 3043	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG
3035 3036 3037 3038 3039 3040 3041 3042 3044 3045 3046 3047	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$95 PSHS A	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK
3035 3036 3037 3038 3040 3041 3042 3043 3044 3045 3046 3047 3048	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 10 26 DC 10 D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER
3035 3036 3037 3038 3039 3040 3041 3042 3043 3044 3045 3046 3047 3048 3049	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 F7 Ø9 85 D675 D6 EB	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR LDB DCDRV	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NUMBER
3035 3036 3037 3038 3040 3041 3042 3043 3044 3045 3046 3047 3048	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 10 26 DC 10 D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER
3035 3036 3037 3038 3040 3041 3042 3043 3044 3045 3046 3047 3048 3049	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR LDB DCDRV LDX #LD7AA LDA BRGRAM ANDA #\$48	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS
3035 3036 3037 3038 3039 3040 3041 3042 3043 3044 3045 3046 3047 3050 3050 3051 3052 3053	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 A8 D67F AA 85	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$95 PSHS A CLR ROYTMR LDB DCDRV LDX #LD7AA LDA DRGRAM ANDA #\$48	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NOT READY TIMER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA
3035 3036 3037 3038 3040 3041 3042 3043 3044 3045 3047 3046 3047 3049 3050 3051 3052 3053 3052	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 A8 D67F AA 85 D681 8A 2Ø	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH ROUITINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CCLR ROYTMR LDB DCDRV LDX #LD7AA LDA #\$A8 ORA B,X ORA #\$20	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DOUBLE DENSITY
3035 3036 3037 3038 3039 3040 3041 3042 3043 3044 3045 3046 3047 3050 3050 3051 3052 3053	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$95 PSHS A CLR ROYTMR LDB DCDRV LDX #LD7AA LDA DRGRAM ANDA #\$48	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NOT READY TIMER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA
3035 3036 3037 3038 3039 3041 3042 3043 3044 3045 3046 3051 3052 3053 3054 3055 3055 3055 3056	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67B 84 A8 D67A A8 5 D681 8A 2Ø D683 D6 EC D685 C1 16	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CCLR RDYTMR LDB DCDRV LDX #LD7AA LDA DAGRAM ANDA #\$A8 ORA B,X ORA #\$20 LDB DCTRK CMPB #22 BLO L068B	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENBALE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DOUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3046 3047 3050 3051 3052 3053 3055 3055 3056 3057 3058	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 48 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 25 Ø2 D689 8A 1Ø	VERIFY  LD667  * DSKCON  LD672	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR LDB DCDRV LDX #LD7AA LDA DRGRAM ANDA #\$A8 ORA #\$20 LDB DCTRK CMPB #22 BLO LD68B ORA #\$10	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DUBLED DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3047 3050 3051 3052 3053 3055 3056 3057 3056 3057 3057 3056 3057	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 C2 9C D689 BA 1Ø D688 1F 89	VERIFY  LD667  * DSKCON	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR ROYTMR LDB DCDRV LDX #LD7AA LDA DRGRAM ANDA #\$A8 ORA B,X ORA #\$20 LDB DCTRK CMPB #22 BLO LD68B ORA #\$10 TFR A,B	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DOUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3046 3047 3050 3051 3052 3053 3055 3055 3056 3057 3058	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 48 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 25 Ø2 D689 8A 1Ø	VERIFY  LD667  * DSKCON  LD672	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR LDB DCDRV LDX #LD7AA LDA DRGRAM ANDA #\$A8 ORA #\$20 LDB DCTRK CMPB #22 BLO LD68B ORA #\$10	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DUBLED DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3046 3050 3051 3052 3053 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3056 3057 3057 3057 3057 3057 3057 3057 3057	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D677 8E D7 AA D67A B6 Ø9 86 D67D 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 C2 9C D689 BA 1Ø D688 1F 89 D680 BA 0Ø D688 1F 89 D688 B7 Ø9 86 D687 B9 86 D689 B7 FF 4Ø	VERIFY  LD667  * DSKCON  LD672	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR ROYTMR LDB DCDRV LDX #LD7AA LDA DRGRAM ANDA #\$A8 ORA B,X ORA #\$20 LDB DCTRK CMPB #22 BLO LD68B ORA #\$10 OFF A,B ORA #\$98 STA DRGRAM STA DSKREG	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DOUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB 'OR' IN MOTOR ON CONTROL BIT SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL REGISTER
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3046 3047 3050 3050 3050 3055 3056 3057 3058 3057 3058 3059 3061 3062 3063	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66B 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 25 Ø2 D689 8A 1Ø D688 T8 89 D688 B7 Ø9 86 D689 B7 F7 4Ø D695 C5 Ø8	VERIFY  LD667  * DSKCON  LD672	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CCLR RDYTMR LDB DCDRV LDX #LD7AA LDA DAGRAM ANDA #\$A8 ORA #\$20 LDB DCTRK CMPB #22 LDB DCTRK CMPB #27 LD668B ORA #\$10 TTR A,B ORA #\$08 STA DSKREG BITB #508	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DOUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB 'OR' IN MOTOR ON CONTROL BIT SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL REGISTER = WERE MOTORS ALREADY ON?
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3046 3050 3051 3052 3053 3055 3055 3056 3057 3058 3059 3061 3062 3063 3063 3064	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67P A8 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D688 C1 16 D687 25 Ø2 D689 8A 1Ø D688 1F 89 D688 B7 Ø9 86 D699 26 7F F4 Ø D695 C5 Ø8 D695 P7 FF 4Ø D695 C5 Ø8 D697 26 Ø6	VERIFY  LD667  * DSKCON  LD672	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR LDB DCDRV LDX #LD7AA LDA DRGRAM ANDA #\$48 ORA #\$20 LDB DCTRK CMPB #22 BLO LD68B ORA #\$10 TFR A,B ORA #\$08 STA DGRAM STA D	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DOUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB 'OR' IN MOTOR ON CONTROL BIT SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL REGISTER = WERE MOTORS ALREADY ON? = DON'T WAIT FOR IT TO COME UP TO SPEED IF ALREADY ON
3035 3036 3037 3038 3040 3041 3042 3043 3044 3045 3046 3047 3051 3051 3052 3053 3054 3055 3055 3056 3057 3058 3059 3060 3061 3062 3063 3062 3063 3066 3066 3066 3066	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67B 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 C2 9C D685 C1 16 D687 C3 W2 D688 1F 89 D680 BA 1Ø D680 BA 1Ø D680 BA 0Ø D680 BA 0Ø D680 BA 0Ø D681 BA 0Ø D682 B7 FF 4Ø D695 C5 Ø8 D697 26 Ø6 D699 BD A7 D1	VERIFY  LD667  * DSKCON  LD672	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH ROUTINE PSHS U,Y,X,B,A LDA #\$95 PSHS A CCLR RDYTMR LDB DCDRV LDX #LD7AA LDA DARGRAM ANDA #\$A8 ORA B,X ORA #\$20 LDB DCTRK CMPB #22 BLO LD68B ORA #\$10 TFR A,B ORA #\$98 STA DRGRAM STA DSKREG BITB #\$08 BNE LD69F JSS LA7D1	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DOUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB 'OR' IN MOTOR ON CONTROL BIT SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL REGISTER = WERE MOTORS ALREADY ON? = DON'T WAIT FOR IT TO COME UP TO SPEED IF ALREADY ON * WAIT A WHILE
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3046 3050 3051 3052 3053 3055 3055 3056 3057 3058 3059 3061 3062 3063 3063 3064	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 25 Ø2 D689 8A 1Ø D688 1F 89 D680 BA Ø8 D687 F 7 Ø9 86 D699 BD A7 D1 D696 BD A7 D1 D696 BD A7 D1 D696 BD A7 D1 D696 BD A7 D1	VERIFY  LD667  * DSKCON  LD672	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR LDB DCDRV LDX #LD7AA LDA DRGRAM ANDA #\$48 ORA B,X ORA #\$20 LDB DCTRK CMPB #22 BLO LD68B ORA #510 TFR A,B ORA #\$10 TFR A,B ORA #\$08 STA DGRAM	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DOUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB 'OR' IN MOTOR ON CONTROL BIT SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL REGISTER = WERE MOTORS ALREADY ON? = DON'T WAIT FOR IT TO COME UP TO SPEED IF ALREADY ON
3035 3036 3037 3038 3049 3044 3044 3045 3046 3047 3050 3051 3052 3053 3054 3055 3056 3057 3058 3059 3050 3061 3062 3063 3066 3066 3066 3066 3066	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66B 86 Ø5 D670 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 25 Ø2 D689 BA 1Ø D68B 1F 89 D68D BA Ø8 D68B 1F 89 D68B B7 Ø9 86 D692 B7 FF 4Ø D695 C5 Ø8 D697 B0 A7 D1 D697 BD A7 D697 BD	LD667  * DSKCON  LD672  LD68B	CLRB CMPA #\$AA BEQ LD667 COMB CMPA #\$88 LBNE LB277 STB DVERVL JMP GETNCH ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CCLR ROYTMR LDB DCDRV LDX #LD7AA LDA DAGRAM ANDA #\$A8 ORA B,X ORA #\$20 LDB DCTRK CMPB #22 BLO LD68B ORA #\$10 TFR A,B ORA #\$48 STA DRGRAM STA DRGRAM STA DKGRAM STA	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DOUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB 'OR' IN MOTOR ON CONTROL BIT SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL REGISTER = WERE MOTORS ALREADY ON? = DON'T WAIT FOR IT TO COME UP TO SPEED IF ALREADY ON * WAIT A WHILE * WAIT SOME MORE FOR MOTOR TO COME UP TO SPEED WAIT UNTIL NOT BUSY OR TIME OUT BRANCH IF TIMED OUT (DOOR OPEN. NO DISK, NO POWER. ETC.)
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3046 3050 3051 3052 3053 3055 3056 3056 3056 3056 3066 3067 3066 3067 3066 3067 3066 3067 3066	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66E 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D67D 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 C5 Ø2 D689 8A 1Ø D688 B7 89 D689 BA 7Ø D689 B7 89 D699 B7 F7 40 D699 B0 A7 D1 D696 B0 A7 D1 D696 B0 A7 D1 D696 B0 A7 D1 D696 B0 A7 D1 D697 B0 A7 D64A 26 ØA D64A ØF FØ	LD667  * DSKCON  LD672  LD68B	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH  ROUITINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR LDB DCDRV LDX #LD7AA LDA DAGRAM ANDA #\$A8 ORA #\$20 LDB DCTRK CMPB #22 BLO L0668B ORA #\$10 TFR A,B ORA #\$50 ORA #	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DRIVE SELECT DATA 'OR' IN DUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB 'OR' IN MOTOR ON CONTROL BIT SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL REGISTER = WERE MOTORS ALREADY ON? = DON'T WAIT FOR IT TO COME UP TO SPEED IF ALREADY ON * WAIT A WHILE * WAIT SOME MORE FOR MOTOR TO COME UP TO SPEED WAIT UNTIL NOT BUSY OR TIME OUT BRANCH IF TIMED OUT (DOOR OPEN. NO DISK, NO POWER. ETC.) CLEAR STATUS REGISTER
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3047 3050 3051 3052 3053 3055 3056 3057 3056 3057 3056 3061 3062 3063 3064 3063 3064 3065 3066 3067 3068 3068 3068 3068 3068 3068 3068 3068	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66B 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D679 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 25 Ø2 D689 8A 1Ø D688 B7 89 D680 BA Ø8 D687 B7 Ø9 86 D699 BD A7 D1 D69C BD A7 D69F 8D 3D D641 26 ØA D643 QF FØ D643 D D64 D699 BD A7 D1 D69C BD A7 D69F BD 3D D641 26 ØA D643 ØF FØ	LD667  * DSKCON  LD672  LD68B	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH N ROUTINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR LDB DCDRV LDX #LD7AA LDA DRGRAM ANDA #\$48 ORA #\$20 LDB DCTRK CMPB #22 BLO LD68B ORA #\$10 TFR A,B ORA #\$10 TFR A,B ORA #\$88 STA DGRAM STA DGRA	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DOUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON MRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB 'OR' IN MOTOR ON CONTROL BIT SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL BETS SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL BETS = WERE MOTORS ALREADY ON? = DON'T WAIT FOR IT TO COME UP TO SPEED IF ALREADY ON * WALT A WHILE * WAIT SOME MORE FOR MOTOR TO COME UP TO SPEED WAIT UNTIL NOT BUSY OR TIME OUT BRANCH IF TIMED OUT (DOOR OPEN. NO DISK, NO POWER. ETC.) CLEAR STATUS REGISTER POINT TO COMMAND JUMP VECTORS
3035 3036 3037 3038 3049 3041 3042 3043 3044 3045 3046 3050 3051 3052 3053 3055 3056 3056 3056 3056 3066 3067 3066 3067 3066 3067 3066 3067 3066	D65C 81 AA D65E 27 Ø7 D66Ø 53 D661 81 88 D663 1Ø 26 DC 1Ø D667 F7 Ø9 87 D66A ØE 9F  D66C 34 76 D66B 86 Ø5 D67Ø 34 Ø2 D672 7F Ø9 85 D675 D6 EB D677 8E D7 AA D67A B6 Ø9 86 D679 84 A8 D67F AA 85 D681 8A 2Ø D683 D6 EC D685 C1 16 D687 25 Ø2 D689 8A 1Ø D688 B7 89 D680 BA Ø8 D687 B7 Ø9 86 D699 BD A7 D1 D69C BD A7 D69F 8D 3D D641 26 ØA D643 QF FØ D643 D D64 D699 BD A7 D1 D69C BD A7 D69F BD 3D D641 26 ØA D643 ØF FØ	LD667  * DSKCON  LD672  LD68B	CLRB CMPA #\$AA BEQ L0667 COMB CMPA #\$88 LBNE L8277 STB DVERVL JMP GETNCH  ROUITINE PSHS U,Y,X,B,A LDA #\$05 PSHS A CLR RDYTMR LDB DCDRV LDX #LD7AA LDA DAGRAM ANDA #\$A8 ORA #\$20 LDB DCTRK CMPB #22 BLO L0668B ORA #\$10 TFR A,B ORA #\$50 ORA #	OFF TOKEN ? YES ON FLAG = \$FF ON TOKEN BRANCH TO 'SYNTAX ERROR' IF NOT ON OR OFF SET VERIFY FLAG GET NEXT CHARACTER FROM BASIC  SAVE REGISTERS * GET RETRY COUNT AND * SAVE IT ON THE STACK RESET DRIVE NOT READY TIMER GET DRIVE NUMBER POINT X TO DRIVE ENABLE MASKS GET DSKREG IMAGE KEEP MOTOR STATUS, DOUBLE DENSITY. HALT ENABLE 'OR' IN DRIVE SELECT DATA 'OR' IN DRIVE SELECT DATA 'OR' IN DUBLE DENSITY GET TRACK NUMBER PRECOMPENSATION STARTS AT TRACK 22 BRANCH IF LESS THAN 22 TURN ON WRITE PRECOMPENSATION IF >= 22 SAVE PARTIAL IMAGE IN ACCB 'OR' IN MOTOR ON CONTROL BIT SAVE IMAGE IN RAM PROGRAM THE 1793 CONTROL REGISTER = WERE MOTORS ALREADY ON? = DON'T WAIT FOR IT TO COME UP TO SPEED IF ALREADY ON * WAIT A WHILE * WAIT SOME MORE FOR MOTOR TO COME UP TO SPEED WAIT UNTIL NOT BUSY OR TIME OUT BRANCH IF TIMED OUT (DOOR OPEN. NO DISK, NO POWER. ETC.) CLEAR STATUS REGISTER

3Ø73	D6AB AD 95		JSR	[B,X]	GO DO IT
3074	D6AD 35 Ø2	LD6AD	PULS		GET RETRY COUNT
3Ø75 3Ø76	D6AF D6 FØ D6B1 27 ØB		LDB BEQ	DCSTA LD6BE	GET STATUS BRANCH IF NO ERRORS
3Ø77	D6B3 4A		DECA		DECREMENT RETRIES COUNTER
3078	D6B4 27 Ø8			LD6BE	BRANCH IF NO RETRIES LEFT
3Ø79 3Ø8Ø	D6B6 34 Ø2 D6B8 8D ØB		PSHS BSR		SAVE RETRY COUNT ON STACK RESTORE HEAD TO TRACK Ø
3081	D6BA 26 F1		BNE	LD6AD	BRANCH IF SEEK ERROR
3Ø82	D6BC 20 B4		BRA	LD672	GO TRY COMMAND AGAIN IF NO ERROR
3Ø83 3Ø84	D6BE 86 78 D6CØ B7 Ø9 85	LD6BE	LDA STA	#120 RDYTMR	120*1/60 = 2 SECONDS (1/60 SECOND FOR EACH IRQ INTERRUPT) WAIT 2 SECONDS BEFORE TURNING OFF MOTOR
3085	D6C3 35 F6			A,B,X,Y,U,PC	RESTORE REGISTERS - EXIT DSKCON
3Ø86			RE HEA	ND TO TRACK Ø	
3Ø87 3Ø88	D6C5 8E Ø9 7E D6C8 D6 EB	LD6C5	LDX LDB	#DRØTRK DCDRV	POINT TO TRACK TABLE GET DRIVE NUMBER
3Ø89	D6CA 6F 85		CLR		ZERO TRACK NUMBER
3Ø9Ø	D6CC 86 Ø3		LDA	#\$Ø3	* RESTORE HEAD TO TRACK Ø, UNLOAD THE HEAD
3Ø91 3Ø92	D6CE B7 FF 48 D6D1 1E 88		STA EXG	FDCREG A,A	* AT START, 30 MS STEPPING RATE
3093	D6D3 1E 88			A,A	= WAIT FOR 1793 TO RESPOND TO COMMAND
3Ø94	D6D5 8D Ø7		BSR	LD6DE	WAIT TILL DRIVE NOT BUSY
3Ø95 3Ø96	D6D7 8D 24 D6D9 84 10			LD6FD #\$10	WAIT SOME MORE 1793 STATUS : KEEP ONLY SEEK ERROR
3097	D6DB 97 FØ		STA	DCSTA	SAVE IN DSKCON STATUS
3Ø98	D6DD 39	LD6DD	RTS		
3Ø99 31ØØ				IE 1793 TO BECOME UNBUSY. IF IT DOI ITERRUPT AND ISSUE A DRIVE NOT REA	
3101	D6DE 9E 8A	LD6DE		ZERO	GET ZERO TO X REGISTER - LONG WAIT
3102	D6EØ 3Ø 1F	LD6EØ		-1,X	DECREMENT LONG WAIT COUNTER
31Ø3 31Ø4	D6E2 27 Ø8 D6E4 B6 FF 48		LDA	LD6EC FDCREG	1F NOT READY BY NOW, FORCE INTERRUPT * GET 1793 STATUS AND TEST
3105	D6E7 85 Ø1			#\$01	* BUSY STATUS BIT
3106	D6E9 26 F5		BNE	LD6EØ	BRANCH IF BUSY
31Ø7 31Ø8	D6EB 39 D6EC 86 DØ	LD6EC	RTS LDA	#\$DØ	* FORCE INTERRUPT COMMAND - TERMINATE ANY COMMAND
3100	D6EE B7 FF 48	LDOLG	STA	FDCREG	* IN PROCESS. DO NOT GENERATE A 1793 INTERRUPT REQUEST
3110	D6F1 1E 88		EXG	A,A	* WAIT BEFORE READING 1793
3111 3112	D6F3 1E 88 D6F5 B6 FF 48		EXG LDA	A,A FDCREG	* RESET INTRQ (FDC INTERRUPT REQUEST)
3113	D6F8 86 8Ø		LDA	#\$80	RETURN DRIVE NOT READY STATUS IF THE DRIVE DID NOT BECOME UNBUSY
3114	D6FA 97 FØ		STA	DCSTA	SAVE DSKCON STATUS BYTE
3115 3116	D6FC 39	* MEDIU	RTS M DELA	v	
3117	D6FD 8E 22 2E	LD6FD		#8750	DELAY FOR A WHILE
3118	D700 30 1F	LD700		-1,X	* DECREMENT DELAY COUNTER AND
3119 3120	D702 26 FC D704 39		BNE RTS	LD700	* BRANCH IF NOT DONE
3121	5754 33	* READ		CTOR	
3122	D7Ø5 86 8Ø	LD7Ø5		#\$80	\$80 IS READ FLAG (1793 READ SECTOR)
3123 3124	D7Ø7 8C	LD707 * WRITE		#\$86AØ	SKIP TWO BYTES
3125	D7Ø8 86 AØ	LD7Ø8		#\$AØ	\$AØ IS WRITE FLAG (1793 WRITE SECTOR)
3126	D7ØA 34 Ø2		PSHS		SAVE READ/WRITE FLAG ON STACK
3127 3128	D70C 8E 09 7E D70F D6 EB		LDX	#DRØTRK DCDRV	POINT X TO TRACK NUMBER TABLE IN RAM GET DRIVE NUMBER
3129	D711 3A		ABX	505	POINT X TO CORRECT DRIVE'S TRACK BYTE
3130	D712 E6 84		LDB	,X	GET TRACK NUMBER OF CURRENT HEAD POSITION
3131 3132	D714 F7 FF 49 D717 D1 EC			FDCREG+1 DCTRK	SEND TO 1793 TRACK REGISTER COMPARE TO DESIRED TRACK
3133	D719 27 1E		BEQ		BRANCH IF ON CORRECT TRACK
3134	D71B 96 EC		LDA	DCTRK	GET TRACK DESIRED
3135 3136	D71D B7 FF 4B D72Ø A7 84		STA STA	FDCREG+3	SEND TO 1793 DATA REGISTER SAVE IN RAM TRACK IMAGE
3137	D722 86 17		LDA	#\$17	* SEEK COMMAND FOR 1793: DO NOT LOAD THE
3138 3139	D724 B7 FF 48 D727 1E 88		STA EXG	FDCREG A,A	* HEAD AT START, VERIFY DESTINATION TRACK, * 30 MS STEPPING RATE - WAIT FOR
3140	D727 1E 88		EXG	A,A	* VALID STATUS FROM 1793
3141	D72B 8D B1		BSR	LD6DE	WAIT TILL NOT BUSY
3142 3143	D72D 26 Ø8 D72F 8D CC		BNE BSR	LD737 LD6FD	RETURN IF TIMED OUT WAIT SOME MORE
3143	D731 84 18			#\$18	KEEP ONLY SEEK ERROR OR CRC ERROR IN ID FIELD
3145	D733 27 Ø4		BEQ	LD739	BRANCH IF NO ERRORS - HEAD ON CORRECT TRACK
3146 3147	D735 97 FØ D737 35 82	LD737		DCSTA A,PC	SAVE IN DSKCON STATUS
3148	D737 33 02			ONED ON CORRECT TRACK	
3149	D739 96 ED	LD739	LDA	DSEC	GET SECTOR NUMBER DESIRED
315Ø 3151	D73B B7 FF 4A D73E 8E D7 98		STA LDX	FDCREG+2 #LD798	SEND TO 1793 SECTOR REGISTER * POINT X TO ROUTINE TO BE VECTORED
3152	D741 BF Ø9 83		STX	DNMISV	* TO BY NMI UPON COMPLETION OF DISK I/O AND SAVE VECTOR
3153	D744 9E EE		LDX	DCBPT	POINT X TO I/O BUFFER
3154 3155	D746 B6 FF 48 D749 B6 Ø9 86		LDA LDA	FDCREG DRGRAM	RESET INTRQ (FDC INTERRUPT REQUEST) GET DSKREG IMAGE
3156	D74C 8A 8Ø		ORA	#\$80	SET FLAG TO ENABLE 1793 TO HALT 6809
3157	D74E 35 Ø4		PULS		GET READ/WRITE COMMAND FROM STACK
3158 3159	D750 10 9E 8A D753 CE FF 48		LDY LDU	ZERO #FDCREG	ZERO OUT Y - TIMEOUT INITIAL VALUE U POINTS TO 1793 INTERFACE REGISTERS
3160	D756 73 Ø9 82		COM		NMI FLAG = \$FF: ENABLE NMI VECTOR
3161	D759 1A 5Ø			#\$50	DISABLE FIRQ, IRQ
3162 3163	D75B F7 FF 48 D75E 1E 88		STB EXG	FDCREG A,A	* SEND READ/WRITE COMMAND TO 1793: SINGLE RECORD, COMPARE * FOR SIDE Ø, NO 15 MS DELAY, DISABLE SIDE SELECT
3164	D760 1E 88		EXG	A , A	* COMPARE, WRITE DATA ADDRESS MARK (FB) - WAIT FOR STATUS
3165	D762 C1 8Ø			#\$80	WAS THIS A READ?
3166 3167	D764 27 1C	* WAIT	BEQ FOR TH	LD782 IE 1793 TO ACKNOWLEDGE READY TO WR:	IF SO, GO LOOK FOR DATA ITE DATA
3168	D766 C6 Ø2			#\$Ø2	DRQ MASK BIT

## DISK BASIC UNRAVELLED II APPENDIX C ORIGIN:SPECTRAL ASSOC DISASSEMBLY OF DISK BASIC 1.0 REVISED:12/26/1999 WALTER K ZYDHEK

3169			
	D768 E5 C4	LD768 BITB ,U	IS 1793 READY FOR A BYTE? (DRQ SET IN STATUS BYTE)
3170	D76A 26 ØC	BNE LD778	BRANCH IF SO
3171	D76C 31 3F	LEAY -1,Y	DECREMENT WAIT TIMER
3172	D76E 26 F8	BNE LD768	KEEP WAITING FOR THE 1793 DRQ
3173	D77Ø 7F Ø9 82	LD770 CLR NMIFLG	RESET NMI FLAG
3174	D773 1C AF	ANDCC #\$AF	ENABLE FIRQ, IRQ
3175	D775 7E D6 EC	JMP LD6EC	FORCE INTERRUPT, SET DRIVE NOT READY ERROR
3176			
3177		* WRITE A SECTOR	
3178	D778 E6 8Ø	LD778 LDB ,X+	GET A BYTE FROM RAM
3179			
	D77A F7 FF 4B	STB FDCREG+3	SEND IT TO 1793 DATA REGISTER
3180	D77D B7 FF 40	STA DSKREG	REPROGRAM FDC CONTROL REGISTER
3181	D78Ø 2Ø F6	BRA LD778	SEND MORE DATA
3182		* WAIT FOR THE 17933 TO ACKNOWLEDGE READY T	
	5700 06 00		
3183	D782 C6 Ø2	LD782 LDB #\$Ø2	DRQ MASK BIT
3184	D784 E5 C4	LD784 BITB ,U	DOES THE 1793 HAVE A BYTE? (DRQ SET IN STATUS BYTE)
3185	D786 26 Ø6	BNE LD78E	YES, GO READ A SECTOR
3186	D788 31 3F	LEAY -1,Y	DECREMENT WAIT TIMER
3187	D78A 26 F8	BNE LD784	KEEP WAITING FOR 1793 DRQ
3188	D78C 2Ø E2	BRA LD77Ø	GENERATE DRIVE NOT READY ERROR
3189			
3190		* READ A SECTOR	
3191	D78E F6 FF 4B	LD78E LDB FDCREG+3	GET DATA BYTE FROM 1793 DATA REGISTER
3192	D791 E7 8Ø	STB ,X+	PUT IT IN RAM
3193	D793 B7 FF 40	STA DSKREG	REPROGRAM FDC CONTROL REGISTER
3194	D796 20 F6	BRA LD78E	KEEP GETTING DATA
	D750 ED 10		
3195		* BRANCH HERE ON COMPLETION OF SECTOR READ/	
3196	D798 1C AF	LD798 ANDCC #\$AF	ENABLE IRQ, FIRO
3197	D79A B6 FF 48	LDA FDCREG	* GET STATUS & KEEP WRITE PROTECT, RECORD TYPE/WRITE
3198	D79D 84 7C	ANDA #\$7C	* FAULT, RECORD NOT FOUND, CRC ERROR OR LOST DATA
			SAVE IN DSKCON STATUS
3199	D79F 97 FØ	STA DCSTA	SAVE IN DOCCOM STATUS
3200	D7A1 39	RTS	
3201			
3202		* DSKCON OPERATION CODE JUMP VECTORS	
3203	D7A2 D6 C5	LD7A2 FDB LD6C5	RESTORE HEAD TO TRACK ZERO
3204	D7A4 D6 DD	FDB LD6DD	NO OP - RETURN
3205	D7A6 D7 Ø5	FDB LD7Ø5	READ SECTOR
3206	D7A8 D7 Ø8	FDB \$D7Ø8	WRITE SECTOR
3207	57710 57 50	155 45750	MATTE SECTION
		+ BOURSO HACKS FOR BIOK BRIVE OF FOT	
3208		* DSKREG MASKS FOR DISK DRIVE SELECT	
3209	D7AA Ø1	LD7AA FCB 1	DRIVE SEL Ø
3210	D7AB Ø2	FDB 2	DRIVE SEL 1
3211	D7AC Ø4	FDB 4	DRIVE SEL 2
3212	D7AD 4Ø	FCB \$4Ø	DRIVE SEL 3
3213			
3213 3214		* NMI SERVICE	
3214		* NMI SERVICE	GFT NMT FLAG
3214 3215	D7AE B6 Ø9 82	* NMI SERVICE DNMISV LDA NMIFLG	GET NMI FLAG
3214 3215 3216	D7AE B6 Ø9 82 D7B1 27 Ø8	* NMI SERVICE DNMISV LDA NMIFLG BEQ LD7BB	RETURN IF NOT ACTIVE
3214 3215	D7AE B6 Ø9 82	* NMI SERVICE DNMISV LDA NMIFLG	
3214 3215 3216	D7AE B6 Ø9 82 D7B1 27 Ø8	* NMI SERVICE DNMISV LDA NMIFLG BEQ LD7BB	RETURN IF NOT ACTIVE
3214 3215 3216 3217 3218	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD78B  LDX DNMISV  STX 10,S	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK
3214 3215 3216 3217 3218 3219	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82	* NMI SERVICE DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR
3214 3215 3216 3217 3218 3219 3220	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD78B  LDX DNMISV  STX 10,S	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK
3214 3215 3216 3217 3218 3219 3220 3221	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD7BB  LDX DNMISV  STX 10,S  CLR NMIFLG  LD7BB RTI	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK
3214 3215 3216 3217 3218 3219 3220	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82	* NMI SERVICE DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK
3214 3215 3216 3217 3218 3219 3220 3221 3222	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B	* NMI SERVICE DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B	* NMI SERVICE DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIAØ+3	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG 63.5 MICRO SECOND OR 60 HZ INTERRUPT?
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82 D7BB 3B	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD7BB  LDX DNMISV  STX 10,S  CLR NMIFLG  LD7BB RTI  * IRQ SERVICE  DIRQSV LDA PIAØ+3  BPL LD7BB	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B	* NMI SERVICE  DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIA0+3 BPL LD7BB LDA PIA0+2	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82 D7BB 3B	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD7BB  LDX DNMISV  STX 10,S  CLR NMIFLG  LD7BB RTI  * IRQ SERVICE  DIRQSV LDA PIAØ+3  BPL LD7BB	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 69 82 D7BB 3B D7BC B6 FF 03 D7BC B6 FF 03 D7BC B6 FF 02 D7C4 B6 09 85	* NMI SERVICE DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIA0+3 BPL LD7BB LDA PIA0+2 LDA RDYTMR	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82 D7BB 3B  D7BC B6 FF Ø3 D7BF 2A FA D7C1 B6 FF Ø2 D7C4 B6 Ø9 85 D7C7 27 11	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD7BB  LDX DNMISV  STX 10,S  CLR NMIFLG  LD7BB RTI  * IRQ SERVICE  DIRQSV LDA PIAØ+3  BPL LD7BB  LDA PIAØ+2  LDA RDYTMR  BEQ LD7DA	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A	* NMI SERVICE  DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIAØ+3 BPL LD7BB LDA PIAØ+2 LDA RDYTMR BEQ LD7DA DECA	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A D7CA B7 09 85	* NMI SERVICE DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIA0+3 BPL LD7BB LDA PIA0+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A	* NMI SERVICE  DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIAØ+3 BPL LD7BB LDA PIAØ+2 LDA RDYTMR BEQ LD7DA DECA	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A D7CA B7 09 85	* NMI SERVICE DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIA0+3 BPL LD7BB LDA PIA0+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A D7CA B7 09 85 D7CD 26 0B D7CF B6 09 86	* NMI SERVICE  DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG  LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIAØ+3 BPL LD7BB LDA PIAØ+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR BNE LD7DA LDA DRGRAM	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE
3214 3215 3216 3217 3218 3219 3221 3222 3223 3224 3225 3226 3227 3228 3229 3231 3231	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82 D7BB 3B  D7BC B6 FF Ø3 D7BF 2A FA D7C1 B6 FF Ø2 D7C4 B6 Ø9 85 D7C7 27 11 D7C9 4A D7CA B7 Ø9 85 D7CA B6 Ø8 D7CA B6 Ø9 86 D7CA B6 Ø9 86 D7CB B6 Ø9 86 D7CB B6 Ø9 86	* NMI SERVICE  DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIA0+3 BPL LD7BB LDA PIA0+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR BNE LD7DA LDA DRGRAM ANDA #\$B0	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3232	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82 D7BB 3B  D7BC B6 FF Ø3 D7BF 2A FA D7C1 B6 FF Ø2 D7C4 B6 Ø9 85 D7C7 27 11 D7C9 4A D7CA B7 Ø9 85 D7CD 26 ØB D7CF B6 Ø9 86 D7CD 84 B0 D7CF B6 Ø9 86 D7CD 84 B0 D7CP 86 Ø9 86 D7CD 84 B0 D7CP 86 Ø9 86 D7CD 84 B0 D7CP 86 Ø9 86	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG  LD7BB RTI  * IRQ SERVICE  DIRQSV LDA PIAØ+3 BPL LD7BB LDA PIAØ+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR BNE LD7DA LDA DRGRAM ANDA #\$BØ STA DRGRAM	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF = PUT IT BACK IN RAM IMAGE
3214 3215 3216 3217 3218 3219 3221 3222 3223 3224 3225 3226 3227 3228 3229 3231 3231	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82 D7BB 3B  D7BC B6 FF Ø3 D7BF 2A FA D7C1 B6 FF Ø2 D7C4 B6 Ø9 85 D7C7 27 11 D7C9 4A D7CA B7 Ø9 85 D7CA B6 Ø8 D7CA B6 Ø9 86 D7CA B6 Ø9 86 D7CB B6 Ø9 86 D7CB B6 Ø9 86	* NMI SERVICE  DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIA0+3 BPL LD7BB LDA PIA0+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR BNE LD7DA LDA DRGRAM ANDA #\$B0	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF
3214 3215 3216 3217 3218 3219 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3232	D7AE B6 Ø9 82 D7B1 27 Ø8 D7B3 BE Ø9 83 D7B6 AF 6A D7B8 7F Ø9 82 D7BB 3B  D7BC B6 FF Ø3 D7BF 2A FA D7C1 B6 FF Ø2 D7C4 B6 Ø9 85 D7C7 27 11 D7C9 4A D7CA B7 Ø9 85 D7CD 26 ØB D7CF B6 Ø9 86 D7CD 84 B0 D7CF B6 Ø9 86 D7CD 84 B0 D7CP 86 Ø9 86 D7CD 84 B0 D7CP 86 Ø9 86 D7CD 84 B0 D7CP 86 Ø9 86	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG  LD7BB RTI  * IRQ SERVICE  DIRQSV LDA PIAØ+3 BPL LD7BB LDA PIAØ+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR BNE LD7DA LDA DRGRAM ANDA #\$BØ STA DRGRAM	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF = PUT IT BACK IN RAM IMAGE
3214 3215 3216 3217 3218 3229 3221 3222 3223 3224 3225 3226 3227 3228 3229 3230 3231 3232 3233 3234 3235	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A D7CA B7 09 85 D7CF B6 09 86 D7CF B6 09 86 D7DZ 84 B0 D7DZ 84 B0 D7DZ 84 B0 D7DZ 87 FF 40	* NMI SERVICE  DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG  LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIAØ+3 BPL LD7BB LDA PIAØ+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR BNE LD7DA LDA DRGRAM ANDA #\$BØ STA DSKREG	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF = PUT IT BACK IN RAM IMAGE SEND TO CONTROL REGISTER (MOTORS OFF)
3214 3215 3216 3217 3218 3220 3221 3222 3223 3224 3225 3226 3227 3228 3230 3231 3232 3233 3234 3235 3235 3236	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A D7CA B7 09 85 D7CF B6 09 86 D7CF B6 09 86 D7DZ 84 B0 D7DZ 84 B0 D7DZ 84 B0 D7DZ 87 FF 40	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG  LD7BB RTI  * IRQ SERVICE  DIRQSV LDA PIAØ+3 BPL LD7BB LDA PIAØ+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR BNE LD7DA LDA DRGRAM ANDA #\$BØ STA DRGRAM	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF = PUT IT BACK IN RAM IMAGE SEND TO CONTROL REGISTER (MOTORS OFF)
3214 3215 3216 3217 3218 3229 3221 3222 3223 3224 3225 3226 3227 3228 3229 3231 3232 3231 3232 3233 3234 3235 3236	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A D7CA B7 09 85 D7CF B6 09 86 D7CF B6 09 86 D7DZ 84 B0 D7DZ 84 B0 D7DZ 84 B0 D7DZ 87 FF 40	* NMI SERVICE  DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG  LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIAØ+3 BPL LD7BB LDA PIAØ+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR BNE LD7DA LDA DRGRAM ANDA #\$BØ STA DRGRAM ANDA #\$BØ STA DRGRAM STA DSKREG LD7DA JMP L8955  * THIS IS THE END OF DISK BASIC.	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF = PUT IT BACK IN RAM IMAGE SEND TO CONTROL REGISTER (MOTORS OFF) JUMP TO EXTENDED BASIC'S IRQ HANDLER
3214 3215 3216 3217 3218 3229 3221 3222 3223 3224 3225 3226 3227 3228 3231 3232 3231 3232 3233 3234 3235 3236 3237 3238	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A D7CA B7 09 85 D7CF B6 09 86 D7CF B6 09 86 D7DZ 84 B0 D7DZ 84 B0 D7DZ 84 B0 D7DZ 87 FF 40	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD7BB  LDX DNMISV  STX 10,S  CLR NMIFLG  LD7BB RTI  * IRQ SERVICE  DIRQSV LDA P1A0+3  BPL LD7BB  LDA P1A0+2  LDA RDYTMR  BEQ LD7DA  DECA  STA ROYTMR  BNE LD7DA  LDA DRGRAM  ANDA #\$BØ  STA DRGRAM  STA DRGRAM  STA DSKREG  LD7DA JMP LB955  * THIS IS THE END OF DISK BASIC.  * THE CODE FROM THIS POINT TO \$DFFF IS GARE	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF = PUT IT BACK IN RAM IMAGE SEND TO CONTROL REGISTER (MOTORS OFF) JUMP TO EXTENDED BASIC'S IRQ HANDLER
3214 3215 3216 3217 3218 3229 3221 3222 3223 3224 3225 3226 3227 3228 3229 3231 3232 3231 3232 3233 3234 3235 3236	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A D7CA B7 09 85 D7CF B6 09 86 D7CF B6 09 86 D7DZ 84 B0 D7DZ 84 B0 D7DZ 84 B0 D7DZ 87 FF 40	* NMI SERVICE  DNMISV LDA NMIFLG BEQ LD7BB LDX DNMISV STX 10,S CLR NMIFLG  LD7BB RTI  * IRQ SERVICE DIRQSV LDA PIAØ+3 BPL LD7BB LDA PIAØ+2 LDA RDYTMR BEQ LD7DA DECA STA RDYTMR BNE LD7DA LDA DRGRAM ANDA #\$BØ STA DRGRAM ANDA #\$BØ STA DRGRAM STA DSKREG LD7DA JMP L8955  * THIS IS THE END OF DISK BASIC.	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF = PUT IT BACK IN RAM IMAGE SEND TO CONTROL REGISTER (MOTORS OFF) JUMP TO EXTENDED BASIC'S IRQ HANDLER
3214 3215 3216 3217 3218 3229 3221 3222 3223 3224 3225 3226 3227 3228 3231 3232 3231 3232 3233 3234 3235 3236 3237 3238	D7AE B6 09 82 D7B1 27 08 D7B3 BE 09 83 D7B6 AF 6A D7B8 7F 09 82 D7BB 3B  D7BC B6 FF 03 D7BF 2A FA D7C1 B6 FF 02 D7C4 B6 09 85 D7C7 27 11 D7C9 4A D7CA B7 09 85 D7CF B6 09 86 D7CF B6 09 86 D7DZ 84 B0 D7DZ 84 B0 D7DZ 84 B0 D7DZ 87 FF 40	* NMI SERVICE  DNMISV LDA NMIFLG  BEQ LD7BB  LDX DNMISV  STX 10,S  CLR NMIFLG  LD7BB RTI  * IRQ SERVICE  DIRQSV LDA P1A0+3  BPL LD7BB  LDA P1A0+2  LDA RDYTMR  BEQ LD7DA  DECA  STA ROYTMR  BNE LD7DA  LDA DRGRAM  ANDA #\$BØ  STA DRGRAM  STA DRGRAM  STA DSKREG  LD7DA JMP LB955  * THIS IS THE END OF DISK BASIC.  * THE CODE FROM THIS POINT TO \$DFFF IS GARE	RETURN IF NOT ACTIVE GET NEW RETURN VECTOR STORE AT STACKED PC SLOT ON STACK RESET NMI FLAG  63.5 MICRO SECOND OR 60 HZ INTERRUPT? RETURN IF 63.5 MICROSECOND RESET 60 HZ PIA INTERRUPT FLAG GET TIMER BRANCH IF NOT ACTIVE DECREMENT THE TIMER SAVE IT BRANCH IF NOT TIME TO TURN OFF DISK MOTORS = GET DSKREG IMAGE = TURN ALL MOTORS AND DRIVE SELECTS OFF = PUT IT BACK IN RAM IMAGE SEND TO CONTROL REGISTER (MOTORS OFF) JUMP TO EXTENDED BASIC'S IRQ HANDLER

ALLCOL	ØØB5	DCTRK	ØØEC	DVEC2Ø	C8BØ	IRQ	FFF8	LAD33	AD33
ANGLE	ØØE8	DEBVAL	Ø11B	DVEC22	C2B2	IRQVEC	Ø1ØC	LAD9E	AD9E
ARYDIS	0008	DEFDRV	Ø95A	DVEC3	CC1C	KEYBUF	Ø152	LADC6	ADC6
ARYEND	ØØ1F	DEFEXT	C2A9	DVEC4	C5BC	KILL	C6EF	LADD4	ADD4
ARYTAB	ØØ1D	DEVCFW	ØØ6A	DVEC5	C848	L813C	813C	LADEB	ADEB
ATTCTR	Ø988	DEVLCF	ØØ6B	DVEC6	C84B	L8168	8168	LAE15	AE15
BACKUP	D262	DEVNUM	ØØ6F	DXCVEC	CFØA	L8311	8311	LAF9A	AF9A
BAKCOL	ØØB3	DEVNOM	ØØ6C	DXIVEC	CF32	L8311	8316		AFA4
								LAFA4	
BASEXT	C2A6	DEVWID	ØØ6D	ENDCHR	0002	L836C	836C	LAFB1	AFB1
BAWMST	AØE8	DEXTBF	Ø954	ENDFLG	0000	L8C1B	8C1B	LASTPT	ØØØD
BEGGRP	ØØBA	DFFLEN	Ø97C	ENDGRP	ØØB7	L95AC	95AC	LBØØC	BØØC
BINEXT	C2AF	DFLBUF	Ø989	EVALEXPB		L962E	962E	LBØ1E	BØ1E
BINVAL	ØØ2B	DFLTYP	Ø957	EXECJP	ØØ9D	L965Ø	9650	LB148	B148
BLKCNT	0094	DIMFLG	ØØØ5	EXPJMP	Ø11D	L96CB	96CB	LB156	B156
BLKLEN	ØØ7D	DIR	CCA9	FATBLØ	Ø8ØØ	L96EC	96EC	LB166	B166
BLKTYP	ØØ7C	DIRQSV	D8AF	FATBL1	Ø84A	L975F	975F	LB244	B244
BOTSTK	0017	DKWMST	CØE7	FATBL2	Ø894	L9FB5	9FB5	LB262	B262
BROMHK	AA1A	DLBAUD	ØØE6	FATBL3	Ø8DE	LAØE2	AØE2	LB26F	B26F
CASBUF	Ø1DA	DLODFL	Ø95D	FCBACT	Ø95B	LA171	A171	LB277	B277
CASFLG	Ø11A	DMRGFL	Ø95E	FCBADR	Ø94A	LA176	A176	LB2CE	B2CE
CBTPHA	ØØ84	DNAMBF	Ø94C	FCBTMP	ØØF1	LA282	A282	LB357	B357
CBUFAD	ØØ7E	DNMISV	D8A1	FCBV1	Ø928	LA35F	A35F	LB3E6	B3E6
CCKSUM	0080	DNMIVC	Ø983	FDCREG	FF48	LA37C	A37C	LB44A	B44A
CFNBUF	Ø1D1	DOS	D6EC	FIELD	DØBC	LA3ED	A3ED	LB4F3	B4F3
CHARAC	0001	DOSBAS	CØØØ	FILES	D15C	LA3FB	A3FB	LB5ØF	B5ØF
CHARAD	ØØA6	DOSBUF	2600	FILSTA	ØØ78	LA4Ø6	A4Ø6	LB516	B516
CHGFLG	ØØDB	DOSCOM	DFØØ	FIRQ	FFF6	LA400	A426	LB654	B654
CINBFL	0070	DOSINI	DF4C	FORCOL	ØØB2	LA429	A429		B657
								LB657	
CINCTR	ØØ79	DOSVEC	CØØA	FPØEXP	ØØ4F	LA42D	A42D	LB659	B659
CINPTR	ØØ7A	DOTVAL	ØØE5	FPØSGN	0054	LA549	A549	LB69B	B69B
CLSTSN	ØØ85	DRØTRK	Ø97E	FP1EXP	ØØ5C	LA59A	A59A	LB6A4	B6A4
CMPØ	0090	DRESFL	Ø95C	FP1SGN	0061	LA5A2	A5A2	LB7ØE	B7ØE
CMP1	0091	DRGRAM	Ø986	FPAØ	0050	LA5A5	A5A5	LB738	B738
CMPMID	ØØ8F	DRIVE	CEC5	FPA1	ØØ5D	LA5AE	A5AE	LB73D	B73D
COEFCT	ØØ55	DRUNFL	Ø959	FPA2	0013	LA5C7	A5C7	LB958	B958
COEFPT	0064	DSEC	ØØED	FPCARY	ØØ5B	LA5DA	A5DA	LB95C	B95C
COMVEC	0120	DSINIT	CØØ8	FPSBYT	ØØ63	LA5E4	A5E4	LB99F	B99F
COPY	D3B9	DSKCON	D75F	FREE	CE9C	LA6Ø3	A6Ø3	LB9A2	B9A2
CPERTM	ØØ83	DSKI	D4ED	FRESPC	ØØ25	LA616	A616	LB9AC	B9AC
CPULWD	0082	DSKINI	D599	FRETOP	0021	LA61C	A61C	LB9AF	B9AF
CSRERR	0081	DSK0	D562	FRQVEC	Ø1ØF	LA61F	A61F	LB9C5	B9C5
CSSVAL	ØØC1	DSKREG	FF4Ø	GARBFL	0007	LA7D1	A7D1	LBB91	BB91
CURLIN	ØØ68	DSKVAR	CØØ6	GETCCH	ØØA5	LA7E9	A7E9	LBC14	BC14
CURPOS	ØØ88	DUSRVC	Ø95F	GETNCH	ØØ9F	LA928	A928	LBC33	BC33
CVN	CDF4	DVECØ	C44B	GIVABF	B4F4	LA951	A951	LBC35	BC35
DA	FF2Ø	DVEC1	C888	GRBLOK	ØØ86	LA974	A974	LBC5F	BC5F
DASCFL	Ø958	DVEC10	CD35	GRPRAM	ØØBC	LAC37	AC37	LBDCC	BDCC
DATEXT	C2AC	DVEC11	C8A9	HORBEG	ØØBD	LAC44	AC44	LBDD9	BDD9
DATPTR	0033	DVEC12	C6E4	HORBYT	ØØB9	LAC46	AC46	LCØØ2	CØØ2
DATTIN	ØØ35	DVEC12	CAE4	HORDEF	ØØC7	LAC6Ø	AC6Ø	LCØØC	CØØC
DATTER	0033 0031	DVEC13	C9ØC	HOREND	ØØC7	LACOD LAC73	AC73	LCØØF	CØØF
DBUFØ	0600 0700	DVEC15	CED2	IFCTR	0004 0007	LACTC	ACTC	LCØ3B	CØ3B
DBUF1	0700 0055	DVEC17	C265	IKEYIM	ØØ87	LACEF	ACEF	LCØ61	CØ61
DCBPT	ØØEE	DVEC18	CA3E	INPFLG	ØØØ9	LAD19	AD19	LCØBD	CØBD
DCDRV	ØØEB	DVEC2	C893	INT	BCEE	LAD21	AD21	LCØC2	CØC2
DCNVEC	CØØ4	DVEC7	CAE9	L8748	8748	LBØ69	BØ69	LC2EA	C2EA
DCOPC	ØØEA	DVEC8	CAF9	L88ØE	88ØE	LB143	B143	LC3Ø6	C3Ø6
DCSTA	ØØFØ	DVERFL	Ø987	L8955	8955	LB146	B146	LC3ØB	C3ØB

LCØFØ	CØFØ	LC627	C627	LC9Ø5	C9Ø5	LCC15	CC15	LDØ13	DØ13
LC109	C1Ø9	LC629	C629	LC932	C932	LCC17	CC17	LDØ15	DØ15
LC139	C139	LC631	C631	LC935	C935	LCC24	CC24	LDØ51	DØ51
LC192	C192	LC64D	C64D	LC938	C938	LCC3A	CC3A	LDØ56	DØ56
LC1F1	C1F1	LC653	C653	LC945	C945	LCC4Ø	CC4Ø	LDØ59	DØ59
LC219	C219	LC658	C658	LC96A	C96A	LCC5E	CC5E	LDØ6E	DØ6E
LC22C	C22C	LC65E	C65E	LC96E	C96E	LCC6A	CC6A	LDØ6F	DØ6F
LC238	C238	LC67D	C67D	LC973	C973	LCC6C	CC6C	LDØ7F	DØ7F
LC244	C244	LC681	C681	LC978	C978	LCC99	CC99	LDØ82	DØ82
LC24E	C24E	LC684	C684	LC97D	C97D	LCCBB	CCBB	LDØ92	DØ92
LC256	C256	LC685	C685	LC994	C994	LCCC5	CCC5	LDØA3	DØA3
LC29Ø	C29Ø	LC68C	C68C	LC99B	C99B	LCCEB	CCEB	LDØA7	DØA7
LC2BF	C2BF	LC69B	C69B	LC99D	C99D	LCDØ8	CDØ8	LDØBØ	DØBØ
LC2C1	C2C1	LC6A5	C6A5	LC9BØ	C9BØ	LCD17	CD17	LDØB9	DØB9
LC2E6	C2E6	LC6BØ	C6BØ	LC9B7	C9B7	LCD18	CD18	LDØC3	DØC3
LC31Ø	C31Ø	LC6B3	C6B3	LC9BE	C9BE	LCD1B	CD1B	LDØC9	DØC9
LC324	C324	LC6C7	C6C7	LC9DØ	C9DØ	LCD1E	CD1E	LDØDA	DØDA
LC33E	C33E	LC6D6	C6D6	LC9DF	C9DF	LCD24	CD24	LDØDF	DØDF
LC352	C352	LC6D9	C6D9	LCAØ4	CAØ4	LCD4B	CD4B	LD119	D119
LC357	C357	LC6E5	C6E5	LCAØ7	CAØ7	LCD5F	CD5F	LD11E	D11E
LC35A	C35A	LC6FC	C6FC	LCA12	CA12	LCD74	CD74	LD132	D132
LC366	C366	LC7ØF	C7ØF	LCA27	CA27	LCD81	CD81	LD143	D143
LC37B	C37B	LC71E	C71E	LCA33	CA33	LCD8E	CD8E	LD157	D157
LC3AD	C3AD	LC739	C739	LCA4F	CA4F	LCD92	CD92	LD15A	D15A
LC3B2	C3B2	LC744	C744	LCA5Ø	CA5Ø	LCD97	CD97	LD181	D181
LC3C5	C3C5	LC749	C749	LCA6C	CA6C	LCD98	CD98	LD189	D189
LC3C8	C3C8	LC755	C755	LCA7B	CA7B	LCDAØ	CDAØ	LD199	D199
LC3CF	C3CF	LC763	C763	LCA7E	CA7E	LCDAC	CDAC	LD1A8	D1A8
LC4Ø5	C4Ø5	LC76E	C76E	LCAA4	CAA4	LCDBØ	CDBØ	LD1AF	D1AF
LC4ØA	C4ØA	LC779	C779	LCAAF	CAAF	LCDB8	CDB8	LD1EF	D1EF
LC421	C421	LC784	C784	LCAB6	CAB6	LCDBC	CDBC	LD1F8	D1F8
LC429	C429	LC786	C786	LCABD	CABD	LCDCB	CDCB	LD2Ø8	D2Ø8
LC42F	C42F	LC796	C796	LCABF	CABF	LCDDØ	CDDØ	LD2ØB	D2ØB
LC43E	C43E	LC79C	C79C	LCAC6	CAC6	LCDD5	CDD5	LD212	D212
LC481	C481	LC79D	C79D	LCADA	CADA	LCDD6	CDD6	LD21B	D21B
LC48D	C48D	LC7BF	C7BF	LCAED	CAED	LCDEC	CDEC	LD222	D222
LC4BB	C4BB	LC7C8	C7C8	LCAF8	CAF8	LCE14	CE14	LD236	D236
LC4C7	C4C7	LC7E6	C7E6	LCBØ1	CBØ1	LCE19	CE19	LD249	D249
LC4E1	C4E1	LC7E8	C7E8	LCBØ6	CBØ6	LCE68	CE68	LD24F	D24F
LC4E8	C4E8	LC7EC	C7EC	LCB2E	CB2E	LCE72	CE72	LD256	D256
LC4F2	C4F2	LC7F8	C7F8	LCB31	CB31	LCEB6	CEB6	LD25F	D25F
LC5Ø4	C5Ø4	LC7FD	C7FD	LCB41	CB41	LCEBD	CEBD	LD27B	D27B
LC5Ø9	C5Ø9	LC8Ø6	C8Ø6	LCB4E	CB4E	LCEE9	CEE9	LD28C	D28C
LC514	C514	LC8Ø7	C8Ø7	LCB52	CB52	LCEEC	CEEC	LD2A4	D2A4
LC519	C519	LC8ØD	C8ØD	LCB54	CB54	LCFØ7	CFØ7	LD2A5	D2A5
LC52D	C52D	LC829	C829	LCB6B	CB6B	LCF2A	CF2A	LD2AE	D2AE
LC538	C538	LC82E	C82E	LCB76	CB76	LCF5C	CF5C	LD2CD	D2CD
LC53C	C53C	LC845	C845	LCB8B	CB8B	LCF68	CF68	LD2D2	D2D2
LC567	C567	LC866	C866	LCB8C	CB8C	LCF9B	CF9B	LD2DD	D2DD
LC586	C586	LC868	C868	LCB93	CB93	LCFB3	CFB3	LD2EF	D2EF
LC5A9	C5A9	LC881	C881	LCB97	CB97	LCFB5	CFB5	LD2FA	D2FA
LC5BA	C5BA	LC8AF	C8AF	LCBAD	CBAD	LCFBB	CFBB	LD2FC	D2FC
LC5C4	C5C4	LC8B2	C8B2	LCBB4	CBB4	LCFC1	CFC1	LD2FF	D2FF
LC5EC	C5EC	LC8C2	C8C2	LCBCØ	CBCØ	LCFDE	CFDE	LD3ØE	D3ØE
LC5F9	C5F9	LC8D1	C8D1	LCBC3	CBC3	LCFE3	CFE3	LD310	D31Ø
LC5FE	C5FE	LC8DA	C8DA	LCBCF	CBCF	LCFFA	CFFA	LD322	D322
LC602	C6Ø2	LC8F3	C8F3	LCBDF	CBDF	LDØØ4	DØØ4	LD326	D326
LC611	C611	LC8FE	C8FE	LCBE9	CBE9	LDØØD	DØØD	LD344	D344

LD357	D357	LD7Ø9	D7Ø9	OLDPTR	ØØ2D	SW3VEC	0100	VD8	ØØD8
LD35E	D35E	LD7ØB	D7ØB	OLDTXT	0029	SWI	FFFA	VD9	ØØD9
LD35F	D35F	LD7ØE	D7ØE	PIAØ	FFØØ	SWI2	FFF4	VDA	ØØDA
LD36C	D36C	LD737	D737	PIA1	FF2Ø	SWI3	FFF2	VERBEG	ØØBF
LD37E	D37E	LD742	D742	PIA2	FF4Ø	SWIVEC	0106	VERDEF	ØØC9
LD399	D399	LD743	D743	PLYTMR	ØØE3	SYNCLN	0092	VEREND	ØØC5
LD3AØ	D3AØ	LD75A	D75A	PMODE	ØØB6	SYNCOMMA		VERIFY	D74E
LD3AD	D3AD	LD765	D765	POTVAL	Ø15A	TEMPO	ØØE2	VIDRAM	0400
LD3CE	D3CE	LD77E	D77E	PRTDEV	ØØ6E	TEMPPT	ØØØB	VOLHI	ØØDF
LD3DC	D3DC	LD792	D792	RDYTMR	Ø985	TEMPTR	ØØØF	VOLLOW	ØØEØ
LD41E	D41E	LD7AØ	D7AØ	RELFLG	ØØØA	TIMOUT	ØØE7	WCOLOR	ØØB4
LD423	D423	LD7B1	D7B1	RELPTR	ØØ3D	TIMVAL	Ø112	WFATVL	Ø97A
LD42C	D42C	LD7B8	D7B8	RENAME	DØ1B	TINPTR	ØØ2F	WRITE	DØ66
LD44E	D44E	LD7DØ	D7DØ	RESETV	FFFE	TMPLOC	0003	XBWMST	8ØCØ
LD455	D455	LD7D1	D7D1	RESSGN	0062	TMPSTK	ØØDC	XVEC15	8846
LD45D	D45D	LD7D3	D7D3	RNBFAD	Ø948	TMPTR1	0011	XVEC17	88FØ
LD45F	D45F	LD7DF	D7DF	ROMPAK	CØØØ	TOPRAM	0074	XVEC18	829C
LD472	D472	LD7FØ	D7FØ	RSET	D1Ø1	TRCFLG	ØØAF	XVEC3	8273
LD476	D476	LD7F3	D7F3	RSTFLG	0071	TRELFL	ØØ3F	XVEC4	8CF1
LD47C	D47C	LD7F8	D7F8	RSTVEC	0072	TXTTAB	0019	XVEC8	8286
LD482	D482	LD7FB	D7FB	RVECØ	Ø15E	UNLOAD	D233	XVEC9	8E9Ø
LD486	D486	LD82A	D82A	RVEC1	Ø161	USRADR	ØØBØ	ZERO	ØØ8A
LD4BA	D4BA	LD82C	D82C	RVEC10	Ø17C	USRJMP	Ø112		
LD4C4	D4C4	LD85B	D85B	RVEC11	Ø17F	V4Ø	0040		
LD4D4	D4D4	LD863	D863	RVEC12	Ø182	V41	0041		
LD4EA	D4EA	LD86B	D86B	RVEC13	Ø185	V42	0042		
LD5Ø8	D5Ø8	LD875	D875	RVEC14	Ø188	V43	0043		
LD519	D519	LD877	D877	RVEC15	Ø18B	V44	0044		
LD51C	D51C	LD881	D881	RVEC16	Ø18E	V45	0045		
LD524	D524	LD88B	D88B	RVEC17	Ø191	V46	0046		
LD527	D527	LD895	D895	RVEC18	Ø194	V47	0047		
LD54A	D54A	LD89D	D89D	RVEC19	Ø197	V48	0048		
LD54D	D54D	LD8AE	D8AE	RVEC2	Ø164	V4A	ØØ4A		
LD553	D553	LD8CD	D8CD	RVEC2Ø	Ø19A	V4B	ØØ4B		
LD577	D577	LDFØ9	DFØ9	RVEC21	Ø19D	V4D	ØØ4D		
LD58F	D58F	LDF36	DF36	RVEC22	Ø1AØ	V973	Ø973		
LD5B2	D5B2	LINBUF	Ø2DC	RVEC23	Ø1A3	V974	Ø974		
LD5B9	D5B9	LINHDR	Ø2DA	RVEC24	Ø1A6	V976	Ø976		
LD5C1	D5C1	LIST	B764	RVEC3	Ø167	V977	Ø977		
LD5C3	D5C3	LOAD	CA48	RVEC4	Ø16A	V978	Ø978		
LD5C4	D5C4	LOC	CE1Ø	RVEC5	Ø16D	VAB	ØØAB		
LD5CE	D5CE	LOF	CE37	RVEC6	0170	VAC	ØØAC		
LD6Ø6	D6Ø6	LPTBTD	ØØ95	RVEC7	Ø173	VAD	ØØAD		
LD612	D612	LPTCFW	ØØ99	RVEC8	Ø176	VAE	ØØAE		
LD62Ø	D62Ø	LPTLCF	ØØ9A	RVEC9	Ø179	VALTMP	ØØØ6		
LD649	D649	LPTLND	ØØ97	RVSEED	Ø115	VARDES	ØØ3B		
LD64F	D64F	LPTPOS	ØØ9C	SAMREG	FFCØ	VARNAM	ØØ37		
LD66F	D66F	LPTWID	ØØ9B	SAVE	C9EØ	VARPTR	ØØ39		
LD672	D672	LSET	D102	SCALE	ØØE9	VARTAB	ØØ1B		
LD688	D688	LSTTXT	ØØ66	SETFLG	ØØC2	VCB	ØØCB		
LD691	D691	MEMSIZ	0027	SNDDUR	ØØ8D	VCD	ØØCD		
LD69A	D69A	MERGE	CA39	SNDTON	ØØ8C	VCF	ØØCF		
LD6C2	D6C2	MKN	CEØ2	STRBUF	Ø3D7	VD1	ØØD1		
LD6C8	D6C8	NMI	FFFC	STRDES	ØØ56	VD3	ØØD3		
LD6D4	D6D4	NMIFLG	Ø982	STRINOUT	B99C	VD4	ØØD4		
LD6F2	D6F2	NMIVEC	0109	STRSTK	Ø1A9	VD5	ØØD5		
LD6FB	D6FB	NOTELN	ØØE1	STRTAB	0023	VD6	ØØD6		
LD7Ø1	D7Ø1	OCTAVE	ØØDE	SW2VEC	0103	VD7	ØØD7		

ALLCOL	ØØB5	DCTRK	ØØEC	DXCVEC	CE2E	L8748	8748	LASTPT	ØØØD
ANGLE	ØØE8	DEBVAL	Ø11B	DXIVEC	CE56	L88ØE	88ØE	LBØØC	вøøс
ARYDIS	0008	DEFDRV	Ø95A	ENDCHR	0002	L8955	8955	LBØ1E	BØ1E
ARYEND	ØØ1F	DEFEXT	C291	ENDFLG	0000	L8C1B	8C1B	LBØ69	BØ69
ARYTAB	ØØ1D	DEVCFW	ØØ6A	ENDGRP	ØØB7	L95AC	95AC	LB143	B143
ATTCTR	Ø988	DEVLCF	ØØ6B	EVALEXPB		L962E	962E	LB146	B146
BACKUP	D175	DEVNUM	ØØ6F	EXECJP	ØØ9D	L965Ø	965Ø	LB148	B148
BAKCOL	ØØB3	DEVPOS	ØØ6C	EXPJMP	Ø11D	L96CB	96CB	LB156	B156
BASEXT	C28E	DEVWID	ØØ6D	FATBLØ	Ø8ØØ	L96EC	96EC	LB166	B166
BAWMST	AØE8	DEXTBF	Ø954	FATBL1	Ø84A	L975F	975F	LB244	B244
BEGGRP	ØØBA	DFFLEN	Ø97C	FATBL2	Ø894	L9FB5	9FB5	LB262	B262
BINEXT	C297	DFLBUF	Ø989	FATBL3	Ø8DE	LAØE2	AØE2	LB26F	B26F
BINVAL	ØØ2B	DFLTYP	Ø957	FCBACT	Ø95B	LA171	A171	LB277	B277
BLKCNT	ØØ25 ØØ94	DIMFLG	ØØØ5	FCBADR	Ø94A	LA171	A176	LB2CE	B2CE
BLKLEN	ØØ7D	DIR	CBCF	FCBTMP	ØØF1	LA282	A282	LB357	B357
BLKTYP	ØØ7C	DIRQSV	D7BC	FCBV1	Ø928	LA35F	A35F	LB3E6	B3E6
BOTSTK	0017	DKWMST	CØD4	FDCREG	FF48	LA37C	A37C	LB44A	B44A
BROMHK	AA1A	DLBAUD	ØØE6	FIELD	CFEØ	LA3ED	A3ED	LB4F3	B4F3
CASBUF	Ø1DA	DLODFL	Ø95D	FILES	DØ8Ø	LA3FB	A3FB	LB5ØF	B5ØF
CASFLG	Ø11A	DMRGFL	Ø95E	FILSTA	ØØ78	LA4Ø6	A4Ø6	LB516	B516
CBTPHA	ØØ84	DNAMBF	Ø94C	FIRQ	FFF6	LA426	A426	LB654	B654
CBUFAD	ØØ7E	DNMISV	D7AE	FORCOL	ØØB2	LA429	A429	LB657	B657
CCKSUM	ØØ8Ø	DNMIVC	Ø983	FPØEXP	ØØ4F	LA42D	A42D	LB659	B659
CFNBUF	Ø1D1	DOSBAS	CØØØ	FPØSGN	ØØ54	LA549	A549	LB69B	B69B
	0001	DOSBUF	2600	FP1EXP	ØØ5C	LAS9A	A59A	LB6A4	B6A4
CHARAC									
CHARAD	ØØA6	DOTVAL	ØØE5	FP1SGN	ØØ61	LA5A2	A5A2	LB7ØE	B7ØE
CHGFLG	ØØDB	DRØTRK	Ø97E	FPAØ	0050	LA5A5	A5A5	LB738	B738
CINBFL	0070	DRESFL	Ø95C	FPA1	ØØ5D	LA5AE	A5AE	LB73D	B73D
CINCTR	0079	DRGRAM	Ø986	FPA2	0013	LA5C7	A5C7	LB958	B958
CINPTR	ØØ7A	DRIVE	CDE9	FPCARY	ØØ5B	LA5DA	A5DA	LB95C	B95C
CLSTSN	ØØ85	DRUNFL	Ø959	FPSBYT	ØØ63	LA5E4	A5E4	LB99F	B99F
CMPØ	0090	DSEC	ØØED	FREE	CDCØ	LA6Ø3	A6Ø3	LB9A2	B9A2
CMP1	0091	DSKCON	D66C	FRESPC	ØØ25	LA616	A616	LB9AC	B9AC
CMPMID	ØØ8F	DSKI	D3FF	FRETOP	0021	LA61C	A61C	LB9AF	B9AF
COEFCT	ØØ55	DSKINI	D4AB	FRQVEC	Ø1ØF	LA61F	A61F	LB9C5	B9C5
COEFPT	ØØ64	DSK0	D474	GARBFL	0007	LA7D1	A7D1	LBB91	BB91
COMVEC	Ø12Ø	DSKREG	FF4Ø	GETCCH	ØØA5	LA7E9	A7E9	LBC14	BC14
COPY	D2CC	DSKVAR	CØØ6	GETNCH	ØØ9F	LA928	A928	LBC33	BC33
									BC35
CPERTM	0083	DUSRVC	Ø95F	GIVABF	B4F4	LA951	A951	LBC35	
CPULWD	0082	DVECØ	C426	GRBLOK	ØØ86	LA974	A974	LBC5F	BC5F
CSRERR	0081	DVEC1	C838	GRPRAM	ØØBC	LAC37	AC37	LBDCC	BDCC
CSSVAL	ØØC1	DVEC10	CC5B	HORBEG	ØØBD	LAC44	AC44	LBDD9	BDD9
CURLIN	ØØ68	DVEC11	C859	HORBYT	ØØB9	LAC46	AC46	LCØØ2	CØØ2
CURPOS	ØØ88	DVEC12	C6B7	HORDEF	ØØC7	LAC6Ø	AC6Ø	LCØØ8	CØØ8
CVN	CD1A	DVEC13	CA36	HOREND	ØØC3	LAC73	AC73	LCØØB	CØØB
DA	FF2Ø	DVEC14	C86Ø	IFCTR	0004	LAC7C	AC7C	LCØ37	CØ37
DASCFL	Ø958	DVEC15	CDF6	IKEYIM	ØØ87	LACEF	ACEF	LCØ57	CØ57
DATEXT	C294	DVEC17	C24D	INPFLG	ØØØ9	LAD19	AD19	LCØB3	CØB3
DATPTR	ØØ33	DVEC18	C99Ø	INT	BCEE	LAD21	AD21	LCØDD	CØDD
DATTMP	ØØ35	DVEC2	C843	IRQ	FFF8	LAD21	AD33	LCØF6	CØF6
DATTXT	0031	DVEC22	C29A	IRQVEC	Ø1ØC	LADGE	ADGE	LC126	C126
DBUFØ	0600	DVEC3	CB4A	KEYBUF	Ø152	LADC6	ADC6	LC17F	C17F
DBUF1	0700	DVEC4	C58F	KILL	C6C2	LADD4	ADD4	LC1DB	C1DB
DCBPT	ØØEE	DVEC5	C818	L813C	813C	LADEB	ADEB	LC2Ø1	C2Ø1
DCDRV	ØØEB	DVEC6	C81B	L8168	8168	LAE15	AE15	LC214	C214
DCNVEC	CØØ4	DVEC7	CA3B	L8311	8311	LAF9A	AF9A	LC22Ø	C22Ø
DCOPC	ØØEA	DVEC8	CA4B	L8316	8316	LAFA4	AFA4	LC22C	C22C
DCSTA	ØØFØ	DVERFL	Ø987	L836C	836C	LAFB1	AFB1	LC236	C236

LC23E	C23E	LC657	C657	LC956	C956	LCCB4	CCB4	LDØ7B	DØ7B
LC278	C278	LC658	C658	LC959	C959	LCCB8	CCB8	LDØ7E	DØ7E
LC2A7	C2A7	LC65F	C65F	LC964	C964	LCCBD	CCBD	LDØA5	DØA5
LC2A9	C2A9	LC66E	C66E	LC979	C979	LCCBE	CCBE	LDØBØ	DØBØ
LC2CC	C2CC	LC678	C678	LC985	C985	LCCC6	CCC6	LDØCØ	DØCØ
LC2DØ	C2DØ	LC683	C683	LC9A1	C9A1	LCCD2	CCD2	LDØCF	DØCF
LC2E8	C2E8	LC686	C686	LC9A2	C9A2	LCCD6	CCD6	LD102	D1Ø2
LC2ED	C2ED	LC69A	C69A	LC9BE	C9BE	LCCDE	CCDE	LD1ØB	D1ØB
LC2F2	C2F2	LC6A9	C6A9	LC9CD	C9CD	LCCE2	CCE2	LD11B	D11B
LC3Ø6	C3Ø6	LC6AC	C6AC	LC9DØ	C9DØ	LCCF1	CCF1	LD11E	D11E
LC32Ø	C32Ø	LC6B8	C6B8	LC9F6	C9F6	LCCF6	CCF6	LD125	D125
LC334	C334	LC6CF	C6CF	LCAØ1	CAØ1	LCCFB	CCFB	LD12E	D12E
LC339	C339	LC6E2	C6E2	LCAØ8	CAØ8	LCCFC	CCFC	LD135	D135
LC33C	C33C	LC6F1	C6F1	LCAØF	CAØF	LCD12	CD12	LD149	D149
LC348	C348	LC7ØC	C7ØC	LCA11	CA11	LCD3A	CD3A	LD15C	D15C
LC35D	C35D	LC714	C714	LCA18	CA18	LCD3D	CD3D	LD162	D162
LC38F	C38F	LC719	C719	LCA2C	CA2C	LCD8C	CD8C	LD169	D169
LC394	C394	LC725	C725	LCA3F	CA3F	LCD96	CD96	LD172	D172
LC3A7	C3A7	LC733	C733	LCA4A	CA4A	LCDDA	CDDA	LD18E	D18E
LC3AA	C3AA	LC73E	C73E	LCA53	CA53	LCDE1	CDE1	LD19F	D19F
LC3B1	C3B1	LC749	C749	LCA58	CA58	LCEØD	CEØD	LD1B7	D1B7
LC3EØ	C3EØ	LC754	C754	LCA8Ø	CA8Ø	LCE10	CE1Ø	LD1B8	D1B8
LC3E5	C3E5	LC756	C756	LCA8E	CA8E	LCE2B	CE2B	LD1C1	D1C1
LC3FC	C3FC	LC766	C766	LCA9B	CA9B	LCE4E	CE4E	LD1EØ	D1EØ
LC4Ø4	C4Ø4	LC76C	C76C	LCA9F	CA9F	LCE8Ø	CE8Ø	LD1E5	D1E5
LC4ØA	C4ØA	LC76D	C76D	LCAA1	CAA1	LCE8C	CE8C	LD1FØ	D1FØ
LC419	C419	LC78F	C78F	LCAB8	CAB8	LCEBF	CEBF	LD2Ø2	D2Ø2
LC45C	C45C	LC798	C798	LCAC3	CAC3	LCED7	CED7	LD2ØD	D2ØD
LC468	C468	LC7B6	C7B6	LCAD8	CAD8	LCED9	CED9	LD211	D211
LC496	C496	LC7B8	C7B8	LCAD9	CAD9	LCEDF	CEDF	LD212	D212
LC4A2	C4A2	LC7BC	C7BC	LCAEØ	CAEØ	LCEE5	CEE5	LD221	D221
LC4BC	C4BC	LC7C8	C7C8	LCAE2	CAE2	LCFØ2	CFØ2	LD223	D223
LC4C2	C4C2	LC7CD	C7CD	LCAEE	CAEE	LCFØ7	CFØ7	LD235	D235
LC4CC	C4CC	LC7D6	C7D6	LCAF1	CAF1	LCF1E	CF1E	LD239	D239
LC4DE	C4DE	LC7D7	C7D7	LCAFD	CAFD	LCF28	CF28	LD257	D257
LC4E3	C4E3	LC7DD	C7DD	LCBØD	CBØD	LCF31	CF31	LD26A	D26A
LC4EC	C4EC	LC7F9	C7F9	LCB17	CB17	LCF37	CF37	LD271	D271
LC5ØØ	C5ØØ	LC7FE	C7FE	LCB43	CB43	LCF39	CF39	LD272	D272
LC5ØB	C5ØB	LC815	C815	LCB45	CB45	LCF75	CF75	LD27F	D27F
LC5ØF	C5ØF	LC836	C836	LCB52	CB52	LCF7A	CF7A	LD291	D291
LC53A	C53A	LC85F	C85F	LCB68	CB68	LCF7D	CF7D	LD2AC	D2AC
LC559	C559	LC884	C884	LCB6E	CB6E	LCF92	CF92	LD2B3	D2B3
LC57C	C57C	LC887	C887	LCB8C	CB8C	LCF93	CF93	LD2CØ	D2CØ
LC58D	C58D	LC88A	C88A	LCB98	CB98	LCFA3	CFA3	LD2E1	D2E1
LC597	C597	LC897	C897	LCB9A	CB9A	LCFA6	CFA6	LD2EE	D2EE
LC5BF	C5BF	LC8BC	C8BC	LCBC3	CBC3	LCFB6	CFB6	LD33Ø	D33Ø
LC5CC	C5CC	LC8CØ LC8C5	C8CØ	LCBC4	CBC4	LCFC7	CFC7	LD335	D335
LC5D1 LC5D5	C5D1 C5D5	LC8CA	C8C5 C8CA	LCBE1 LCBEB	CBE1 CBEB	LCFCB LCFD4	CFCB CFD4	LD33E LD360	D33E D36Ø
		LC8CF							
LC5E4 LC5FA	C5E4 C5FA	LC8E6	C8CF C8E6	LCC11 LCC2E	CC11 CC2E	LCFDD LCFE7	CFDD CFE7	LD367 LD36F	D367 D36F
LC5FA LC5FC	C5FC	LC8ED	C8ED	LCC3D	CC3D	LCFE7 LCFED	CFE7	LD36F LD371	D36F D371
LC6Ø4	C6Ø4	LC8EF	C8EF	LCC44	CC44	LCFFE	CFFE	LD371 LD384	D371
LC626	C626	LC9Ø2	C9Ø2	LCC44	CC44	LDØØ3	DØØ3	LD388	D388
LC62B	C62B	LC902 LC909	C902	LCC71	CC71	LDØØ3	DØ3D	LD38E	D38E
LC62B	C631	LC910	C909 C910	LCC85	CC85	LDØ3D LDØ42	DØ42	LD394	D394
LC650	C65Ø	LC922	C910	LCC9A	CC9A	LDØ42 LDØ56	DØ42	LD394 LD398	D394 D398
LC658	C654	LC922	C922	LCC47	CCA7	LDØ50 LDØ67	DØ50	LD398	D396
L0034	0054	L0331	0331	LUCAT	CCAI	בטשט/	וטשט	LDJCC	D300

LD3D6	D3D6	LD768	D768	RVEC13	Ø185	V42	0042
LD3E6	D3E6	LD77Ø	D77Ø	RVEC14	Ø188	V43	0043
LD3FC	D3FC	LD778	D778	RVEC15	Ø18B	V44	0044
LD41A	D41A	LD782	D782	RVEC16	Ø18E	V45	0045
LD42B	D42B	LD784	D784	RVEC17	Ø191	V46	0046
LD42E	D42E	LD78E	D78E	RVEC18	Ø194	V47	0047
LD436	D436	LD798	D798	RVEC19	Ø197	V48	0048
LD439	D439	LD7A2	D7A2	RVEC2	Ø164	V4A	ØØ4A
LD45C	D45C	LD7AA	D7AA	RVEC2Ø	Ø19A	V4B	ØØ4B
LD45F	D45F	LD7BB	D7BB	RVEC21	Ø19D	V4D	ØØ4D
LD465	D465	LD7DA	D7DA	RVEC22	Ø1AØ	V973	Ø973
LD489	D489	LINBUF	Ø2DC	RVEC23	Ø1A3	V974	Ø974
LD4A1	D4A1	LINHDR	Ø2DA	RVEC24	Ø1A6	V976	Ø976
LD4C4	D4C4	LIST	B764	RVEC3	Ø167	V977	Ø977
LD4CB	D4CB	LOAD	C99A	RVEC4	Ø16A	V978	Ø978
LD4D3	D4D3	LOC	CD36	RVEC5	Ø16D	VAB	ØØAB
LD4D5	D4D5	LOF	CD5B	RVEC6	0170	VAC	ØØAC
LD4D6	D4D6	LPTBTD	ØØ95	RVEC7	Ø173	VAD	ØØAD
LD4EØ	D4EØ	LPTCFW	ØØ99	RVEC8	Ø176	VAE	ØØAE
LD519	D519	LPTLCF	ØØ9A	RVEC9	Ø179	VALTMP	0006
LD525	D525	LPTLND	ØØ97	RVSEED	Ø115	VARDES	ØØ3B
	D523					VARDES	
LD533		LPTPOS	ØØ9C	SAMREG	FFCØ		ØØ37
LD55C	D55C	LPTWID	ØØ9B	SAVE	C932	VARPTR	0039
LD562	D562	LSET	DØ26	SCALE	ØØE9	VARTAB	ØØ1B
LD582	D582	LSTTXT	ØØ66	SETFLG	ØØC2	VCB	ØØCB
LD585	D585	MEMSIZ	ØØ27	SNDDUR	ØØ8D	VCD	ØØCD
LD59B	D59B	MERGE	C98B	SNDTON	ØØ8C	VCF	ØØCF
LD5AD	D5AD	MKN	CD28	STRBUF	Ø3D7	VD1	ØØD1
LD5D5	D5D5	NMI	FFFC	STRDES	ØØ56	VD3	ØØD3
LD5DB	D5DB	NMIFLG	Ø982	STRINOUT	B99C	VD4	ØØD4
LD5FF	D5FF	NMIVEC	0109	STRSTK	Ø1A9	VD5	ØØD5
LD6Ø8	D6Ø8	NOTELN	ØØE1	STRTAB	ØØ23	VD6	ØØD6
LD6ØE	D6ØE	OCTAVE	ØØDE	SW2VEC	Ø1Ø3	VD7	ØØD7
LD616	D616	OLDPTR	ØØ2D	SW3VEC	0100	VD8	ØØD8
LD618	D618	OLDTXT	0029	SWI	FFFA	VD9	ØØD9
LD61B		PIAØ		SWI2	FFF4	VDA	ØØDA
	D61B		FFØØ				
LD644	D644	PIA1	FF2Ø	SWI3	FFF2	VERBEG	ØØBF
LD64F	D64F	PIA2	FF4Ø	SWIVEC	0106	VERDEF	ØØC9
LD65Ø	D65Ø	PLYTMR	ØØE3	SYNCLN	ØØ92	VEREND	ØØC5
LD667	D667	PMODE	ØØB6	SYNCOMMA	B26D	VERIFY	D65B
LD672	D672	POTVAL	Ø15A	TEMP0	ØØE2	VIDRAM	0400
LD68B	D68B	PRTDEV	ØØ6E	TEMPPT	ØØØB	VOLHI	ØØDF
LD69F	D69F	RDYTMR	Ø985	TEMPTR	ØØØF	VOLLOW	ØØEØ
LD6AD						WCOLOR	ØØB4
	D6AD	RELFLG	ØØØA	TIMOUT	ØØE7		
LD6BE	D6BE	RELPTR	ØØ3D	TIMVAL	Ø112	WFATVL	Ø97A
LD6C5	D6C5	RENAME	CF3F	TINPTR	ØØ2F	WRITE	CF8A
LD6D4	D6D4	RESETV	FFFE	TMPLOC	ØØØ3	XBWMST	8ØCØ
LD6DD	D6DD	RESSGN	ØØ62	TMPSTK	ØØDC	XVEC15	8846
LD6DE	D6DE	RNBFAD	Ø948	TMPTR1	0011	XVEC17	88FØ
LD6EØ	D6EØ	ROMPAK	CØØØ	TOPRAM	0074	XVEC18	829C
							8273
LD6EC	D6EC	RSET	DØ25	TRCFLG	ØØAF	XVEC3	
LD6FD	D6FD	RSTFLG	0071	TRELFL	ØØ3F	XVEC4	8CF1
LD7ØØ	D7ØØ	RSTVEC	0072	TXTTAB	0019	XVEC8	8286
LD7Ø5	D7Ø5	RVECØ	Ø15E	UNLOAD	D146	XVEC9	8E9Ø
LD7Ø7	D7Ø7	RVEC1	Ø161	USRADR	ØØBØ	ZER0	ØØ8A
LD7Ø8	D7Ø8	RVEC10	Ø17C	USRJMP	Ø112		
LD737	D737	RVEC11	Ø17F	V4Ø	0040		
LD737	D737	RVEC12	Ø182	V45 V41	0041		
LU/ 33	טוט	NVLUIL	DIOL	A T	דדעמ		

MODIFIED REGISTERS	1.1 ADDRESS	1.0 ADDRESS	DESCRIPTION
B,X,Y,U	C48D	C468	OPEN DISK FILE - enter with the mode (I,O,D,R) in ACCA, file number in ACCB, filename.ext in DNAMBF, and DFLTYP, DASCFL & DFFLEN initialized. An FCB will be opened and initialized but any errors will cause control to be returned to BASIC.
A,B,X,U	C52D	C5ØØ	INITIALIZE FCB FOR INPUT - enter with the address of the desired FCB in FCBTMP and valid directory information in V973-V978. The FCB will be properly initialized and the directory will be loaded with the number of bytes in the last sector of the file.
A,B,X,U	C538	C5ØB	INITITALIZE FCB - this is the same as initialize FCB for input except that the number of bytes in the last sector is not written into the directory.
A,B,X,U	C567	C53A	SET UP DIRECTORY ENTRY - scan the directory for the first unused entry and open a file for the information contained in DNAMBF, DFLTYP, and DASCFL. The first unused directory entry will be allocated to the file as will the first unused granule.
A	C5C4	C597	DISK CONSOLE IN - get a byte from the already OPENed disk file specified by DEVNUM. Return the byte in ACCA.
ALL	C68C	C65F	SEARCH DIRECTORY - search the directory for the filename and extension located in DNAMBF. Set the variables V973-V978 accordingly.
ALL	C6F5	C6C8	KILL FILE - kill the file whose name is in DNAMBF.
A,B,X,U	C7ØF	C6E2	FREE FILE GRANULES - enter with the number of the first granule in a file in ACCB. Free (set to \$FF) all of the granules in that file and save the new FAT on the disk.
X	C744	C714	SET X TO FILE BUFFER - enter with ACCB containing the file number. Return with X pointing to the correct FCB and the flags set according to the file type.
Х	C755	C725	SET X TO FAT - point X to the FAT RAM image for the drive number stored in DCDRV.
А,В	C763	C733	CONVERT GRANULE TO TRACK & SECTOR - enter with X pointing to an FCB. The current granule number (FCBCGR) will be converted to the equivalent track and sector numbers in DCTRK & DSEC.
A,B	C779	C749	MULTIPLY ACCD BY NINE - multiply the value in ACCD by nine.
А,В	C784	C754	CONVERT SECTORS TO GRANULE - enter with a total number of sectors in ACCD. Convert this number into

			the number of complete granules (0-67) contained in that many sectors and return the count in ACCD.
А,В,Х	C79D	C76D	READ FAT DATA - load the RAM image of the FAT with data from the disk. Data will not be loaded into the RAM image if any disk files are OPEN.
A,B,X	C7BF	C78F	FIND FREE GRANULE - find the first free granule. Enter with the granule at which to start the search in ACCB. The found granule is marked with a \$CØ to indicate that it is the last granule in the file and the number of the granule is returned in ACCA.
В,Х	C8Ø7	C7D7	FILE OPEN CHECK - check all active files to make sure a file is not already OPEN. Enter with ACCA containing a file type to disable the AO error for that file type.
A,B,X,U	C935	C887	GET FILENAME.EXT:DRIVE FROM BASIC - get the file- name extension and drive number from a BASIC input line.
A,B,X,U	CD1E	CC44	GET GRANULE COUNT - enter with the granule number of the first granule in a file. The number of whole granules in that file will be returned in ACCA. ACCB will contain the data from the last granule in the file.
А,В,Х	CEA8	CDCC	GET FREE GRANULE COUNT - enter with a drive number (0-3) in ACCB, return the number of free granules in floating point accumulator 0.
NONE	D75F	D66C	DSKCON - universal disk I/O routine. A detailed explanation is available in the Color Computer disk user's manual.
А,В,Х	D7B8	D6C5	RESTORE HEAD TO TRACK ZERO - restore the head for the drive in DCDRV to track zero. Return DCSTA = \$10 if there is a SEEK error.
ALL	D7F8	D7Ø5	READ ONE SECTOR - read one sector as specified by the DSKCON parameters (DSEC,DCTRK,DCDRV) and store the data at the address in DCBPT.
ALL	D7FB	D7Ø8	WRITE ONE SECTOR - write one sector as specified by the DSKCON parameters (DSEC,DCTRK,DCDRV) and get the data to go on the disk from the address in DCBPT.

1.1		1.Ø		
START	END	START	END	DESCRIPTION
CØØØ	CØØ1	CØØØ	CØØ1	DISK BASIC ROM IDENTIFIER
CØØ4	CØØB	CØØ4	CØØ7	INDIRECT JUMP TABLE
C1Ø9	C112	CØF6	CØFF	COMMAND INTERPRETATION TABLE ROM IMAGE
C113	C138	C100	C125	RAM HOOKS ROM IMAGE
C139	C191	C126	C17E	COPYRIGHT MESSAGES
C192	C1FØ	C17F	C1DA	PRIMARY RESERVED WORD TABLE
C1F1	C218	C1DB	C2ØØ	PRIMARY RESERVED WORD DISPATCH TABLE
C219	C22B	C2Ø1	C213	SECONDARY RESERVED WORD TABLE
C22C	C237	C214	C21F	SECONDARY RESERVED WORD DISPATCH TABLE
C29Ø	C2A5	C278	C28D	ERROR MESSAGES
C2A6	C2B1	C28E	C299	DISK FILE EXTENSION MESSAGES
D35F	D398	D272	D2AB	INSERT SOURCE/DESTINATION MESSAGES
D6D4	D6EB	D5E7	D5FE	DISK FORMATTING DATA TABLE
D895	D89C	D7A2	D7A9	DSKCON OPERATION CODE JUMP VECTORS
D89D	D8AØ	D7AA	D7AD	DSKREG MASKS FOR DRIVE SELECT

There are times when it is useful to cause an error message to be printed to the screen in the same manner that BASIC prints its error messages. The following table is provided to give the user the DISK BASIC entry points which will cause error messages to be printed to the screen. A JMP to one of these error message routines will cause the two letter short form error message to be printed on the screen and a pseudo warm start into BASIC will be taken. The pseudo warm start will reset the stack, the string stack and the continue pointer and jump to BASIC s direct mode (OK).

## DISK BASIC ERROR JUMPS

		1.1	1.0	1.1	1.0	
NAME	NBR	LABEL	LABEL	ADDR	ADDR	DESCRIPTION
ΙΟ	20	LD7Ø9	LD616	D7Ø9	D616	INPUT/OUTPUT
ΙE	23	LC352	LC334	C352	C334	INPUT PAST END OF FILE
NE	26	LC6E5	LC6B8	C6E5	C6B8	FILE NOT FOUND
BR	27	LC3ØB	LC2ED	C3ØB	C2ED	BAD RECORD
DF	28	LC7F8	LC7C8	C7F8	C7C8	DISK FULL
OB	29	LC5Ø4	LC4DE	C5Ø4	C4DE	OUT OF BUFFER SPACE
FN	31	LC978	LC8CA	C978	C8CA	BAD FILE NAME
FS	32	LC653	LC626	C653	C626	BAD FILE STRUCTURE
F0	34	LDØDA	LCFFE	DØDA	CFFE	FIELD OVERFLOW
SE	35	LD119	LDØ3D	D119	DØ3D	SET TO NON-FIELDED STRING
۷F	36	LD74A	LD657	D74A	D657	VERIFICATION ERROR
ER	37	LCDCB	LCFF1	CDCB	CFF1	WRITE OR INPUT PAST END OF RECORD

There are no unconditional jump entry points for error #30 (WP -Write Proctected) or error #33 (AE - file Already Exists). These errors may be generated by loading a value equal to 2\*(error number) into ACCB and then JMPing to AC46.

## DISPLAY CHARACTER SET

HEX	VALUE		HEX VALUE			HEX VALUE		
Non-		CHARACTER	Non-		CHARACTER	Non-		CHARACTER
Inverted	Inverted		Inverted	Inverted		Inverted	Inverted	
ØØ	40	@	18	58	Х	3Ø	40	Ø
Ø1	41	Α	19	59	Y	31	41	1
Ø2	42	В	1 A	5 A	Z	32	42	2
Ø3	43	С	1B	5B	[	33	43	3
Ø4	44	D	1 C	5 C	\	34	44	4
Ø5	45	E	1 D	5 D	]	35	45	5
Ø6	46	F	1 E	5 E	<b>^</b>	36	46	6
Ø7	47	G	1 F	5 F	<del>(</del>	37	47	7
Ø8	48	Н	20	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	4 C	<
ØD	4 D	М	25	65	%	3D	4 D	=
ØE	4 E	N	26	66	&	3E	4 E	>
ØF	4 F	0	27	67	•	3F	4 F	?
10	50	Р	28	68	(			
11	51	Q	29	69	)			
12	52	R	2A	6 A	*			
13	53	S	2B	6B	+			
14	54	T	2C	6C	,			
15	55	U	2 D	6 D	-			
16	56	V	2 E	6 E				
17	57	W	2F	6 F	/			